AWS Batch
User Guide
# Table of Contents

What Is AWS Batch? ................................................................................................................................. 1
Components of AWS Batch .......................................................................................................................... 1
Jobs ......................................................................................................................................................... 1
Job Definitions ...................................................................................................................................... 1
Job Queues ........................................................................................................................................... 1
Compute Environment .............................................................................................................................. 1
Getting Started ..................................................................................................................................... 2
Setting Up .............................................................................................................................................. 3
Sign Up for AWS .................................................................................................................................... 3
Create an IAM User ................................................................................................................................. 3
Create IAM Roles for your Compute Environments and Container Instances ...................................... 5
Create a Key Pair .................................................................................................................................. 5
Create a Virtual Private Cloud .............................................................................................................. 7
Create a Security Group ......................................................................................................................... 7
Install the AWS CLI ............................................................................................................................... 8
Getting Started .................................................................................................................................... 9
Step 1: Define a Job ............................................................................................................................... 9
Step 2: Configure the Compute Environment and Job Queue .............................................................. 11
Jobs ...................................................................................................................................................... 14
Submitting a Job .................................................................................................................................... 14
Job States .............................................................................................................................................. 16
Job Environment Variables .................................................................................................................... 17
Automated Job Retries ............................................................................................................................ 18
Job Dependencies ................................................................................................................................. 19
Job Timeouts ........................................................................................................................................ 19
Array Jobs .......................................................................................................................................... 20
Example Array Job Workflow ............................................................................................................... 22
Tutorial: Using Array Job Index ............................................................................................................. 24
Multi-node Parallel Jobs ......................................................................................................................... 28
Environment Variables .......................................................................................................................... 29
Node Groups ...................................................................................................................................... 29
Job Lifecycle ......................................................................................................................................... 30
Compute Environment Considerations ................................................................................................. 30
GPU Jobs ............................................................................................................................................ 31
Job definitions ....................................................................................................................................... 32
Creating a job definition .......................................................................................................................... 32
Creating a multi-node parallel job definition .......................................................................................... 36
Job definition template ............................................................................................................................ 39
Job definition parameters ......................................................................................................................... 43
Job definition name ................................................................................................................................ 43
Type ..................................................................................................................................................... 43
Parameters .......................................................................................................................................... 44
Platform capabilities ............................................................................................................................... 44
Propagate tags ...................................................................................................................................... 44
Container properties .............................................................................................................................. 45
Node properties .................................................................................................................................... 58
Retry strategy ....................................................................................................................................... 59
Tags ...................................................................................................................................................... 61
Timeout ................................................................................................................................................ 61
Using the awslogs log driver ............................................................................................................... 61
Available awslogs log driver options .................................................................................................... 62
Specifying a log configuration in your job definition ............................................................................... 63
Specifying sensitive data ......................................................................................................................... 64
Using Secrets Manager ......................................................................................................................... 65
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM Policies, Roles, and Permissions</td>
<td>114</td>
</tr>
<tr>
<td>AWS Batch on AWS Fargate</td>
<td>109</td>
</tr>
<tr>
<td>Compute environment</td>
<td>81</td>
</tr>
<tr>
<td>Job queues</td>
<td>76</td>
</tr>
<tr>
<td>Job definitions on Fargate</td>
<td>73</td>
</tr>
<tr>
<td>Job queue template</td>
<td>77</td>
</tr>
<tr>
<td>Job queue parameters</td>
<td>77</td>
</tr>
<tr>
<td>Job queue name</td>
<td>77</td>
</tr>
<tr>
<td>State</td>
<td>77</td>
</tr>
<tr>
<td>Priority</td>
<td>78</td>
</tr>
<tr>
<td>Compute environment order</td>
<td>78</td>
</tr>
<tr>
<td>Tags</td>
<td>78</td>
</tr>
<tr>
<td>Job Scheduling</td>
<td>80</td>
</tr>
<tr>
<td>Compute environment parameters</td>
<td>81</td>
</tr>
<tr>
<td>Managed compute environments</td>
<td>81</td>
</tr>
<tr>
<td>Unmanaged compute environments</td>
<td>82</td>
</tr>
<tr>
<td>Create a compute environment</td>
<td>82</td>
</tr>
<tr>
<td>Compute resource AMIs</td>
<td>83</td>
</tr>
<tr>
<td>Compute resource AMI specification</td>
<td>83</td>
</tr>
<tr>
<td>Creating a compute resource AMI</td>
<td>83</td>
</tr>
<tr>
<td>Using a GPU workload AMI</td>
<td>85</td>
</tr>
<tr>
<td>Launch template support</td>
<td>89</td>
</tr>
<tr>
<td>Amazon EC2 user data in launch templates</td>
<td>90</td>
</tr>
<tr>
<td>Creating a compute environment using AWS Fargate resources</td>
<td>93</td>
</tr>
<tr>
<td>To create a managed compute environment using AWS Fargate resources</td>
<td>93</td>
</tr>
<tr>
<td>To create a managed compute environment using EC2 resources</td>
<td>94</td>
</tr>
<tr>
<td>To create an unmanaged compute environment using EC2 resources</td>
<td>97</td>
</tr>
<tr>
<td>Compute environment template</td>
<td>98</td>
</tr>
<tr>
<td>Compute environment parameters</td>
<td>99</td>
</tr>
<tr>
<td>Compute environment name</td>
<td>99</td>
</tr>
<tr>
<td>Type</td>
<td>99</td>
</tr>
<tr>
<td>State</td>
<td>99</td>
</tr>
<tr>
<td>Compute resources</td>
<td>100</td>
</tr>
<tr>
<td>Service role</td>
<td>105</td>
</tr>
<tr>
<td>Tags</td>
<td>106</td>
</tr>
<tr>
<td>Allocation strategies</td>
<td>106</td>
</tr>
<tr>
<td>Memory Management</td>
<td>107</td>
</tr>
<tr>
<td>Reserve System Memory</td>
<td>107</td>
</tr>
<tr>
<td>AWS Batch on AWS Fargate</td>
<td>109</td>
</tr>
<tr>
<td>Job definitions on Fargate</td>
<td>109</td>
</tr>
<tr>
<td>Job queues on Fargate</td>
<td>110</td>
</tr>
<tr>
<td>Compute environments on Fargate</td>
<td>110</td>
</tr>
<tr>
<td>Elastic Fabric Adapter</td>
<td>112</td>
</tr>
<tr>
<td>IAM Policies, Roles, and Permissions</td>
<td>114</td>
</tr>
<tr>
<td>Policy Structure</td>
<td>114</td>
</tr>
<tr>
<td>Policy Syntax</td>
<td>115</td>
</tr>
<tr>
<td>Actions for AWS Batch</td>
<td>115</td>
</tr>
<tr>
<td>Amazon Resource Names for AWS Batch</td>
<td>116</td>
</tr>
<tr>
<td>Testing Permissions</td>
<td>116</td>
</tr>
<tr>
<td>Supported Resource-Level Permissions</td>
<td>117</td>
</tr>
<tr>
<td>Example Policies</td>
<td>122</td>
</tr>
<tr>
<td>Read-Only Access</td>
<td>123</td>
</tr>
<tr>
<td>Restricting User, Image, Privilege, Role</td>
<td>123</td>
</tr>
<tr>
<td>Restrict Job Submission</td>
<td>124</td>
</tr>
</tbody>
</table>
What Is AWS Batch?

AWS Batch helps you to run batch computing workloads on the AWS Cloud. Batch computing is a common way for developers, scientists, and engineers to access large amounts of compute resources, and AWS Batch removes the undifferentiated heavy lifting of configuring and managing the required infrastructure, similar to traditional batch computing software. This service can efficiently provision resources in response to jobs submitted in order to eliminate capacity constraints, reduce compute costs, and deliver results quickly.

As a fully managed service, AWS Batch helps you to run batch computing workloads of any scale. AWS Batch automatically provisions compute resources and optimizes the workload distribution based on the quantity and scale of the workloads. With AWS Batch, there is no need to install or manage batch computing software, which allows you to focus on analyzing results and solving problems.

Components of AWS Batch

AWS Batch is a regional service that simplifies running batch jobs across multiple Availability Zones within a region. You can create AWS Batch compute environments within a new or existing VPC. After a compute environment is up and associated with a job queue, you can define job definitions that specify which Docker container images to run your jobs. Container images are stored in and pulled from container registries, which may exist within or outside of your AWS infrastructure.

Jobs

A unit of work (such as a shell script, a Linux executable, or a Docker container image) that you submit to AWS Batch. It has a name, and runs as a containerized application on AWS Fargate or Amazon EC2 resources in your compute environment, using parameters that you specify in a job definition. Jobs can reference other jobs by name or by ID, and can be dependent on the successful completion of other jobs. For more information, see Jobs (p. 14).

Job Definitions

A job definition specifies how jobs are to be run; you can think of it as a blueprint for the resources in your job. You can supply your job with an IAM role to provide programmatic access to other AWS resources, and you specify both memory and CPU requirements. The job definition can also control container properties, environment variables, and mount points for persistent storage. Many of the specifications in a job definition can be overridden by specifying new values when submitting individual Jobs. For more information, see Job definitions (p. 32)

Job Queues

When you submit an AWS Batch job, you submit it to a particular job queue, where it resides until it is scheduled onto a compute environment. You associate one or more compute environments with a job queue, and you can assign priority values for these compute environments and even across job queues themselves. For example, you could have a high priority queue that you submit time-sensitive jobs to, and a low priority queue for jobs that can run anytime when compute resources are cheaper.

Compute Environment

A compute environment is a set of managed or unmanaged compute resources that are used to run jobs. Managed compute environments allow you to specify desired compute type (Fargate or EC2) at
several levels of detail. You can set up compute environments that use a particular type of EC2 instance, a particular model such as c5.2xlarge or m5.10xlarge, or simply specify that you want to use the newest instance types. You can also specify the minimum, desired, and maximum number of vCPUs for the environment, along with the amount you are willing to pay for a Spot Instance as a percentage of the On-Demand Instance price and a target set of VPC subnets. AWS Batch will efficiently launch, manage, and terminate compute types as needed. You can also manage your own compute environments. In this case you are responsible for setting up and scaling the instances in an Amazon ECS cluster that AWS Batch creates for you. For more information, see Compute environment (p. 81).

Getting Started

Get started with AWS Batch by creating a job definition, compute environment, and a job queue in the AWS Batch console.

The AWS Batch first-run wizard gives you the option of creating a compute environment and a job queue and submitting a sample hello world job. If you already have a Docker image you would like to launch in AWS Batch, you can create a job definition with that image and submit that to your queue instead. For more information, see Getting Started with AWS Batch (p. 9).
Setting Up with AWS Batch

If you’ve already signed up for Amazon Web Services (AWS) and have been using Amazon Elastic Compute Cloud (Amazon EC2) or Amazon Elastic Container Service (Amazon ECS), you are close to being able to use AWS Batch. The setup process for these services is very similar, as AWS Batch uses Amazon ECS container instances in its compute environments. To use the AWS CLI with AWS Batch, you must use a version of the AWS CLI that supports the latest AWS Batch features. If you do not see support for an AWS Batch feature in the AWS CLI, you should upgrade to the latest version. For more information, see http://aws.amazon.com/cli/.

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

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

1. Sign Up for AWS (p. 3)
2. Create an IAM User (p. 3)
3. Create IAM Roles for your Compute Environments and Container Instances (p. 5)
4. Create a Key Pair (p. 5)
5. Create a Virtual Private Cloud (p. 7)
6. Create a Security Group (p. 7)
7. Install the AWS CLI (p. 8)

Sign Up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all services, including Amazon EC2 and AWS Batch. 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

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 AWS Batch, 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 IAM user's credentials.

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 below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

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

3. For User name, enter Administrator.

4. Select the check box next to AWS Management Console access. Then select Custom password, and then enter your new password in the text box.

5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to User must create a new password at next sign-in to allow the new user to reset their password after they sign in.

6. Choose Next: Permissions.

7. Under Set permissions, choose Add user to group.

8. Choose Create group.

9. In the Create group dialog box, for Group name enter Administrators.

10. Choose Filter policies, and then select AWS managed -job function to filter the table contents.

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

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

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

13. Choose Next: Tags.

14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM entities in the IAM User Guide.

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

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

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where your_aws_account_id is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):
Enter the IAM user name and password that you just created. When you're signed in, the navigation bar
displays "your_user_name @ your_aws_account_id".

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

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

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

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

Create IAM Roles for your Compute Environments
and Container Instances

Your AWS Batch compute environments and container instances require AWS account credentials to
make calls to other AWS APIs on your behalf. You must create an IAM role that provides these credentials
to your compute environments and container instances, then associate that role with your compute
environments.

Note
The AWS Batch compute environment and container instance roles are automatically created
for you in the console first-run experience, so if you intend to use the AWS Batch console,
you can move ahead to the next section. If you plan to use the AWS CLI instead, complete the
procedures in AWS Batch Service IAM Role (p. 126) and Amazon ECS Instance Role (p. 129)
before creating your first compute environment.

Create a Key Pair

AWS uses public-key cryptography to secure the login information for your instance. A Linux instance,
such as an AWS Batch compute environment 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
create your compute environment, 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. Note that 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 an instance in the US West (Oregon) region, you must create a key pair for the
instance in the same region.
3. In the navigation pane, choose **Key Pairs, Create Key Pair**.

4. In the **Create Key Pair** dialog box, for **Key pair name**, enter a name for the new key pair, and choose **Create**. Choose a name that you can remember, such as your IAM user name, followed by –key-pair, plus the Region name. For example, me-key-pair-uswest2.

5. 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. You'll need to provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.

6. If you will use an SSH client on a Mac 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 Mac 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'll need to install it and use the following procedure to convert the .pem file to a .ppk file.

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

1. Download and install PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/. Be sure to install the entire suite.

2. Start PuTTYgen (for example, from the **Start** menu, choose **All Programs, PuTTY, and PuTTYgen**).

3. Under **Type of key to generate**, choose **RSA**. If you're using an earlier version of PuTTYgen, choose **SSH-2 RSA**.

4. Choose **Load**. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, choose 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.

If you have a default VPC, you also can skip this section and move to the next task, Create a Security Group (p. 7). To determine whether you have a default VPC, see Supported Platforms in the Amazon EC2 Console in the Amazon EC2 User Guide for Linux Instances. Otherwise, you can create a nondefault VPC in your account using the steps below.

**Important**

If your account supports 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 **Start 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 for **VPC name**. 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 compute environment 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.

Note that 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*.

**Note**

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 need to 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 you can remember, such as your IAM user name, followed by _SG_, plus the Region name. For example, me_SG_useast1.

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

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

6. AWS Batch 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 containers in jobs with Docker commands. You can also add rules for HTTP if you want your container instance to host a job that runs a web server. Complete the following steps to add these optional security group rules.

On the Inbound tab, create the following rules and choose Create:

- Choose Add Rule. For Type, choose HTTP. For Source, choose Anywhere (0.0.0.0/0).
- Choose Add Rule. For Type, choose SSH. For Source, 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.

   Note
   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

To use the AWS CLI with AWS Batch, 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 AWS Batch

Get started with AWS Batch by creating a job definition, compute environment, and a job queue in the AWS Batch console.

The AWS Batch first-run wizard gives you the option of creating a compute environment and a job queue and submitting a sample hello world job. If you already have a Docker image you would like to launch in AWS Batch, you can create a job definition with that image and submit that to your queue instead.

**Important**
Before you begin, be sure that you've completed the steps in Setting Up with AWS Batch (p. 3) and that your AWS user has the required permissions (admin users don't need to worry about permissions issues). For more information, see Creating Your First IAM Admin User and Group in the IAM User Guide.

Step 1: Define a Job

In this section, you choose to define your job definition or move ahead to creating a compute environment and job queue without a job definition.

**To configure job options**

2. To create an AWS Batch job definition, compute environment, and job queue and then submit your job, choose **Using Amazon EC2**. To only create the compute environment and job queue without submitting a job, choose **No job submission**.
3. If you chose to create a job definition, then complete the next four sections of the first-run wizard, **Job run-time**, **Environment**, **Parameters**, and **Environment variables** and then choose **Next**. If you are not creating a job definition, choose **Next** and move on to Step 2: Configure the Compute Environment and Job Queue (p. 11).

**To specify job run time**

1. If you are creating a new job definition, for **Job definition name**, specify a name for your job definition.
2. (Optional) For **Job role**, you can specify an IAM role that provides the container in your job with permissions to use the AWS APIs. This feature uses Amazon ECS IAM roles for task functionality. For more information about this feature, including configuration prerequisites, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

**Note**
Only roles that have the Amazon Elastic Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your AWS Batch jobs, see Creating an IAM Role and Policy for your Tasks in the Amazon Elastic Container Service Developer Guide.

3. For **Container image**, choose the Docker image to use for your job. Images in the Docker Hub registry are available by default. You can also specify other repositories with repository-
url/image:tag. 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.

Note
Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, ARM-based Docker images can only run on ARM-based compute resources.

- Images in Amazon ECR repositories use the full registry/repository:tag naming convention. For example, aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest
- 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).

To specify resources for your environment

1. For Command, specify the command to pass 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.

   Note
   You can use parameter substitution default values and placeholders in your command. For more information, see Parameters (p. 44).

2. For vCPUs, specify the number of vCPUs to 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. Each vCPU is equivalent to 1,024 CPU shares.

3. For Memory, specify the hard limit (in MiB) of memory to present to the job's container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run.

4. For Job attempts, specify the maximum number of times to attempt your job (in case it fails). For more information, see Automated Job Retries (p. 18).

Parameters

You can optionally specify parameter substitution default values and placeholders in your command. For more information, see Parameters (p. 44).

1. For Key, specify the key for your parameter.
2. For Value, specify the value for your parameter.

To specify environment variables

You can optionally specify environment variables to pass to your job's 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 don't recommend using plaintext environment variables for sensitive information, such as credential data.

1. For Key, specify the key for your environment variable.
2. For Value, specify the value for your environment variable.

Step 2: Configure the Compute Environment and Job Queue

A compute environment is a way to reference your compute resources (Amazon EC2 instances): the settings and constraints that tell AWS Batch how instances should be configured and automatically launched. You submit your jobs to a job queue that stores jobs until the AWS Batch scheduler runs the job on a compute resource within your compute environment.

Note
At this time, you can only create a managed compute environment in the first run wizard. To create an unmanaged compute environment, see Creating a compute environment (p. 93).

To configure your compute environment type

1. For Compute environment name, specify a unique name for your compute environment.
2. For Service role, choose to create a new role or use an existing role that allows the AWS Batch service to make calls to the required AWS APIs on your behalf. For more information, see AWS Batch Service IAM Role (p. 126). If you choose to create a new role, the required role (AWSBatchServiceRole) is created for you.
3. For EC2 instance role, choose to create a new role or use an existing role that allows the Amazon ECS container instances that are created for your compute environment to make calls to the required AWS APIs on your behalf. For more information, see Amazon ECS Instance Role (p. 129). If you choose to create a new role, the required role (ecsInstanceRole) is created for you.

To configure your instances

1. For Provisioning model, choose On-Demand to launch Amazon EC2 On-Demand instances or Spot to use Amazon EC2 Spot Instances.
2. If you chose to use Amazon EC2 Spot Instances:
   a. For Maximum bid price, choose the maximum percentage that a Spot Instance price must be when compared with the On-Demand price for that instance type before instances are launched. For example, if your bid percentage is 20%, then the Spot price must be less than 20% of the current On-Demand price for that EC2 instance. You always pay the lowest (market) price and never more than your maximum percentage.
   b. For Spot fleet role, choose to create a new role or use an existing Amazon EC2 Spot Fleet IAM role to apply to your Spot compute environment. If you choose to create a new role, the required role (aws-ec2-spot-fleet-role) is created for you. For more information, see Amazon EC2 Spot Fleet Role (p. 130).
3. For Allowed instance types, choose the Amazon EC2 instance types that can be launched. You can specify instance families to launch any instance type within those families (for example, c5, c5n, or p3), or you can specify specific sizes within a family (such as c5.8xlarge). Note that metal instance types are not in the instance families (for example c5 does not include c5.metal.) You can also choose optimal to pick instance types (from the C4, M4, and R4 instance families) on the fly that match the demand of your job queues.

Note
When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can't mix x86 and ARM instances in the same compute environment.
Note
Currently, optimal uses instance types from the C4, M4, and R4 instance families. In Regions that don't have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

4. For **Minimum vCPUs**, choose the minimum number of EC2 vCPUs that your compute environment should maintain, regardless of job queue demand.

5. For **Desired vCPUs**, choose the number of EC2 vCPUs with which your compute environment should launch. As your job queue demand increases, AWS Batch can increase the desired number of vCPUs in your compute environment and add EC2 instances, up to the maximum vCPUs, and as demand decreases, AWS Batch can decrease the desired number of vCPUs in your compute environment and remove instances, down to the minimum vCPUs.

6. For **Maximum vCPUs**, choose the maximum number of EC2 vCPUs that your compute environment can scale out to, regardless of job queue demand.

**To set up your networking**

Compute resources are launched into the VPC and subnets that you specify here. This allows you to control the network isolation of AWS Batch compute resources.

**Important**
Compute resources need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your compute resources having public IP addresses.

For more information about interface VPC endpoints, see Amazon ECS Interface VPC Endpoints (AWS PrivateLink) in the Amazon Elastic Container Service Developer Guide.

If you do not have an interface VPC endpoint configured and your compute resources 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. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Compute Environments (p. 147).

1. For **VPC Id**, choose a VPC into which to launch your instances.

2. For **Subnets**, choose which subnets in the selected VPC should host your instances. By default, all subnets within the selected VPC are chosen.

3. For **Security groups**, choose a security group to attach to your instances. By default, the default security group for your VPC is chosen.

**To tag your instances**

You can optionally apply key-value pair tags to instances that are launched in your compute environment. For example, you can specify "Name": "AWS Batch Instance – C4OnDemand" as a tag so that each instance in your compute environment has that name (this is helpful for recognizing your AWS Batch instances in the Amazon EC2 console). By default, the compute environment name is used to tag your instances.

1. For **Key**, specify the key for your tag.

2. For **Value**, specify the value for your tag.

**To set up your job queue**

You submit your jobs to a job queue which stores jobs until the AWS Batch scheduler runs the job on a compute resource within your compute environment.

- For **Job queue name**, choose a unique name for your job queue.
To review and create

The Connected compute environments for this job queue section shows that your new compute environment is associated with your new job queue and its order. Later, you can associate other compute environments with the job queue. The job scheduler uses the compute environment order to determine which compute environment should start a given job. Compute environments must be in the VALID state before you can associate them with a job queue. You can associate up to three compute environments with a job queue.

- Review the compute environment and job queue configuration and choose Create to create your compute environment.
Jobs

Jobs are the unit of work executed by AWS Batch. Jobs can be executed as containerized applications running on Amazon ECS container instances in an ECS cluster.

Containerized jobs can reference a container image, command, and parameters. For more information, see Job definition parameters (p. 43).

You can submit a large number of independent, simple jobs.

Topics
• Submitting a Job (p. 14)
• Job States (p. 16)
• AWS Batch Job Environment Variables (p. 17)
• Automated Job Retries (p. 18)
• Job Dependencies (p. 19)
• Job Timeouts (p. 19)
• Array Jobs (p. 20)
• Multi-node Parallel Jobs (p. 28)
• GPU Jobs (p. 31)

Submitting a Job

After you have registered a job definition, you can submit it as a job to an AWS Batch job queue. Many of the parameters that are specified in the job definition can be overridden at runtime.

To submit a job

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Jobs, Submit job.
4. For Job name, choose a unique name for your job.
5. For Job definition, choose a previously created job definition for your job. For more information, see Creating a job definition (p. 32).
6. For Job queue, choose a previously created job queue. For more information, see Creating a job queue (p. 76).
7. For Job type, choose Single for a single job or Array to submit an array job. For more information, see Array Jobs (p. 20). This option isn't available for multi-node parallel jobs.
8. (Array jobs only) For Array size, specify an array size between 2 and 10,000.
9. (Optional) Declare any job dependencies. A job may have up to 20 dependencies. For more information, see Job Dependencies (p. 19).
   a. For Job depends on, enter the job IDs for any jobs that must finish before this job starts.
b. (Array jobs only) For **N-To-N job dependencies**, specify one or more job IDs for any array jobs for which each child job index of this job should depend on the corresponding child job index of the dependency. For example, JobB: 1 depends on JobA: 1, and so on.

c. (Array jobs only) Select **Run children sequentially** to create a **SEQUENTIAL** dependency for the current array job. This ensures that each child index job waits for its earlier sibling to finish. For example, JobA: 1 depends on JobA: 0 and so on.

10. For **Job attempts**, specify the maximum number of times to attempt your job (in case it fails). For more information, see Automated Job Retries (p. 18).

11. (Optional) For **Execution timeout**, specify the maximum number of seconds to allow your job attempts to run. If an attempt exceeds the timeout duration, it is stopped and the status moves to **FAILED**. For more information, see Job Timeouts (p. 19).

    **Important**
    Jobs run on Fargate resources can’t expect to run for more than 14 days. After 14 days, the Fargate resources may no longer be available and the job will be terminated.

12. (Optional) In the **Parameters** section, you can specify parameter substitution default values and placeholders to use in the command that your job’s container runs when it starts. For more information, see Parameters (p. 44).

    a. Choose **Add parameter**.
    b. For **Key**, specify the key for your parameter.
    c. For **Value**, specify the value for your parameter.

13. For **vCPUs**, specify the number of vCPUs to 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. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

14. For **Memory**, specify the hard limit (in MiB) of memory to present to the job’s container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run. You must specify at least 4 MiB of memory for a job.

15. (Optional) For **Number of GPUs**, specify the number of GPUs your job will use.

    The job will run on a container with the specified number of GPUs pinned to that container.

16. For **Command**, specify the command to pass to the container. For simple commands, you can type the command as you would at a command prompt in the **Space delimited** tab. Verify that the JSON result (which is passed to the Docker daemon) is correct. For more complicated commands (for example, with special characters), you can switch to the **JSON** tab and enter the string array equivalent there.

    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.

    **Note**
    You can use parameter substitution default values and placeholders in your command. For more information, see Parameters (p. 44).

17. (Optional) You can specify environment variables to pass to your job’s 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.

    a. Choose **Add environment variable**.
    b. For **Key**, specify the key for your environment variable.
Note

Environment variables must not start with AWS_BATCH; this naming convention is reserved for variables that are set by the AWS Batch service.

c. For Value, specify the value for your environment variable.

18. (Optional) In the Tags section, you can specify the key and value for each tag to associate with the job. For more information, see Tagging your AWS Batch resources (p. 163).

19. Choose Submit job.

Note

Logs for RUNNING, SUCCEEDED, and FAILED jobs are available in CloudWatch Logs; the log group is /aws/batch/job, and the log stream name format is first200CharsOfJobDefinitionName/default/ecs_task_id (this format may change in the future).

After a job reaches the RUNNING status, you can programmatically retrieve its log stream name with the DescribeJobs API operation. For more information, see View Log Data Sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide. By default, these logs are set to never expire, but you can modify the retention period. For more information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

Job States

When you submit a job to an AWS Batch job queue, the job enters the SUBMITTED state. It then passes through the following states until it succeeds (exits with code 0) or fails (exits with a non-zero code). AWS Batch jobs can have the following states:

SUBMITTED

A job that has been submitted to the queue, and has not yet been evaluated by the scheduler. The scheduler evaluates the job to determine if it has any outstanding dependencies on the successful completion of any other jobs. If there are dependencies, the job is moved to PENDING. If there are no dependencies, the job is moved to RUNNABLE.

Note

If your jobs do not progress to STARTING, see Jobs Stuck in RUNNABLE Status (p. 169) in the troubleshooting section.

PENDING

A job that resides in the queue and isn't yet able to run due to a dependency on another job or resource. After the dependencies are satisfied, the job is moved to RUNNABLE.

RUNNABLE

A job that resides in the queue, has no outstanding dependencies, and is therefore ready to be scheduled to a host. Jobs in this state are started as soon as sufficient resources are available in one of the compute environments that are mapped to the job's queue. However, jobs can remain in this state indefinitely when sufficient resources are unavailable.

Note

If your jobs do not progress to STARTING, see Jobs Stuck in RUNNABLE Status (p. 169) in the troubleshooting section.

STARTING

These jobs have been scheduled to a host and the relevant container initiation operations are underway. After the container image is pulled and the container is up and running, the job transitions to RUNNING.

RUNNING

The job is running as a container job on an Amazon ECS container instance within a compute environment. When the job's container exits, the process exit code determines whether the job
succeeded or failed. An exit code of 0 indicates success, and any non-zero exit code indicates failure. If the job associated with a failed attempt has any remaining attempts left in its optional retry strategy configuration, the job is moved to RUNNABLE again. For more information, see Automated Job Retries (p. 18).

Note
Logs for RUNNING jobs are available in CloudWatch Logs; the log group is /aws/batch/job, and the log stream name format is first200CharsOfJobDefinitionName/default/ecs_task_id (this format may change in the future). After a job reaches the RUNNING status, you can programatically retrieve its log stream name with the DescribeJobs API operation. For more information, see View Log Data Sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide. By default, these logs are set to never expire, but you can modify the retention period. For more information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

SUCCEEDED

The job has successfully completed with an exit code of 0. The job state for SUCCEEDED jobs is persisted in AWS Batch for at least 24 hours.

Note
Logs for SUCCEEDED jobs are available in CloudWatch Logs; the log group is /aws/batch/job, and the log stream name format is first200CharsOfJobDefinitionName/default/ecs_task_id (this format may change in the future). After a job reaches the RUNNING status, you can programatically retrieve its log stream name with the DescribeJobs API operation. For more information, see View Log Data Sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide. By default, these logs are set to never expire, but you can modify the retention period. For more information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

FAILED

The job has failed all available attempts. The job state for FAILED jobs is persisted in AWS Batch for at least 24 hours.

Note
Logs for FAILED jobs are available in CloudWatch Logs; the log group is /aws/batch/job, and the log stream name format is first200CharsOfJobDefinitionName/default/ecs_task_id (this format may change in the future). After a job reaches the RUNNING status, you can programatically retrieve its log stream with the DescribeJobs API operation. For more information, see View Log Data Sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide. By default, these logs are set to never expire, but you can modify the retention period. For more information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

AWS Batch Job Environment Variables

AWS Batch automatically sets specific environment variables in container jobs. These environment variables provide introspection for the containers inside jobs, and you can use the values of these variables in the logic of your applications. All variables that are set by AWS Batch begin with the prefix, AWS_BATCH_. This is a protected environment variable prefix, and you cannot use this prefix for your own variables in job definitions or overrides.

The following environment variables are available in job containers:

AWS_BATCH_CE_NAME

This variable is set to the name of the compute environment in which your job is placed.
AWS Batch User Guide
Automated Job Retries

AWS_BATCH_JOB_ARRAY_INDEX

This variable is only set in child array jobs. The array job index begins at 0, and each child job receives a unique index number. For example, an array job with 10 children has index values of 0-9. You can use this index value to control how your array job children are differentiated. For more information, see Tutorial: Using the Array Job Index to Control Job Differentiation (p. 24).

AWS_BATCH_JOB_ATTEMPT

This variable is set to the job attempt number. The first attempt is numbered 1. For more information, see Automated Job Retries (p. 18).

AWS_BATCH_JOB_ID

This variable is set to the AWS Batch job ID.

AWS_BATCH_JOB_MAIN_NODE_INDEX

This variable is only set in multi-node parallel jobs. This variable is set to the index number of the job’s main node. Your application code can compare the AWS_BATCH_JOB_MAIN_NODE_INDEX to the AWS_BATCH_JOB_NODE_INDEX on an individual node to determine if it is the main node.

AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS

This variable is only set in multi-node parallel job child nodes (it isn't present on the main node). This variable is set to the private IPv4 address of the job's main node. Your child node’s application code can use this address to communicate with the main node.

AWS_BATCH_JOB_NODE_INDEX

This variable is only set in multi-node parallel jobs. This variable is set to the node index number of the node. The node index begins at 0, and each node receives a unique index number. For example, a multi-node parallel job with 10 children has index values of 0-9.

AWS_BATCH_JOB_NUM_NODES

This variable is only set in multi-node parallel jobs. This variable is set to the number of nodes that you have requested for your multi-node parallel job.

AWS_BATCH_JQ_NAME

This variable is set to the name of the job queue to which your job was submitted.

Automated Job Retries

You can apply a retry strategy to your jobs and job definitions that allows failed jobs to be automatically retried. Possible failure scenarios include:

- Any non-zero exit code from a container job
- Amazon EC2 instance failure or termination
- Internal AWS service error or outage

When a job is submitted to a job queue and placed into the RUNNING state, that is considered an attempt. By default, each job is given one attempt to move to either the SUCCEEDED or FAILED job state. However, both the job definition and the job submission workflows allow you to specify a retry strategy with between 1 and 10 attempts. For more information, see Retry strategy (p. 59).

At runtime, the AWS_BATCH_JOB_ATTEMPT environment variable is set to the container’s corresponding job attempt number. The first attempt is numbered 1, and subsequent attempts are in ascending order (2, 3, 4, and so on).
If a job attempt fails for any reason, and the number of attempts specified in the retry configuration is greater than the AWS_BATCH_JOB_ATTEMPT number, then the job is placed back in the Runnable state. For more information, see Job States (p. 16).

**Note**
Jobs that have been cancelled or terminated are not retried. Also, jobs that fail due to an invalid job definition are not retried.

For more information, see Creating a job definition (p. 32) and Submitting a Job (p. 14).

### Job Dependencies

When you submit an AWS Batch job, you can specify the job IDs on which the job depends. When you do so, the AWS Batch scheduler ensures that your job is run only after the specified dependencies have successfully completed. After they succeed, the dependent job transitions from PENDING to Runnable and then to STARTING and RUNNING. If any of the job dependencies fail, the dependent job automatically transitions from PENDING to FAILED.

For example, Job A can express a dependency on up to 20 other jobs that must succeed before it can run. You can then submit additional jobs that have a dependency on Job A and up to 19 other jobs.

For array jobs, you can specify a Sequential type dependency without specifying a job ID so that each child array job completes sequentially, starting at index 0. You can also specify an N TO N type dependency with a job ID. That way, each index child of this job must wait for the corresponding index child of each dependency to complete before it can begin. For more information, see Array Jobs (p. 20).

To submit an AWS Batch job with dependencies, see Submitting a Job (p. 14).

### Job Timeouts

You can configure a timeout duration for your jobs so that if a job runs longer than that, AWS Batch terminates the job. For example, you might have a job that you know should only take 15 minutes to complete. Sometimes your application gets stuck in a loop and runs forever, so you can set a timeout of 30 minutes to terminate the stuck job.

You specify an attemptDurationSeconds parameter, which must be at least 60 seconds, either in your job definition, or when you submit the job. When this number of seconds has passed following the job attempt's startedAt timestamp, AWS Batch terminates the job. On the compute resource, your job's container receives a SIGTERM signal to give your application a chance to shut down gracefully. If the container is still running after 30 seconds, a SIGKILL signal is sent to forcefully shut down the container.

Timeout terminations are handled on a best-effort basis. You shouldn't expect your timeout termination to happen exactly when the job attempt times out (it may take a few seconds longer). If your application requires precise timeout execution, you should implement this logic within the application. If you have a large number of jobs timing out concurrently, the timeout terminations behave as a first in, first out queue, where jobs are terminated in batches.

If a job is terminated for exceeding the timeout duration, it isn't retried. If a job attempt fails on its own, then it can retry if retries are enabled, and the timeout countdown is started over for the new attempt.

**Important**
Jobs run on Fargate resources can't expect to run for more than 14 days. If the timeout duration exceeds 14 days, the Fargate resources may no longer be available and the job will be terminated.
For array jobs, child jobs have the same timeout configuration as the parent job.

For information about submitting an AWS Batch job with a timeout configuration, see Submitting a Job (p. 14).

Array Jobs

An array job is a job that shares common parameters, such as the job definition, vCPUs, and memory. It runs as a collection of related, yet separate, basic jobs that may be distributed across multiple hosts and may run concurrently. Array jobs are the most efficient way to run extremely parallel jobs such as Monte Carlo simulations, parametric sweeps, or large rendering jobs.

AWS Batch array jobs are submitted just like regular jobs. However, you specify an array size (between 2 and 10,000) to define how many child jobs should run in the array. If you submit a job with an array size of 1000, a single job runs and spawns 1000 child jobs. The array job is a reference or pointer to manage all the child jobs. This allows you to submit large workloads with a single query.

When you submit an array job, the parent array job gets a normal AWS Batch job ID. Each child job has the same base ID, but the array index for the child job is appended to the end of the parent ID, such as example_job_ID:0 for the first child job of the array.

At runtime, the AWS_BATCH_JOB_ARRAY_INDEX environment variable is set to the container's corresponding job array index number. The first array job index is numbered 0, and subsequent attempts are in ascending order (1, 2, 3, and so on). You can use this index value to control how your array job children are differentiated. For more information, see Tutorial: Using the Array Job Index to Control Job Differentiation (p. 24).

For array job dependencies, you can specify a type for a dependency, such as SEQUENTIAL or N_TO_N. You can specify a SEQUENTIAL type dependency (without specifying a job ID) so that each child array job completes sequentially, starting at index 0. For example, if you submit an array job with an array size of 100, and specify a dependency with type SEQUENTIAL, 100 child jobs are spawned sequentially, where the first child job must succeed before the next child job starts. The figure below shows Job A, an array job with an array size of 10. Each job in Job A's child index is dependent on the previous child job. Job A:1 can't start until job A:0 finishes.
You can also specify an \texttt{N\_TO\_N} type dependency with a job ID for array jobs so that each index child of this job must wait for the corresponding index child of each dependency to complete before it can begin. The figure below shows Job A and Job B, two array jobs with an array size of 10,000 each. Each job in Job B's child index is dependent on the corresponding index in Job A. Job B:1 can't start until job A:1 finishes.
If you cancel or terminate a parent array job, all of the child jobs are cancelled or terminated with it. You can cancel or terminate individual child jobs (which moves them to the FAILED status) without affecting the other child jobs. However, if a child array job fails (on its own or by cancelling/terminating manually), the parent job also fails.

**Example Array Job Workflow**

A common workflow for AWS Batch customers is to run a prerequisite setup job, run a series of commands against a large number of input tasks, and then conclude with a job that aggregates results and writes summary data to Amazon S3, DynamoDB, Amazon Redshift, or Aurora.

For example:

- **JobA**: A standard, non-array job that performs a quick listing and metadata validation of objects in an Amazon S3 bucket, `BucketA`. The SubmitJob JSON syntax is shown below.
Example Array Job Workflow

• **JobA**: An array job that runs CPU-intensive commands against each object in BucketA and uploads results to BucketB. The *SubmitJob* JSON syntax is shown below.

```json
{
  "jobName": "JobA",
  "jobQueue": "ProdQueue",
  "jobDefinition": "JobA-list-and-validate:1"
}
```

• **JobB**: An array job with 10,000 copies that is dependent upon JobA, that runs CPU-intensive commands against each object in BucketA and uploads results to BucketB. The *SubmitJob* JSON syntax is shown below.

```json
{
  "jobName": "JobB",
  "jobQueue": "ProdQueue",
  "jobDefinition": "JobB-CPU-Intensive-Processing:1",
  "containerOverrides": {
    "vcpus": 32,
    "memory": 4096
  }
}
```

• **JobC**: Another 10,000 copy array job that is dependent upon JobB with an N_TO_N dependency model, that runs memory-intensive commands against each item in BucketB, writes metadata to DynamoDB, and uploads the resulting output to BucketC. The *SubmitJob* JSON syntax is shown below.

```json
{
  "jobName": "JobC",
  "jobQueue": "ProdQueue",
  "jobDefinition": "JobC-Memory-Intensive-Processing:1",
  "containerOverrides": {
    "vcpus": 1,
    "memory": 32768
  }
}
```

• **JobD**: An array job that performs 10 validation steps that each need to query DynamoDB and may interact with any of the above Amazon S3 buckets. Each of the steps in JobD run the same command, but the behavior is different based on the value of the `AWS_BATCH_JOB_ARRAY_INDEX` environment variable within the job's container. These validation steps run sequentially (for example, JobD:0, then JobD:1, and so on. The *SubmitJob* JSON syntax is shown below.

```json
{
  "jobName": "JobD",
  "jobQueue": "ProdQueue",
  "jobDefinition": "JobD-validate:1",
  "containerOverrides": {
    "vcpus": 1,
    "memory": 4096
  }
}
```
"jobDefinition": "JobD-Sequential-Validation:1",
"containerOverrides": {
  "vcpus": 1,
  "memory": 32768
}
"arrayProperties": {
  "size": 10
},
"dependsOn": [
  {
    "jobId": "JobC_job_ID"
  },
  {
    "type": "SEQUENTIAL"
  }
]

• JobE: A final, non-array job that performs some simple cleanup operations and sends an Amazon
SNS notification with a message that the pipeline has completed and a link to the output URL. The
SubmitJob JSON syntax is shown below.

{
  "jobName": "JobE",
  "jobQueue": "ProdQueue",
  "jobDefinition": "JobE-Cleanup-and-Notification:1",
  "parameters": {
    "SourceBucket": "s3://JobD-Output-Bucket",
    "Recipient": "pipeline-notifications@mycompany.com"
  },
  "dependsOn": [
    {
      "jobId": "JobD_job_ID"
    }
  ]
}

Tutorial: Using the Array Job Index to Control Job Differentiation

This tutorial shows how to use the AWS_BATCH_JOB_ARRAY_INDEX environment variable (that each
child job is assigned) to differentiate the child jobs. The example works by using the child job’s index
number to read a specific line in a file and substitute the parameter associated with that line number
with a command inside the job’s container. The result is that you can have multiple AWS Batch jobs
running the same Docker image and command arguments, but the results are different because the array
job index is used as a modifier.

In this tutorial, you create a text file that has all of the colors of the rainbow, each on its own line. Then,
you create an entrypoint script for a Docker container that converts the index into a value that can be
used for a line number in the color file (the index starts at zero, but line numbers start at one). Create a
Dockerfile that copies the color and index files to the container image and sets the image’s ENTRYPOINT
to the entrypoint script. The Dockerfile and resources are built to a Docker image that is pushed to
Amazon ECR. You then register a job definition that uses your new container image, submit an AWS
Batch array job with that job definition, and view the results.
Prerequisites

This tutorial has the following prerequisites:

- An AWS Batch compute environment. For more information, see Creating a compute environment (p. 93).
- An AWS Batch job queue and associated compute environment. For more information, see Creating a job queue (p. 76).
- The AWS CLI installed on your local system. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.
- Docker installed on your local system. For more information, see About Docker CE in the Docker documentation.

Step 1: Build a Container Image

Although you can use the AWS_BATCH_JOB_ARRAY_INDEX in a job definition in the command parameter, it's easier and more powerful to create a container image that uses the variable in an entrypoint script. This section describes how to create such a container image.

To build your Docker container image

1. Create a new directory to use as your Docker image workspace and navigate to it.
2. Create a file called colors.txt in your workspace directory and paste the contents below into it.

   red
   orange
   yellow
   green
   blue
   indigo
   violet

3. Create a file called print-color.sh in your workspace directory and paste the contents below into it.

   ```bash
   #!/bin/sh
   LINE=$((AWS_BATCH_JOB_ARRAY_INDEX + 1))
   COLOR=$(sed -n ${LINE}p /tmp/colors.txt)
   echo My favorite color of the rainbow is $COLOR.
   ```

4. Create a file called Dockerfile in your workspace directory and paste the contents below into it. This Dockerfile copies the previous files to your container and sets the entrypoint script to run when the container starts.

   ```bash
   FROM busybox
   COPY print-color.sh /tmp/print-color.sh
   COPY colors.txt /tmp/colors.txt
   RUN chmod +x /tmp/print-color.sh
   ENTRYPOINT /tmp/print-color.sh
   ```

5. Build your Docker image:
6. Test your container with the following script. This script sets the `AWS_BATCH_JOB_ARRAY_INDEX` variable to 0 locally and then increments it to simulate what an array job with seven children would do.

```bash
AWS_BATCH_JOB_ARRAY_INDEX=0
while [ $AWS_BATCH_JOB_ARRAY_INDEX -le 6 ];
  do
    docker run -e AWS_BATCH_JOB_ARRAY_INDEX="$AWS_BATCH_JOB_ARRAY_INDEX" print-color
    AWS_BATCH_JOB_ARRAY_INDEX=$((AWS_BATCH_JOB_ARRAY_INDEX + 1))
  done
```

Output:

```
My favorite color of the rainbow is red.
My favorite color of the rainbow is orange.
My favorite color of the rainbow is yellow.
My favorite color of the rainbow is green.
My favorite color of the rainbow is blue.
My favorite color of the rainbow is indigo.
My favorite color of the rainbow is violet.
```

**Step 2: Push Your Image to Amazon ECR**

Now that you have built and tested your Docker container, you must push it to an image repository. This example uses Amazon ECR, but you can also choose to use another registry, such as DockerHub.

1. Create an Amazon ECR image repository to store your container image. For the sake of brevity, this example uses the AWS CLI, but you can also use the AWS Management Console. For more information, see Creating a Repository in the Amazon Elastic Container Registry User Guide.

   ```bash
   aws ecr create-repository --repository-name print-color
   ```

2. Tag your `print-color` image with your Amazon ECR repository URI that was returned from the previous step.

   ```bash
   docker tag print-color aws_account_id.dkr.ecr.region.amazonaws.com/print-color
   ```

3. Login to your Amazon ECR registry. For more information, see Registry Authentication in the Amazon Elastic Container Registry User Guide.

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

4. Push your image to Amazon ECR:

   ```bash
   docker push aws_account_id.dkr.ecr.region.amazonaws.com/print-color
   ```
Step 3: Create and Register a Job Definition

Now that your Docker image is in an image registry, you can specify it in an AWS Batch job definition and use it later to run an array job. For the sake of brevity, this example uses the AWS CLI, but you can also use the AWS Management Console. For more information, see Creating a job definition (p. 32).

To create a job definition

1. Create a file called `print-color-job-def.json` in your workspace directory and paste the contents below into it. Replace the image repository URI with your own image's URI.

   ```json
   {
   "jobDefinitionName": "print-color",
   "type": "container",
   "containerProperties": {
   "image": "aws_account_id.dkr.ecr.region.amazonaws.com/print-color",
   "vcpus": 1,
   "memory": 250
   }
   }
   ```

2. Register the job definition with AWS Batch:

   ```bash
   aws batch register-job-definition --cli-input-json file://print-color-job-def.json
   ```

Step 4: Submit an AWS Batch Array Job

After you have registered your job definition, you can submit an AWS Batch array job that uses your new container image.

To submit an AWS Batch array job

1. Create a file called `print-color-job.json` in your workspace directory and paste the contents below into it.

   ```json
   {
   "jobName": "print-color",
   "jobQueue": "first-run-job-queue",
   "arrayProperties": {
   "size": 7
   },
   "jobDefinition": "print-color"
   }
   ```

2. Submit the job to your AWS Batch job queue. Note the job ID that is returned in the output.

   ```bash
   aws batch submit-job --cli-input-json file://print-color-job.json
   ```

3. Describe the job's status and wait for the job to move to SUCCEEDED.
Step 5: View Your Array Job Logs

After your job reaches the succeeded status, you can view the CloudWatch Logs from the job's container.

To view your job's logs in CloudWatch Logs

2. In the left navigation pane, choose Jobs.
3. For Job queue, select a queue.
4. In the Status section, choose succeeded.
5. To display all of the child jobs for your array job, select the job ID that was returned in the previous section.
6. To see the logs from the job's container, select one of the child jobs and choose View logs.

7. View the other child job's logs. Each job returns a different color of the rainbow.

Multi-node Parallel Jobs

Multi-node parallel jobs enable you to run single jobs that span multiple Amazon EC2 instances. With AWS Batch multi-node parallel jobs, you can run large-scale, tightly coupled, high performance computing applications and distributed GPU model training without the need to launch, configure, and manage Amazon EC2 resources directly. An AWS Batch multi-node parallel job is compatible with any framework that supports IP-based, internode communication, such as Apache MXNet, TensorFlow, Caffe2, or Message Passing Interface (MPI).

Multi-node parallel jobs are submitted as a single job. However, your job definition (or job submission node overrides) specifies the number of nodes to create for the job and what node groups to create. Each multi-node parallel job contains a main node, which is launched first. After the main node is up, the child nodes are launched and started. If the main node exits, the job is considered finished, and the child nodes are stopped. For more information, see Node Groups (p. 29).

Multi-node parallel job nodes are single-tenant, meaning that only a single job container is run on each Amazon EC2 instance.

The final job status (succeeded or failed) is determined by the final job status of the main node. To get the status of a multi-node parallel job, you can describe the job using the job ID that was returned.
when you submitted the job. If you need the details for child nodes, then you must describe each child node individually. Nodes are addressed using \$N notation. For example, to access the details of the second node of a job, you need to describe `aws_batch_job_id#2` using the AWS Batch DescribeJobs API action. The `started`, `stoppedAt`, `statusReason`, and `exit` information for a multi-node parallel job is populated from the main node.

If you specify job retries, then a main node failure triggers another attempt; child node failures do not. Each new attempt of a multi-node parallel job updates the corresponding attempt of its associated child nodes.

To run multi-node parallel jobs on AWS Batch, your application code must contain the frameworks and libraries necessary for distributed communication.

### Environment Variables

At runtime, in addition to the standard environment variables that all AWS Batch jobs receive, each node is configured with the following environment variables that are specific to multi-node parallel jobs:

**AWS_BATCH_JOB_MAIN_NODE_INDEX**

This variable is set to the index number of the job’s main node. Your application code can compare the `AWS_BATCH_JOB_MAIN_NODE_INDEX` to the `AWS_BATCH_JOB_NODE_INDEX` on an individual node to determine if it is the main node.

**AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS**

This variable is only set in multi-node parallel job child nodes (it isn’t present on the main node). This variable is set to the private IPv4 address of the job’s main node. Your child node’s application code can use this address to communicate with the main node.

**AWS_BATCH_JOB_NODE_INDEX**

This variable is set to the node index number of the node. The node index begins at 0, and each node receives a unique index number. For example, a multi-node parallel job with 10 children has index values of 0-9.

**AWS_BATCH_JOB_NUM_NODES**

This variable is set to the number of nodes that you have requested for your multi-node parallel job.

### Node Groups

A node group is an identical group of job nodes that all share the same container properties. AWS Batch lets you specify up to five distinct node groups for each job.

Each group can have its own container images, commands, environment variables, and so on. For example, you can submit a job that requires a single `c4.xlarge` instance for the main node, and five `c4.xlarge` instance child nodes; each of these distinct node groups may specify different container images or commands to run for each job.

Alternatively, all of the nodes in your job can use a single node group, and your application code can differentiate node roles (main node vs. child node) by comparing the `AWS_BATCH_JOB_MAIN_NODE_INDEX` environment variable against its own value for `AWS_BATCH_JOB_NODE_INDEX`. You may have up to 1000 nodes in a single job. This is the default limit for instances in an Amazon ECS cluster, which can be increased on request.

**Note**

Currently all node groups in a multi-node parallel job must use the same instance type.
Job Lifecycle

When you submit a multi-node parallel job, the job enters the SUBMITTED status, and it waits for any job dependencies to finish. Then the job moves to the RUNNABLE status, and AWS Batch provisions the instance capacity required to run your job and launches these instances.

Each multi-node parallel job contains a main node. The main node is a single subtask that AWS Batch monitors to determine the outcome of the submitted multi-node job. The main node is launched first and it moves to the STARTING status.

When the main node reaches the RUNNING status (after the node's container is running), the child nodes are launched and they also move to the STARTING status. The child nodes come up in random order. There are no guarantees on the timing or ordering of child node launch. To ensure that the all the nodes of the jobs are in the RUNNING status (after the node's container is running), your application code can either query the AWS Batch API to get the main node and child node information, or coordinate within the application code to wait until all nodes are online before starting any distributed processing task. The private IP address of the main node is available as the AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS environment variable in each child node. Your application code may use this information to coordinate and communicate data between each task.

As individual nodes exit, they move to SUCCEEDED or FAILED, depending on their exit code. If the main node exits, the job is considered finished, and all of the child nodes are stopped. If a child node dies, AWS Batch does not take any action on the other nodes in the job. If you do not want your job to continue with a reduced number of nodes, you must factor this into your application code to terminate or cancel the job.

Compute Environment Considerations

There are several things to consider when configuring compute environments to run multi-node parallel jobs with AWS Batch.

- If you intend to submit multi-node parallel jobs to a compute environment, consider creating a cluster placement group in a single Availability Zone and associating it with your compute resources. This keeps your multi-node parallel jobs on a logical grouping of instances in close proximity with high network flow potential. For more information, see Placement Groups in the Amazon EC2 User Guide for Linux Instances.

- Multi-node parallel jobs are not supported on compute environments that use Spot Instances.

- AWS Batch multi-node parallel jobs use the Amazon ECS awsvpc network mode, which gives your multi-node parallel job containers the same networking properties as Amazon EC2 instances. Each multi-node parallel job container gets its own elastic network interface, a primary private IP address, and an internal DNS hostname. The network interface is created in the same VPC subnet as its host compute resource. Any security groups that are applied to your compute resources are also applied to it. For more information, see Task Networking with the awsvpc Network Mode in the Amazon Elastic Container Service Developer Guide.

- Your compute environment may have no more than five security groups associated with it.

- The elastic network interfaces that are created and attached to your compute resources cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an elastic network interface that is associated with a running job. To release the elastic network interfaces for a task, terminate the job.

- Your compute environment must have enough maximum vCPUs to support your multi-node parallel job.

- Your Amazon EC2 instance limits must be able to satisfy the number of instances required to run your job. For example, if your job requires 30 instances, but your account can only run 20 instances in a Region, your job gets stuck in the RUNNABLE status.
• If you specify an instance type for a node group in a multi-node parallel job, your compute environment must be able to launch that instance type.

GPU Jobs

GPU Jobs help you to run jobs that use an instance's GPUs.

The following Amazon EC2 GPU-based instance types are supported. For more information, see Amazon EC2 G3 Instances, Amazon EC2 G4 Instances, Amazon EC2 P2 Instances, and Amazon EC2 P3 Instances.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>GPUs</th>
<th>GPU Memory</th>
<th>vCPUs</th>
<th>Memory</th>
<th>Network Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>g3s.xlarge</td>
<td>1</td>
<td>8 GiB</td>
<td>4</td>
<td>30.5 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>g3.4xlarge</td>
<td>1</td>
<td>8 GiB</td>
<td>16</td>
<td>122 GiB</td>
<td>Up to 10 Gbps</td>
</tr>
<tr>
<td>g3.8xlarge</td>
<td>2</td>
<td>16 GiB</td>
<td>32</td>
<td>244 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>g3.16xlarge</td>
<td>4</td>
<td>32 GiB</td>
<td>64</td>
<td>488 GiB</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>g4dn.xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>4</td>
<td>16 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g4dn.2xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>8</td>
<td>32 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g4dn.4xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>16</td>
<td>64 GiB</td>
<td>Up to 25 Gbps</td>
</tr>
<tr>
<td>g4dn.8xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>32</td>
<td>128 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>g4dn.12xlarge</td>
<td>4</td>
<td>64 GiB</td>
<td>48</td>
<td>192 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>g4dn.16xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>64</td>
<td>256 GiB</td>
<td>50 Gbps</td>
</tr>
<tr>
<td>p2.xlarge</td>
<td>1</td>
<td>12 GiB</td>
<td>4</td>
<td>61 GiB</td>
<td>High</td>
</tr>
<tr>
<td>p2.8xlarge</td>
<td>8</td>
<td>96 GiB</td>
<td>32</td>
<td>488 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>p2.16xlarge</td>
<td>16</td>
<td>192 GiB</td>
<td>64</td>
<td>732 GiB</td>
<td>20 Gbps</td>
</tr>
<tr>
<td>p3.2xlarge</td>
<td>1</td>
<td>16 GiB</td>
<td>8</td>
<td>61 GiB</td>
<td>Up to 10 Gbps</td>
</tr>
<tr>
<td>p3.8xlarge</td>
<td>4</td>
<td>64 GiB</td>
<td>32</td>
<td>244 GiB</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>p3.16xlarge</td>
<td>8</td>
<td>128 GiB</td>
<td>64</td>
<td>488 GiB</td>
<td>25 Gbps</td>
</tr>
<tr>
<td>p3dn.24xlarge</td>
<td>8</td>
<td>256 GiB</td>
<td>96</td>
<td>768 GiB</td>
<td>100 Gbps</td>
</tr>
</tbody>
</table>

The resourceRequirements parameter for the job definition specifies the number of GPUs to be pinned to the container. This number of GPUs isn’t available to any other job running on that instance for the duration of that job. All instance types in a compute environment that run GPU jobs should be from the p2, p3, g3, g3s, or g4 instance families. If this isn’t done a GPU job might get stuck in the RUNNABLE status.

Jobs that don’t use the GPUs can be run on GPU instances. However, they might cost more to run on the GPU instances than on similar non-GPU instances. Depending on the specific vCPU, memory, and time needed, these non-GPU jobs might block GPU jobs from running.
Job definitions

AWS Batch job definitions specify how jobs are to be run. While each job must reference a job definition, many of the parameters that are specified in the job definition can be overridden at runtime.

Contents
- Creating a job definition (p. 32)
- Creating a multi-node parallel job definition (p. 36)
- Job definition template (p. 39)
- Job definition parameters (p. 43)
- Using the awslogs log driver (p. 61)
- Specifying sensitive data (p. 64)
- Example job definitions (p. 73)

Some of the attributes specified in a job definition include:
- Which Docker image to use with the container in your job
- How many vCPUs and how much memory to use with the container
- The command the container should run when it is started
- What (if any) environment variables should be passed to the container when it starts
- Any data volumes that should be used with the container
- What (if any) IAM role your job should use for AWS permissions

For a complete description of the parameters available in a job definition, see Job definition parameters (p. 43).

Creating a job definition

Before you can run jobs in AWS Batch, you must create a job definition. This process varies slightly for single-node and multi-node parallel jobs. This topic covers creating a job definition for an AWS Batch job that's not a multi-node parallel job.

To create a multi-node parallel job definition, see Creating a multi-node parallel job definition (p. 36).
For more information about multi-node parallel jobs, see Multi-node Parallel Jobs (p. 28).

To create a new job definition
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Job definitions, Create.
4. For Name, enter a unique name for your job definition. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
5. For **Platform**, choose **EC2** if the job runs on EC2 instances, or **Fargate**, if the job runs on AWS Fargate capacity. For more information, see [AWS Batch on AWS Fargate](p. 109).

6. In the **Retry Strategies** section, you can specify the number of times to retry a job. You can also create conditions to decide whether a failed job should be retried. These conditions are based on string matching of the error code and reasons listed for the job attempt. For more information, see [Automated Job Retries](p. 18).

   a. For **Job attempts**, specify the number of times to attempt your job (in case it fails). This number must be between one (1) and ten (10), inclusive.

   b. (Optional) Select **Add evaluate on exit** to add up to five (5) conditions to match string patterns with the exit code, status reason, and reason that is returned in the job attempt. For each set of conditions, **Action** must be set to either **Retry** (to retry until the number of job attempts has been reached), or **Exit** to stop retrying the job.

7. (Optional) For **Execution timeout**, specify the maximum number of seconds you would like to allow your job attempts to run. If an attempt exceeds the timeout duration, it is stopped and the status moves to **FAILED**. For more information, see [Job Timeouts](p. 19).

8. For **Multi-node parallel**, leave this box unchecked. To create a multi-node parallel job definition instead, see [Creating a multi-node parallel job definition](p. 36).

9. In **Container properties**, you can specify properties that are passed to the Docker daemon when the job is placed.

   a. For **Image**, choose the Docker image to use for your job. Images in the Docker Hub registry are available by default. You can also specify other repositories with `repository-url/image:tag`. 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`.

      **Note**
      Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, ARM-based Docker images can only run on ARM-based compute resources.

      - Images in Amazon ECR repositories use the full `registry/repository:tag` naming convention. For example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`
      - 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`).

   b. For **Command**, specify the command to pass to the container. For simple commands, you can type the command as you would at a command prompt in the **Space delimited** tab. Then, verify that the JSON result (which is passed to the Docker daemon) is correct. For more complicated commands (for example, with special characters), you can switch to the **JSON** tab and enter the string array equivalent there.

      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](https://docs.docker.com/engine/reference/builder/#cmd).

      **Note**
      You can use default values for parameter substitution as well as placeholders in your command. For more information, see [Parameters](p. 44).
c. For **vCPUs**, specify the number of vCPUs to 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`. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

d. For **Memory**, specify the hard limit (in MiB) of memory to present to the job's container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the `--memory` option to `docker run`. You must specify at least 4 MiB of memory for a job.

**Note**
You can maximize your resource utilization by prioritizing memory for jobs of a specific instance type. For more information about how to do this, see Compute Resource Memory Management (p. 107).

e. (Optional) For **Number of GPUs**, specify the number of GPUs your job uses.

The job runs on a container with the specified number of GPUs pinned to that container.

f. In the **Additional configuration** section, you can specify additional parameters to be used with the container.

i. (Optional) For **Job role**, you can specify an IAM role that provides the container in your job with permissions to use the AWS APIs. This feature uses Amazon ECS IAM roles for task functionality. For more information, including configuration prerequisites, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

**Note**
A job role is required for jobs that are running on Fargate resources.

**Note**
Only roles that have the Amazon Elastic Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your AWS Batch jobs, see Creating an IAM Role and Policy for your Tasks in the Amazon Elastic Container Service Developer Guide.

ii. For **Execution role**, you can specify an IAM role that grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf. This feature uses Amazon ECS IAM roles for task functionality. For more information, including configuration prerequisites, see Amazon ECS task execution IAM roles in the Amazon Elastic Container Service Developer Guide.

**Note**
An execution role is required for jobs running on Fargate resources.

iii. (Optional) In the **Volumes** section, you can specify data volumes for your job to pass to your job's container.

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

B. (Optional) For **Source Path**, enter the path on the host instance to present to the container. If you leave this field empty, then the Docker daemon assigns a host path for you. If you specify a source path, then 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.

iv. (Optional) In the **Mount points** section, you can configure mount points for your job's container to access.

A. For **Container path**, enter the path on the container at which to mount the host volume.

B. For **Source volume**, enter the name of the volume to mount.
C. To make the volume read-only for the container, choose **Read-only**.

v. (Optional) In the **Ulimits** section, you can configure any `ulimit` values to use for your job's container.

   A. Choose **Add limit**.
   
   B. For **Limit name**, choose a `ulimit` to apply.
   
   C. For **Soft limit**, choose the soft limit to apply for the `ulimit` type.
   
   D. For **Hard limit**, choose the hard limit to apply for the `ulimit` type.

vi. (Optional) In the **Environment variables** section, you can specify environment variables to pass to your job's 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 don't recommend using plaintext environment variables for sensitive information, such as credential data.

   A. Choose **Add environment variable**.
   
   B. For **Key**, specify the key for your environment variable.

   **Note**
   
   Environment variables must not start with `AWS_BATCH`; this naming convention is reserved for variables that are set by the AWS Batch service.

   C. For **Value**, specify the value for your environment variable.

vii. (Optional) In the **Security** section, you can configure security options for your job's container.

   A. To give your job's container elevated privileges on the host instance (similar to the root user), select **Privileged**. This parameter maps to `Privileged` in the Create a container section of the Docker Remote API and the `--privileged` option to `docker run`.
   
   B. For **User**, enter the user name to use inside the container. This parameter maps to `User` in the Create a container section of the Docker Remote API and the `--user` option to `docker run`.

viii. (Optional) In the **Linux Parameters** section, you can configure any device mappings to use for your job's container. This allows the container to be able to access a device on the host instance.

   A. In the **Devices** section, choose **Add device**.
   
   B. For **Host path**, specify the path of a device in the host instance.
   
   C. For **Container path**, specify the path of in the container instance to expose the device mapped to the host instance. If this is left blank (unspecified), then the host path is used in the container.
   
   D. For **Permissions**, choose one or more permissions to apply to the device in the container. The available permissions are `READ`, `WRITE`, and `MKNOD`.

10. (Optional) In the **Parameters** section, you can specify parameter substitution default values and placeholders to use in the command that your job's container runs when it starts. For more information, see Parameters (p. 144).

   a. Choose **Add parameter**.
   
   b. For **Key**, specify the key for your parameter.
   
   c. For **Value**, specify the value for your parameter.

11. (Optional) In the **Tags** section, you can specify the key and value for each tag to associate with the job definition. For more information, see Tagging your AWS Batch resources (p. 163).

12. Choose **Create job definition**.
Creating a multi-node parallel job definition

Before you can run jobs in AWS Batch, you must create a job definition. This process varies slightly for single-node and multi-node parallel jobs. This topic covers creating a job definition for an AWS Batch multi-node parallel job. For more information, see Multi-node Parallel Jobs (p. 28).

**Note**
AWS Fargate doesn't support multi-node parallel jobs.

To create a single-node job definition, see Creating a job definition (p. 32).

**To create a multi-node parallel job definition**

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose "Job definitions, Create."
4. For **Name**, enter a unique name for your job definition. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
5. For **Platform**, choose "EC2."
6. In the **Retry Strategies** section, you can specify the number of times to retry a job. You can also create conditions to decide whether a failed job should be retried. This is based on string matching of the error code and reasons listed for the job attempt. For more information, see Automated Job Retries (p. 18).
   a. For **Job attempts**, specify the number of times to attempt your job (in case it fails). This number must be between one (1) and ten (10), inclusive.
   b. (Optional) Select Add evaluate on exit to add up to five (5) conditions to match string patterns with the exit code, status reason, and reason that is returned in the job attempt. For each set of conditions, **Action** must be set to either **Retry** (to retry until the number of job attempts has been reached), or **Exit** to stop retrying the job.
7. (Optional) For **Execution timeout**, specify the maximum number of seconds you would like to allow your job attempts to run. If an attempt exceeds the timeout duration, it is stopped and the status moves to FAILED. For more information, see Job Timeouts (p. 19).
8. For **Multi-node parallel**, select Enable multi-node parallel and then complete the following substeps. To create a single node parallel job definition instead, see Creating a job definition (p. 32).
   a. For **Number of nodes**, enter the total number of nodes to use for your job.
   b. For **Main node**, enter the node index to use for the main node. The default main node index is 0.
   c. Select **Add node range**. This creates a **Node range** section.
      i. For **Target nodes**, specify the range for your node group, using range_start:range_end notation.
         You can create up to five node ranges for the number of nodes you specified for your job. Node ranges use the index value for a node, and the node index begins at 0. The range end index value of your final node group should be the number of nodes you specified in Step 8.a (p. 36), minus one. For example, if you specified 10 nodes, and you want to use a single node group, then your end range should be 9.
      ii. In **Container properties**, you can specify properties that are passed to the Docker daemon for the nodes in the node range.
         A. For **Image**, choose the Docker image to use for your job. Images in the Docker Hub registry are available by default. You can also specify other repositories with repository-url/image:tag. 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.

**Note**
Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, ARM-based Docker images can only run on ARM-based compute resources.

- Images in Amazon ECR repositories use the full registry/repository:tag naming convention. For example, aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest
- 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).

B. For **Command**, specify the command to pass to the container. For simple commands, you can type the command as you would at a command prompt in the Space delimited tab. Then, verify that the JSON result (which is passed to the Docker daemon) is correct. For more complicated commands (for example, with special characters), you can switch to the JSON tab and enter the string array equivalent there.

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.

**Note**
You can use default values for parameter substitution and placeholders in your command. For more information, see Parameters (p. 44).

C. For **vCPUs**, specify the number of vCPUs to 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. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU.

D. For **Memory**, specify the hard limit (in MiB) of memory to present to the job's container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run. You must specify at least 4 MiB of memory for a job.

**Note**
If you're trying to maximize your resource utilization by providing your jobs as much memory as possible for a particular instance type, see Compute Resource Memory Management (p. 107).

E. (Optional) For **Number of GPUs**, specify the number of GPUs your job uses.

The job runs on a container with the specified number of GPUs pinned to that container.

F. In the **Additional configuration** section, you can specify additional parameters to be used with the container.

I. (Optional) For **Job role**, you can specify an IAM role that provides the container in your job with permissions to use the AWS APIs. This feature uses Amazon ECS IAM roles for task functionality. For more information, including configuration prerequisites, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.
Creating a multi-node parallel job definition

**Note**
A job role is required for jobs that are running on Fargate resources.

**Note**
Only roles that have the Amazon Elastic Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your AWS Batch jobs, see Creating an IAM Role and Policy for your Tasks in the Amazon Elastic Container Service Developer Guide.

II. (Optional) In the **Volumes** section, you can specify data volumes for your job to pass to your job's container.

1. For **Name**, enter a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. (Optional) For **Source Path**, enter the path on the host instance to present to the container. If you leave this field empty, then the Docker daemon assigns a host path for you. If you specify a source path, then 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.

III. (Optional) In the **Mount points** section, you can configure mount points for your job's container to access.

1. For **Container path**, enter the path on the container at which to mount the host volume.
2. For **Source volume**, enter the name of the volume to mount.
3. To make the volume read-only for the container, choose **Read-only**.

IV. (Optional) In the **Ulimits** section, you can configure any ulimit values to use for your job's container.

1. Choose **Add limit**.
2. For **Limit name**, choose a ulimit to apply.
3. For **Soft limit**, choose the soft limit to apply for the ulimit type.
4. For **Hard limit**, choose the hard limit to apply for the ulimit type.

V. (Optional) In the **Environment variables** section, you can specify environment variables to pass to your job's 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 don't recommend using plaintext environment variables for sensitive information, such as credential data.

1. Choose **Add environment variable**.
2. For **Key**, specify the key for your environment variable.

**Note**
Environment variables must not start with `AWS_BATCH`; this naming convention is reserved for variables that are set by the AWS Batch service.

3. For **Value**, specify the value for your environment variable.

VI. (Optional) In the **Security** section, you can configure security options for your job's container.

1. To give your job's container elevated privileges on the host instance (similar to the root user), select **Privileged**. This parameter maps to `Privileged` in the
Create a container section of the Docker Remote API and the --privileged option to docker run.

2. For User, enter the user name to use inside the container. This parameter maps to User in the Create a container section of the Docker Remote API and the --user option to docker run.

VII. (Optional) In the Linux Parameters section, you can configure any device mappings to use for your job's container so that the container can access a device on the host instance.

1. In the Devices section, choose Add device.
2. For Host path, specify the path of a device in the host instance.
3. For Container path, specify the path of in the container instance to expose the device mapped to the host instance. If this is left blank then the host path is used in the container.
4. For Permissions, choose one or more permissions to apply to the device in the container. The available permissions are READ, WRITE, and MKNOD.

9. Return to Step 8.c.i (p. 36) and repeat for each node group to configure for your job.

10. (Optional) In the Parameters section, you can specify parameter substitution default values and placeholders to use in the command that your job's container runs when it starts. For more information, see Parameters (p. 44).

   a. Choose Add parameter.
   b. For Key, specify the key for your parameter.
   c. For Value, specify the value for your parameter.

11. (Optional) In the Tags section, you can specify the key and value for each tag to associate with the job definition. For more information, see Tagging your AWS Batch resources (p. 163).

12. Choose Create job definition.

Job definition template

The following is an empty job definition template. You can use this template to create your job definition, which can then be saved to a file and used with the AWS CLI --cli-input-json option. For more information about these parameters, see Job definition parameters (p. 43).

```json
{
   "jobDefinitionName": "",
   "type": "container",
   "parameters": {
      "KeyName": ""
   },
   "containerProperties": {
      "image": "",
      "vcpus": 0,
      "memory": 0,
      "command": [
         ""
      ],
      "jobRoleArn": "",
      "executionRoleArn": "",
      "volumes": [
         {
            "host": {
               "sourcePath": ""
            },
            "name": ""
         }
      ]
}
```
"environment": [ 
  
  ],
  "mountPoints": [ 
    
  ],
  "readonlyRootFilesystem": true,
  "privileged": true,
  "ulimits": [ 
    
  ],
  "user": "",
  "instanceType": "",
  "resourceRequirements": [ 
    
  ],
  "linuxParameters": { 
    "devices": [ 
    ]
  },
  "initProcessEnabled": true,
  "sharedMemorySize": 0,
  "tmpfs": [ 
    
  ],
  "maxSwap": 0,
  "swappiness": 0
},
"logConfiguration": { 
  "logDriver": "journald",
  "options": { 
    "KeyName": ""
  },
  "secretOptions": [ 
    
  ]}
null
],
  "linuxParameters": {
    "devices": [
      {
        "hostPath": "",
        "containerPath": "",
        "permissions": [
          "MKNOD"
        ]
      }
    ],
    "initProcessEnabled": true,
    "sharedMemorySize": 0,
    "tmpfs": [
      {
        "containerPath": "",
        "size": 0,
        "mountOptions": [
          ""
        ]
      }
    ],
    "maxSwap": 0,
    "swappiness": 0
  },
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "KeyName": ""
    },
    "secretOptions": [
      {
        "name": "",
        "valueFrom": ""
      }
    ],
    "secrets": [
      {
        "name": "",
        "valueFrom": ""
      }
    ],
    "networkConfiguration": {
      "assignPublicIp": "DISABLED"
    },
    "fargatePlatformConfiguration": {
      "platformVersion": ""
    }
  },
  "retryStrategy": {
    "attempts": 0,
    "evaluateOnExit": [
      {
        "onStatusReason": "",
        "onReason": "",
        "onExitCode": "",
        "action": "EXIT"
      }
    ],
    "propagateTags": true,
    "timeout": {
      "seconds": 42
    }
  }
}
### Job definition parameters

Job definitions are split into four basic parts: the job definition name, the type of the job definition, parameter substitution placeholder defaults, and the container properties for the job.

#### Contents

- Job definition name (p. 43)
- Type (p. 43)
- Parameters (p. 44)
- Platform capabilities (p. 44)
- Propagate tags (p. 44)
- Container properties (p. 45)
- Node properties (p. 58)
- Retry strategy (p. 59)
- Tags (p. 61)
- Timeout (p. 61)

#### Job definition name

**jobDefinitionName**

When you register a job definition, you specify a name. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. The first job definition that's registered with that name is given a revision of 1. Any subsequent job definitions that are registered with that name are given an incremental revision number.

Type: String

Required: Yes

#### Type

**type**

When you register a job definition, you specify the type of job. If the job runs on Fargate resources, then `multinode` isn't supported. For more information about multi-node parallel jobs, see the section called "Creating a multi-node parallel job definition" (p. 36).
Parameters

When you submit a job, you can specify parameters that should replace the placeholders or override the default job definition parameters. Parameters in job submission requests take precedence over the defaults in a job definition. This allows you to use the same job definition for multiple jobs that use the same format, and programmatically change values in the command at submission time.

Type: String to string map

When you register a job definition, you can use parameter substitution placeholders in the command field of a job's container properties. For example:

```
"command": [ "ffmpeg", "-i", "Ref::inputfile", "-c", "Ref::codec", "-o", "Ref::outputfile" ]
```

In the above example, there are `Ref::inputfile`, `Ref::codec`, and `Ref::outputfile` parameter substitution placeholders in the command. The `parameters` object in the job definition allows you to set default values for these placeholders. For example, to set a default for the `Ref::codec` placeholder, you specify the following in the job definition:

```
"parameters": {"codec": "mp4"}
```

When this job definition is submitted to run, the `Ref::codec` argument in the container's command is replaced with the default value, `mp4`.

Platform capabilities

The platform capabilities required by the job definition. If no value is specified, it defaults to EC2. Jobs run on Fargate resources specify FARGATE.

Type: String

Valid values: EC2 | FARGATE

Required: No

Propagate tags

Specifies whether to propagate the tags from the job or job definition to the corresponding Amazon ECS task. If no value is specified, the tags aren't propagated. Tags can only be propagated to
the tasks during task creation. For tags with the same name, job tags are given priority over job definitions tags. If the total number of combined tags from the job and job definition is over 50, the job's moved to the FAILED state.

Type: Boolean

Required: No

**Container properties**

When you register a job definition, you must specify a list of container properties that are passed to the Docker daemon on a container instance when the job is placed. The following container properties are allowed in a job definition. For single-node jobs, these container properties are set at the job definition level. For multi-node parallel jobs, container properties are set in the Node properties (p. 58) level, for each node group.

**command**

The command that's passed to the container. This parameter maps to `Cmd` in the Create a container section of the Docker Remote API and the `COMMAND` parameter to `docker run`. For more information about the Docker `CMD` parameter, see [https://docs.docker.com/engine/reference/builder/#cmd](https://docs.docker.com/engine/reference/builder/#cmd).

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

Type: String array

Required: No

**environment**

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

**Note**

Environment variables must not start with `AWS_BATCH`; this naming convention is reserved for variables that are set by the AWS Batch service.

Type: Array of key-value pairs

Required: No

**name**

The name of the environment variable.

Type: String

Required: Yes, when environment is used.

**value**

The value of the environment variable.

Type: String

Required: Yes, when environment is used.
"environment" : [
    { "name" : "envName1", "value" : "envValue1" },
    { "name" : "envName2", "value" : "envValue2" }
]

executionRoleArn

When you register a job definition, you can specify an IAM role. The role provides the Amazon ECS container agent with permissions to call the API actions that are specified in its associated policies on your behalf. Jobs that are running on Fargate resources must provide an execution role. For more information, see AWS Batch execution IAM role (p. 157).

Type: String
Required: No

fargatePlatformConfiguration

The platform configuration for jobs that are running on Fargate resources. Jobs that are running on EC2 resources must not specify this parameter.

Type: FargatePlatformConfiguration object
Required: No

platformVersion

The AWS Fargate platform version use for the jobs, or LATEST to use a recent, approved version of the AWS Fargate platform.

Type: String
Default: LATEST
Required: No

image

The image used to start a job. 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 repository-url/image:tag. 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.

Note

Docker image architecture must match the processor architecture of the compute resources that they're scheduled on. For example, ARM-based Docker images can only run on ARM-based compute resources.

- Images in Amazon ECR repositories use the full registry/repository:tag naming convention. For example, aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest
- 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).

Type: String
Required: Yes
instanceType

The instance type to use for a multi-node parallel job. All node groups in a multi-node parallel job must use the same instance type. This parameter isn't valid for single-node container jobs or for jobs run on Fargate resources.

Type: String
Required: No

jobRoleArn

When you register a job definition, you can specify an IAM role. The role provides the job container with permissions to call the API actions that are specified in its associated policies on your behalf. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

Type: String
Required: No

linuxParameters

Linux-specific modifications that are applied to the container, such as details for device mappings.

```
"linuxParameters": { 
  "devices": [ 
    { 
      "hostPath": "string",
      "containerPath": "string",
      "permissions": [ 
        "READ", "WRITE", "MKNOD"
      ]
    },
    "initProcessEnabled": true|false,
    "sharedMemorySize": 0,
    "tmpfs": [ 
      { 
        "containerPath": "string",
        "size": integer,
        "mountOptions": [ 
          "string"
        ]
      }
    ],
    "maxSwap": integer,
    "swappiness": integer
  }
}
```

Type: LinuxParameters object
Required: No

devices

List of devices mapped into 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**

This parameter isn't applicable to jobs running on Fargate resources and shouldn't be provided.

Type: Array of Device objects
Required: No

**hostPath**

Path at which the device available in the host container instance.

Type: String

Required: Yes

**containerPath**

Path at which the device is exposed in the container. If this isn't specified the device is exposed at the same path as the host path.

Type: String

Required: No

**permissions**

Permissions for the device in the container. If this isn't specified the permissions are set to READ, WRITE, and MKNOD.

Type: Array of strings

Required: No

Valid values: READ | WRITE | MKNOD

**initProcessEnabled**

If true, 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. To check the Docker Remote API version on your container instance, log into your container instance and run the following command: sudo docker version | grep "Server API version"

Type: Boolean

Required: No

**maxSwap**

The total amount of swap memory (in MiB) a job can use. This parameter is translated to the --memory-swap option to docker run where the value is the sum of the container memory plus the maxSwap value. For more information, see --memory-swap details in the Docker documentation.

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

**Note**

This parameter isn't applicable to jobs running on Fargate resources and shouldn't be provided.

Type: Integer

Required: No

**sharedMemorySize**

The value for the size (in MiB) of the /dev/shm volume. This parameter maps to the --shm-size option to docker run.
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 isn't specified, a default value of 60 is used. If a value isn't specified for maxSwap then this parameter is ignored. If maxSwap is set to 0, the container doesn't use swap. This parameter maps to the --memory-swappiness option to docker run.

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 don't 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 parameters are only supported for job definitions using EC2 resources.
- If the maxSwap and swappiness parameters are omitted from a job 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.

Note

This parameter isn't applicable to jobs running on Fargate resources and shouldn't be provided.

tmpfs

The container path, mount options, and size of the tmpfs mount.

Type: Array of Tmpfs objects

Note

This parameter isn't applicable to jobs running on Fargate resources and shouldn't be provided.

Required: No
Container properties

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"

Type: Array of strings

Required: No

size

The size (in MiB) of the tmpfs volume.

Type: Integer

Required: Yes

logConfiguration

The log configuration specification for the job.

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 can 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 either configured on the container instance or on another log server to provide remote logging options. For more information about the options for different supported log drivers, see Configure logging drivers in the Docker documentation.

Note

AWS Batch currently supports a subset of the logging drivers available to the Docker daemon (shown in the LogConfiguration data type).

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance and run the following command: sudo docker version | grep "Server API version"

```
"logConfiguration": { 
  "devices": [
    {
      "logDriver": "string",
      "options": {
        "optionName1": "optionValue1",
        "optionName2": "optionValue2"
      }
    },
    {
      "name": "secretOptionName1",
      "valueFrom": "secretOptionArn1"
    },
    {
      "name": "secretOptionName2",
      "valueFrom": "secretOptionArn2"
    }
  ]
}
```

Type: LogConfiguration object
Container properties

Required: No

**logDriver**

The log driver to use for the job. By default, AWS Batch enables the `awslogs` log driver. The valid values listed for this parameter are log drivers that the Amazon ECS container agent can communicate with by default.

This parameter maps to `LogConfig` in the [Create a container](https://docs.docker.com/engine/api/v1.41/#create-container) section of the [Docker Remote API](https://docs.docker.com/api/remote/) and the `--log-driver` option to `docker run`. By default, jobs use the same logging driver that the Docker daemon uses. However, the job can use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the job definition. If you want to specify another logging driver for a job, then the log system must be configured on the container instance in the compute environment. Or, alternatively, you should configure it on another log server to provide remote logging options. For more information about the options for different supported log drivers, see [Configure logging drivers](https://docs.docker.com/engine/configuration/logging/) in the Docker documentation.

**Note**

AWS Batch currently supports a subset of the logging drivers available to the Docker daemon. Additional log drivers might be available in future releases of the Amazon ECS container agent.

The supported log drivers are `awslogs`, `fluentd`, `gelf`, `json-file`, `journald`, `logentries`, `syslog`, and `splunk`.

**Note**

Jobs that are running on Fargate resources are restricted to the `awslogs` and `splunk` log drivers.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance. To check the Docker Remote API version on your container instance, log into your container instance and run the following command: `sudo docker version | grep "Server API version"

**Note**

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. Otherwise, the containers placed on that instance can't use these log configuration options. For more information, see [Amazon ECS Container Agent Configuration](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/container-agent-configuring.html) in the [Amazon Elastic Container Service Developer Guide](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/)

**awslogs**

Specifies the Amazon CloudWatch Logs logging driver. For more information, see [Using the `awslogs` log driver](https://docs.docker.com/config/containers/logging/awslogs/) and [Amazon CloudWatch Logs logging driver](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using-awslogs-log-driver.html) in the Docker documentation.

**fluentd**

Specifies the Fluentd logging driver. For more information, including usage and options, see [Fluentd logging driver](https://docs.docker.com/config/containers/logging/fluentd/) in the Docker documentation.

**gelf**

Specifies the Graylog Extended Format (GELF) logging driver. For more information, including usage and options, see [Graylog Extended Format logging driver](https://docs.docker.com/config/containers/logging/gelf/) in the Docker documentation.

**journald**

Specifies the journald logging driver. For more information, including usage and options, see [Journald logging driver](https://docs.docker.com/config/containers/logging/journald/) in the Docker documentation.
json-file

Specifies the JSON file logging driver. For more information, including usage and options, see JSON File logging driver in the Docker documentation.

splunk

Specifies the Splunk logging driver. For more information, including usage and options, see Splunk logging driver in the Docker documentation.

glog

Specifies the syslog logging driver. For more information, including usage and options, see Syslog logging driver in the Docker documentation.

Type: String

Required: Yes

Valid values: awslogs | fluentd | gelf | journald | json-file | splunk | syslog

**Note**

If you have a custom driver that's not listed earlier that you would like to work with the Amazon ECS container agent, you can fork the Amazon ECS container agent project that's available on GitHub and customize it to work with that driver. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support that are running modified copies of this software.

**options**

Log configuration options to send to a log driver for the job.

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

Type: String to string map

Required: No

**secretOptions**

An object representing the secret to pass to the log configuration. For more information, see Specifying sensitive data (p. 64).

Type: object array

Required: No

**name**

The name of the log driver option to set in the job.

Type: String

Required: Yes

**valueFrom**

The ARN of the secret to expose to the log configuration of the container. The supported values are either the full ARN of the Secrets Manager secret or the full ARN of the parameter in the SSM Parameter Store.

**Note**

If the SSM Parameter Store parameter exists in the same Region as the task you're 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.
## Container properties

### memory

Type: String
Required: Yes

This parameter is deprecated and not supported for jobs run on Fargate resources. Use ResourceRequirement instead. For jobs run on EC2 resources that aren't using ResourceRequirement, the number of MiB of memory reserved for the job. For other jobs, see resourceRequirements. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run. You must specify at least 4 MiB of memory for a job. This is required but can be specified in several places for multi-node parallel (MNP) jobs. It must be specified for each node at least once.

**Note**
If you’re trying to maximize your resource utilization by providing your jobs as much memory as possible for a particular instance type, see Compute Resource Memory Management (p. 107).

Type: Integer
Required: Yes

### mountPoints

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.

```
"mountPoints": [
  {
    "sourceVolume": "string",
    "containerPath": "string",
    "readOnly": true|false
  }
]
```

Type: Object array
Required: No

#### sourceVolume

The name of the volume to mount.

Type: String
Required: Yes, when mountPoints is used.

#### containerPath

The path on the container at which to mount the host volume.

Type: String
Required: Yes, when mountPoints is used.

#### readOnly

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.

Type: Boolean
Required: No
networkConfiguration

The network configuration for jobs that are running on Fargate resources. Jobs that are running on EC2 resources must not specify this parameter.

"networkConfiguration": {  
  "assignPublicIp": "string"  
}

Type: Object array

assignPublicIp

Indicates whether the job should have a public IP address. This is required if the job needs outbound network access.

Type: String

Valid values: ENABLED | DISABLED

Required: No

Default: DISABLED

privileged

When this parameter is true, the container is given elevated permissions 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. This parameter isn't applicable to jobs running on Fargate resources and shouldn't be provided, or specified as false.

"privileged": true/false

Type: Boolean

Required: No

readonlyRootFilesystem

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.

"readonlyRootFilesystem": true/false

Type: Boolean

Required: No

resourceRequirements

The type and amount of a resource to assign to a container. The supported resources include GPU, MEMORY, and VCPU.

"resourceRequirements": [  
  {  
    "type": "GPU",  
  }  
]
"value": "number"
}
]

Type: Object array
Required: No
type

The type of resource to assign to a container. The supported resources include GPU, MEMORY, and VCPU.
Type: String
Required: Yes, when resourceRequirements is used.
value

The quantity of the specified resource to reserve for the container. The values vary based on the type specified.
type="GPU"

The number of physical GPUs to reserve for the container. The number of GPUs reserved for all containers in a job shouldn’t exceed the number of available GPUs on the compute resource that the job is launched on.
type="MEMORY"

The hard limit (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run. You must specify at least 4 MiB of memory for a job. This is required but can be specified in several places for multi-node parallel (MNP) jobs; it must be specified for each node at least once. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run.

**Note**
If you’re trying to maximize your resource utilization by providing your jobs as much memory as possible for a particular instance type, see Compute Resource Memory Management (p. 107).

For jobs that are running on Fargate resources, then value must match one of the supported values. Moreover, the VCPU values must be one of the values supported for that memory value.

<table>
<thead>
<tr>
<th>VCPU</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25 vCPU</td>
<td>512, 1024, and 2048 MiB</td>
</tr>
<tr>
<td>0.5 vCPU</td>
<td>1024, 2048, 3072, and 4096 MiB</td>
</tr>
<tr>
<td>1 vCPU</td>
<td>2048, 3072, 4096, 5120, 6144, 7168, and 8192 MiB</td>
</tr>
<tr>
<td>2 vCPU</td>
<td>4096, 5120, 6144, 7168, 8192, 9216, 10240, 11264, 12288, 13312, 14336, 15360, and 16384 MiB</td>
</tr>
<tr>
<td>4 vCPU</td>
<td>8192, 9216, 10240, 11264, 12288, 13312, 14336, 15360, 16384, 17408, 18432, 19456, 20480, 21504, 22528, 23552, 24576, 25600, 26624, 27648, 28672, 29696, and 30720 MiB</td>
</tr>
</tbody>
</table>
type="VCPU"

The number of vCPUs reserved for the job. This parameter maps to CpuShares in the Create a container section of the Docker Remote API and the --cpu-shares option to docker run. Each vCPU is equivalent to 1,024 CPU shares. For jobs that are running on EC2 resources, you must specify at least one vCPU. This is required but can be specified in several places; it must be specified for each node at least once.

For jobs that are running on Fargate resources, then value must match one of the supported values and the MEMORY values must be one of the values supported for that VCPU value. The supported values are 0.25, 0.5, 1, 2, and 4

Type: String

Required: Yes, when resourceRequirements is used.

secrets

The secrets for the job that are exposed as environment variables. For more information, see Specifying sensitive data (p. 64).

"secrets": [
  {
    "name": "secretName1",
    "valueFrom": "secretArn1"
  },
  {
    "name": "secretName2",
    "valueFrom": "secretArn2"
  }
...]

Type: Object array

Required: No

name

The name of the environment variable that contains the secret.

Type: String

Required: Yes, when secrets is used.

valueFrom

The secret to expose to the container. The supported values are either the full ARN of the Secrets Manager secret or the full ARN of the parameter in the SSM Parameter Store.

Note

If the SSM Parameter Store parameter exists in the same Region as the job you’re 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.

Type: String

Required: Yes, when secrets is used.

ulimits

A list of ulimits values to set in the container. This parameter maps to Ulimits in the Create a container section of the Docker Remote API and the --ulimit option to docker run.
"ulimits": [
  {
    "name": "string",
    "softLimit": integer,
    "hardLimit": integer
  }
  ...
]  

Type: Object array
Required: No

name

The type of the ulimit.
Type: String
Required: Yes, when ulimits is used.

hardLimit

The hard limit for the ulimit type.
Type: Integer
Required: Yes, when ulimits is used.

softLimit

The soft limit for the ulimit type.
Type: Integer
Required: Yes, when ulimits is used.

user

The user name to use inside the container. This parameter maps to User in the Create a container section of the Docker Remote API and the --user option to docker run.

"user": "string"

Type: String
Required: No

vcpus

The number of vCPUs reserved 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. Each vCPU is equivalent to 1,024 CPU shares. You must specify at least one vCPU. This is required but can be specified in several places for multi-node parallel (MNP) jobs; it must be specified for each node at least once. Jobs that are running on Fargate resources must specify the vCPU requirement for the job using resourceRequirements. Other jobs can specify the vCPU requirement for the job using resourceRequirements.

Type: Integer
Required: Yes
volumes

When you register a job definition, you can specify a list of volumes that are passed to the Docker daemon on a container instance. The following parameters are allowed in the container properties:

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

**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`.

Type: String

Required: No

**host**

The contents of the `host` parameter determine whether your 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. However, the data isn't guaranteed to persist after the container associated with it stops running.

**Note**

This parameter isn't applicable to jobs running on Fargate resources and shouldn't be provided.

Type: Object

Required: No

**sourcePath**

The path on the host container instance that's 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 container instance until you delete it manually. If the `sourcePath` value 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.

Type: String

Required: No

---

**Node properties**

`nodeProperties`

When you register a multi-node parallel job definition, you must specify a list of node properties. These node properties should define the number of nodes to use in your job, the main node index, and the different node ranges to use. If the job runs on Fargate resources, then you cannot specify
nodeProperties. Rather, you should use containerProperties instead. The following node properties are allowed in a job definition. For more information, see Multi-node Parallel Jobs (p. 28).

Type: NodeProperties object

Required: No

mainNode

Specifies the node index for the main node of a multi-node parallel job. This node index value must be fewer than the number of nodes.

Type: Integer

Required: Yes

numNodes

The number of nodes associated with a multi-node parallel job.

Type: Integer

Required: Yes

nodeRangeProperties

A list of node ranges and their properties associated with a multi-node parallel job.

Type: Array of NodeRangeProperty objects

Required: Yes

targetNodes

The range of nodes, using node index values. A range of 0:3 indicates nodes with index values of 0 through 3. If the starting range value is omitted (:n), then 0 is used to start the range. If the ending range value is omitted (n:), then the highest possible node index is used to end the range. Your accumulative node ranges must account for all nodes (0:n). You can nest node ranges, for example 0:10 and 4:5, in which case the 4:5 range properties override the 0:10 properties.

Type: String

Required: No

container

The container details for the node range. For more information, see Container properties (p. 45).

Type: ContainerProperties object

Required: No

## Retry strategy

retryStrategy

When you register a job definition, you can optionally specify a retry strategy to use for failed jobs that are submitted with this job definition. Any retry strategy that's specified during a SubmitJob operation overrides the retry strategy defined here. By default, each job is attempted one time. If you specify more than one attempt, the job is retried if it fails. Examples of a fail attempt include the
job returns a non-zero exit code or the container instance is terminated. For more information, see Automated job retries.

Type: RetryStrategy object

Required: No

attempts

The number of times to move a job to the RUNNABLE status. You can specify between 1 and 10 attempts. If attempts is greater than one, the job is retried that many times if it fails, until it has moved to RUNNABLE.

"attempts": integer

Type: Integer

Required: No

evaluateOnExit

Array of up to 5 objects that specify conditions under which the job should be retried or failed. If this parameter is specified, then the attempts parameter must also be specified.

"evaluateOnExit": [
  {
    "action": "string",
    "onExitCode": "string",
    "onReason": "string",
    "onStatusReason": "string"
  }
]

Type: Array of EvaluateOnExit objects

Required: No

action

Specifies the action to take if all of the specified conditions (onStatusReason, onReason, and onExitCode) are met. The values aren't case sensitive.

Type: String

Required: Yes

Valid values: RETRY | EXIT

onExitCode

Contains a glob pattern to match against the decimal representation of the ExitCode that's returned for a job. The pattern can be up to 512 characters long. It can contain only numbers (not letters or other special characters). It can optionally end with an asterisk (*) so that only the start of the string needs to be an exact match.

Type: String

Required: No

onReason

Contains a glob pattern to match against the Reason that's returned for a job. The pattern can be up to 512 characters long. It can contain letters, numbers, periods (.), colons (:), and
white space (spaces, tabs). It can optionally end with an asterisk (*) so that only the start of the string needs to be an exact match.

Type: String
Required: No

onStatusReason

Contains a glob pattern to match against the StatusReason that's returned for a job. The pattern can be up to 512 characters long. It can contain letters, numbers, periods (.), colons (:), and white space (spaces, tabs), and can optionally end with an asterisk (*) so that only the start of the string needs to be an exact match.

Type: String
Required: No

Tags

tags

Key-value pair tags to associate with the job definition. For more information, see Tagging your AWS Batch resources (p. 163).

Type: String to string map
Required: No

Timeout

timeout

You can configure a timeout duration for your jobs so that if a job runs longer than that, AWS Batch terminates the job. For more information, see Job Timeouts (p. 19). If a job is terminated due to a timeout, it isn't retried. Any timeout configuration that's specified during a SubmitJob operation overrides the timeout configuration defined here. For more information, see Job Timeouts (p. 19).

Type: JobTimeout object
Required: No

attemptDurationSeconds

The time duration in seconds (measured from the job attempt's startedAt timestamp) after which AWS Batch terminates unfinished jobs. The minimum value for the timeout is 60 seconds.

Type: Integer
Required: No

Using the awslogs log driver

AWS Batch automatically enables the awslogs log driver to send log information to CloudWatch Logs. You can use this feature to view different logs from your containers in one convenient location and prevent your container logs from taking up disk space on your container instances. This topic helps you configure the awslogs log driver in your job definitions.
Note
The type of information that's logged by the containers in your job 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 about how Docker logs are processed, including alternative ways to capture different file data or streams, see View logs for a container or service in the Docker documentation.

To send system logs from your container instances to CloudWatch Logs, see Using CloudWatch Logs with AWS Batch (p. 141). For more information about CloudWatch Logs, see Monitoring Log Files and CloudWatch Logs quotas in the Amazon CloudWatch Logs User Guide.

Available awslogs log driver options

The awslogs log driver supports the following options in AWS Batch job definitions. For more information, see CloudWatch Logs logging driver in the Docker documentation.

awslogs-region

Required: No

Specify the Region to which the awslogs log driver should send your Docker logs. By default, the Region that is used is the same one as the job. You can choose to send all of your logs from jobs in different Regions to a single Region in CloudWatch Logs. Doing this allows them to be visible all from one location. Alternatively, you can separate them by Region for more granular approach. However, when you choose this option, make sure that the specified log groups exists in the Region that you specified.

awslogs-group

Required: Optional

The awslogs-group option allows you to specify the log group to which the awslogs log driver sends its log streams. If it's not specified, aws/batch/job is used.

awslogs-stream-prefix

Required: Optional

The awslogs-stream-prefix option allows you to associate a log stream with the specified prefix, and the Amazon ECS task ID of the AWS Batch job to which the container belongs. If you specify a prefix with this option, then the log stream takes the following format:

prefix-name/default/ecs-task-id

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` in the Docker documentation.

This option is ignored if `awslogs-datetime-format` is also configured.

**Note**
Multiline logging performs regular expression parsing and matching of all log messages. This may have a negative impact on logging performance.

`awslogs-create-group`

Required: No

Specify whether you want the log group automatically created. If this option isn't specified, it defaults to `false`.

**Warning**
This option isn't recommended. We recommend that you create the log group in advance using the CloudWatch Logs `CreateLogGroup` API action as each job tries to create the log group, increasing the chance that the job fails.

**Note**
The IAM policy for your execution role must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

---

**Specifying a log configuration in your job definition**

AWS Batch enables the `awslogs` log driver by default. This section describes how to customize the `awslogs` log configuration for a job. For more information, see Creating a job definition (p. 32).

The following log configuration JSON snippets have a `logConfiguration` object specified for each job; one for a WordPress job that sends logs to a log group called `awslogs-wordpress`, and one for a MySQL container that sends logs to a log group called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```json
"logConfiguration": {  
  "logDriver": "awslogs",  
  "options": {  
    "awslogs-group": "awslogs-wordpress",  
    "awslogs-stream-prefix": "awslogs-example"  
  } 
}
```

```json
"logConfiguration": {  
  "logDriver": "awslogs",  
  "options": {  
    "awslogs-group": "awslogs-mysql",  
    "awslogs-stream-prefix": "awslogs-example"  
  } 
}
```
In the AWS Batch console, the log configuration for the `wordpress` job definition is specified as shown in the following image.

After you have registered a task definition with the `awslogs` log driver in a job definition log configuration, you can submit a job with that job definition to start sending logs to CloudWatch Logs. For more information, see Submitting a Job (p. 14).

**Specifying sensitive data**

AWS Batch enables you to inject sensitive data into your jobs by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your job definition.

Secrets can be exposed to a job in the following ways:

- To inject sensitive data into your containers as environment variables, use the `secrets` job definition parameter.
• To reference sensitive information in the log configuration of a job, use the `secretOptions` job definition parameter.

**Topics**

- Specifying sensitive data using Secrets Manager (p. 65)
- Specifying sensitive data using Systems Manager Parameter Store (p. 70)

### Specifying sensitive data using Secrets Manager

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

When you inject a secret as an environment variable, you can specify a JSON key or version of a secret to inject. This process helps you control the sensitive data exposed to your job. 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 jobs.

- To inject a secret using a specific JSON key or version of a secret, the container instance in your compute environment must have version 1.37.0 or later of the Amazon ECS 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 in the Amazon Elastic Container Service Developer Guide.

  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 aren't supported.

- When using a job definition that references Secrets Manager secrets to retrieve sensitive data for your jobs, if you're also using interface VPC endpoints, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the AWS Secrets Manager User Guide.

- Sensitive data is injected into your job when the job is initially started. If the secret is subsequently updated or rotated, the job doesn't receive the updated value automatically. You must launch a new job to force the service to launch a fresh job with the updated secret value.

### Required IAM permissions for AWS Batch secrets

To use this feature, you must have the execution role and reference it in your job definition. This allows the container agent to pull the necessary Secrets Manager resources. For more information, see AWS Batch execution IAM role (p. 157).

**Important**

It's necessary that you use the Amazon ECS container agent configuration variable `ECS_ENABLE_AMSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `/etc/ecs/ecs.config` file during container instance creation or on an existing instance.
and then restart the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Configuration in the Amazon Elastic Container Service Developer Guide.

To provide access to the Secrets Manager secrets that you create, manually add the following permissions as an inline policy to the execution role. For more information, see Adding and Removing IAM Policies in the IAM User Guide.

- secretsmanager:GetSecretValue—Required if you're referencing a Secrets Manager secret.
- kms:Decrypt—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions.

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

### Injecting sensitive data as an environment variable

Within your job definition, you can specify the following:

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

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

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name
```

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

**json-key**

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

**version-stage**

Specifies the staging label of the version of a secret that you want to use. If a version staging label is specified, you can't specify a version ID. If no version stage is specified, the default behavior is to retrieve the secret with the AWSCURRENT staging label.
Staging labels are used to keep track of different versions of a secret when they 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 can't specify a version staging label. If no version ID is specified, the default behavior is to retrieve the secret with the `AWSCURRENT` staging label.

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

### Example container definitions

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

#### Example referencing a full secret

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

```json
{
    "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.

```json
{
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981dd39ab30c",
    "SecretString": "{"username1":"password1","username2":"password2"," username3":"password3"}",
    "VersionStages": ["AWSCURRENT"],
    "CreatedDate": 1581968848.921
}
```

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

```json
{
    "containerDefinitions": [{
      "secrets": [{
        "name": "environment_variable_name",
      }]
    }]
}
```
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.

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

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

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

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

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

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

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

```json
{
    "containerDefinitions": [{
        "secrets": [{
```

68
To specify a specific key and version ID, use the following syntax.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
    }]
  }
}
```

**Injecting sensitive data in a log configuration**

Within your job 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 job 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 CMK specifically for this secret. To create your own AWS KMS CMK, you must have permissions to create CMKs in your account.

6. Choose Next.

7. For Secret name, type an optional path and name, such as production/MyAwesomeAppSecret or development/TestSecret, and choose Next. You can optionally add a description to help you remember the purpose of this secret later.

The secret name must be ASCII letters, digits, or any of the following characters: /_+=.@-

8. (Optional) At this point, you can configure rotation for your secret. For this procedure, leave it at Disable automatic rotation and choose Next.

For information about how to configure rotation on new or existing secrets, see Rotating Your AWS Secrets Manager Secrets.

9. Review your settings, and then choose Store secret to save everything you entered as a new secret in Secrets Manager.

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. 70)
- Required IAM permissions for Amazon ECS secrets (p. 71)
- Injecting sensitive data as an environment variable (p. 71)
- Injecting sensitive data in a log configuration (p. 72)
- Creating an AWS Systems Manager Parameter Store parameter (p. 72)

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.

- 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 in the Amazon Elastic Container Service Developer Guide.

- Sensitive data is injected into the container for your job when the container is initially started. If the secret or Parameter Store parameter is subsequently updated or rotated, the container doesn’t receive
the updated value automatically. You must launch a new job to force the launch of a fresh job with updated secrets.

**Required IAM permissions for Amazon ECS secrets**

To use this feature, you must have the execution role and reference it in your job definition. This allows the Amazon ECS container agent to pull the necessary AWS Systems Manager resources. For more information, see [AWS Batch execution IAM role](#).

**Important**

You must use the Amazon ECS agent configuration variable

ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see [Amazon ECS Container Agent Configuration](#) in the Amazon Elastic Container Service Developer Guide.

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 execution role. For more information, see [Adding and Removing IAM Policies](#) in the IAM User Guide.

- `ssm:GetParameters`—Required if you're referencing a Systems Manager Parameter Store parameter in a task definition.
- `secretsmanager:GetSecretValue`—Required if you're referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions:

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

**Injecting sensitive data as an environment variable**

Within your container definition, specify secrets with the name of the environment variable to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter. If the Systems Manager Parameter Store parameter exists in the same
Region as the task you're 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.

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

### Injecting sensitive data in a log configuration

Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set in the container and the full ARN of 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're launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then the full ARN must be specified.

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

```json
{
  "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**

2. In the navigation pane, choose Parameter Store, Create parameter.
3. For Name, type a hierarchy and a parameter name. For example, type `test/database_password`.
4. For Description, type an optional description.
5. For Type, choose String, StringList, or SecureString.

**Note**

- If you choose SecureString, the KMS Key ID field appears. If you don't provide a KMS CMK ID, a KMS CMK ARN, an alias name, or an alias ARN, then the system uses alias/
aws/ssm, which is the default KMS CMK for Systems Manager. To avoid using this key, choose a custom key. For more information, see Use Secure String Parameters in the AWS Systems Manager User Guide.

- When you create a secure string parameter in the console by using the key-id parameter with either a custom KMS CMK alias name or an alias ARN, you must specify the prefix alias/ before the alias. The following is an ARN example:

  arn:aws:kms:us-east-2:123456789012:alias/MyAliasName

The following is an alias name example:

  alias/MyAliasName

6. For Value, type a value. For example, MyFirstParameter. If you chose SecureString, the value is masked as you type.

7. Choose Create parameter.

Example job definitions

The following example job definitions illustrate how to use common patterns such as environment variables, parameter substitution, and volume mounts.

Use environment variables

The following example job definition uses environment variables to specify a file type and Amazon S3 URL. This particular example is from the Creating a Simple "Fetch & Run" AWS Batch Job compute blog post. The fetch_and_run.sh script that's described in the blog post uses these environment variables to download the myjob.sh script from S3 and declare its file type.

Even though the command and environment variables are hardcoded into the job definition in this example, you can specify command and environment variable overrides to make the job definition more versatile.

```json
{
  "jobDefinitionName": "fetch_and_run",
  "type": "container",
  "containerProperties": {
    "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/fetch_and_run",
    "vcpus": 2,
    "memory": 2000,
    "command": [
      "myjob.sh",
      "60"
    ],
    "jobRoleArn": "arn:aws:iam::123456789012:role/AWSBatchS3ReadOnly",
    "environment": [
      { "name": "BATCH_FILE_S3_URL", "value": "s3://my-batch-scripts/myjob.sh" },
      { "name": "BATCH_FILE_TYPE", "value": "script" }
    ],
    "user": "nobody"
}
Using parameter substitution

The following example job definition illustrates how to allow for parameter substitution and to set default values.

The `Ref::` declarations in the command section are used to set placeholders for parameter substitution. When you submit a job with this job definition, you specify the parameter overrides to fill in those values, such as the `inputfile` and `outputfile`. The parameters section that follows sets a default for `codec`, but you can override that parameter as needed.

For more information, see Parameters (p. 44).

```
{
  "jobDefinitionName": "ffmpeg_parameters",
  "type": "container",
  "parameters": {"codec": "mp4"},
  "containerProperties": {
    "image": "my_repo/ffmpeg",
    "vcpus": 2,
    "memory": 2000,
    "command": [
      "ffmpeg",
      "-i",
      "Ref::inputfile",
      "-c",
      "Ref::codec",
      "-o",
      "Ref::outputfile"
    ],
    "jobRoleArn": "arn:aws:iam::123456789012:role/ECSTask-S3FullAccess",
    "user": "nobody"
  }
}
```

Test GPU functionality

The following example job definition tests if the GPU workload AMI described in Using a GPU workload AMI (p. 85) is configured properly. This example job definition runs the TensorFlow deep MNIST classifier example from GitHub.

```
{
  "containerProperties": {
    "image": "tensorflow/tensorflow:1.8.0-devel-gpu",
    "vcpus": 8,
    "command": [
      "sh",
      "-c",
      "cd /tensorflow/tensorflow/examples/tutorials/mnist; python mnist_deep.py"
    ],
    "memory": 32000
  },
  "type": "container",
  "jobDefinitionName": "tensorflow_mnist_deep"
}
```

You can create a file with the preceding JSON text called `tensorflow_mnist_deep.json` and then register an AWS Batch job definition with the following command:
aws batch register-job-definition --cli-input-json file://tensorflow_mnist_deep.json

Multi-node parallel job

The following example job definition illustrates a multi-node parallel job. For more information, see Building a tightly coupled molecular dynamics workflow with multi-node parallel jobs in AWS Batch in the AWS Compute blog.

```json
{
    "jobDefinitionName": "gromacs-jobdef",
    "revision": 6,
    "status": "ACTIVE",
    "type": "multinode",
    "parameters": {},
    "nodeProperties": {
        "numNodes": 2,
        "mainNode": 0,
        "nodeRangeProperties": [
            {
                "targetNodes": "0:1",
                "container": {
                    "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/gromacs_mpi:latest",
                    "vcpus": 8,
                    "memory": 24000,
                    "command": [],
                    "jobRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
                    "ulimits": [],
                    "instanceType": "p3.2xlarge"
                }
            }
        ]
    }
}
```
Job queues

Jobs are submitted to a job queue where they reside until they can be scheduled to run in a compute environment. An AWS account can have multiple job queues. For example, you can create a queue that uses Amazon EC2 On-Demand instances for high priority jobs and another queue that uses Amazon EC2 Spot Instances for low-priority jobs. Job queues have a priority that's used by the scheduler to determine which jobs in which queue should be evaluated for execution first.

Creating a job queue

Before you can submit jobs in AWS Batch, you must create a job queue. When you create a job queue, you associate one or more compute environments to the queue and assign an order of preference for the compute environments.

You also set a priority to the job queue that determines the order in which the AWS Batch scheduler places jobs onto its associated compute environments. For example, if a compute environment is associated with more than one job queue, the job queue with a higher priority is given preference for scheduling jobs to that compute environment.

To create a job queue

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Job queues, Create.
4. For Job queue name, enter a unique name for your job queue. Up to 128 letters (uppercase and lowercase), numbers, and underscores are allowed.
5. For Priority, enter an integer value for the job queue's priority. Job queues with a higher priority (or a higher integer value for the priority parameter) are evaluated first when associated with the same compute environment. Priority is determined in descending order, for example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1.
6. (Optional) Expand Additional configuration.
   - For State, select Enabled so that your job queue can accept job submissions.
7. (Optional) In the Tags section, you can specify the key and value for each tag to associate with the job queue. For more information, see Tagging your AWS Batch resources (p. 163).
8. In the Connected compute environments section, select one or more compute environments from the list to associate with the job queue, in the order that the queue should attempt placement. The job scheduler uses compute environment order to determine which compute environment should start a given job. Compute environments must be in the VALID state before you can associate them with a job queue. You can associate up to three compute environments with a job queue.

   Note
   All compute environments that are associated with a job queue must share the same provisioning model, either EC2 (On-Demand and Spot) or Fargate (Fargate and Fargate Spot). AWS Batch doesn't support mixing provisioning models in a single job queue.

   Note
   All compute environments that are associated with a job queue must share the same architecture. AWS Batch doesn't support mixing compute environment architecture types in a single job queue.

You can change the order of compute environments by choosing the up and down arrows next to the Order column in the table.
9. Choose **Create** to finish and create your job queue.

### Job queue template

An empty job queue template is shown below. You can use this template to create your job queue which can then be saved to a file and used with the AWS CLI `--cli-input-json` option. For more information about these parameters, see `CreateJobQueue` in the *AWS Batch API Reference*.

```json
{
    "jobQueueName": "",
    "state": "DISABLED",
    "priority": 0,
    "computeEnvironmentOrder": [
        {
            "order": 0,
            "computeEnvironment": ""
        }
    ],
    "tags": {
        "KeyName": ""
    }
}
```

**Note**

You can generate the preceding job queue template with the following AWS CLI command.

```
$ aws batch create-job-queue --generate-cli-skeleton
```

### Job queue parameters

Job queues are split into four basic components: the name, state, and priority of the job queue, and the compute environment order.

#### Job queue name

**jobQueueName**

The name for your job queue. Up to 128 letters (uppercase and lowercase), numbers, and underscores are allowed.

- **Type:** String
- **Required:** Yes

#### State

**state**

The state of the job queue. If the job queue state is **ENABLED** (the default value), it can accept jobs. If the job queue state is **DISABLED**, new jobs can't be added to the queue, but jobs already in the queue can finish.

- **Type:** String
Valid values: ENABLED | DISABLED

Required: No

**Priority**

**priority**

The priority of the job queue. Job queues with a higher priority (or a higher integer value for the priority parameter) are evaluated first when associated with same compute environment. Priority is determined in descending order, for example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1. All of the compute environments must be either EC2 (EC2 or SPOT) or Fargate (FARGATE or FARGATE_SPOT); EC2 and Fargate compute environments can’t be mixed.

Type: Integer

Required: Yes

**Compute environment order**

**computeEnvironmentOrder**

The set of compute environments mapped to a job queue and their order relative to each other. The job scheduler uses this parameter to determine which compute environment should run a specific job. Compute environments must be in the VALID state before you can associate them with a job queue. You can associate up to three compute environments with a job queue. All of the compute environments must be either EC2 (EC2 or SPOT) or Fargate (FARGATE or FARGATE_SPOT); EC2 and Fargate compute environments can’t be mixed.

**Note**

All compute environments that are associated with a job queue must share the same architecture. AWS Batch doesn't support mixing compute environment architecture types in a single job queue.

Type: Array of `ComputeEnvironmentOrder` objects

Required: Yes

**computeEnvironment**

The Amazon Resource Name (ARN) of the compute environment.

Type: String

Required: Yes

**order**

The order of the compute environment. Compute environments are tried in ascending order. For example, if two compute environments are associated with a job queue, the compute environment with a lower order integer value is tried for job placement first.

**Tags**

**tags**

Key-value pair tags to associate with the job queue. For more information, see Tagging your AWS Batch resources (p. 163).
Tags

Type: String to string map
Required: No
Job Scheduling

The AWS Batch scheduler evaluates when, where, and how to run jobs that have been submitted to a job queue. Jobs run in approximately the order in which they are submitted as long as all dependencies on other jobs have been met.
Job queues are mapped to one or more compute environments. Compute environments contain the Amazon ECS container instances that are used to run containerized batch jobs. A specific compute environment can also be mapped to one or many job queues. Within a job queue, the associated compute environments each have an order that's used by the scheduler to determine where jobs that are ready to be run should run. If the first compute environment has available resources, the job is scheduled to a container instance within that compute environment. If the compute environment can't provide a suitable compute resource, the scheduler attempts to run the job on the next compute environment.

Managed compute environments help you to meet your business requirements. In a managed compute environment, AWS Batch manages the capacity and instance types of the compute resources within the environment. This is based on the compute resource specification that you define when you create the compute environment. You can choose either to use EC2 On-Demand Instances and EC2 Spot Instances. Or, you can alternatively use Fargate and Fargate Spot capacity in your managed compute environment. You can optionally set a maximum price so that Spot Instances only launch when the Spot Instance price is under a specified percentage of the On-Demand price.

Managed compute environments launch Amazon ECS container instances into the VPC and subnets that you specify when you create the compute environment. Amazon ECS container instances need external network access to communicate with the Amazon ECS service endpoint. Some subnets don't provide container instances with public IP addresses. If your container instances don't have public IP addresses, they must use network address translation (NAT) to gain this access. For more information, see NAT gateways in the Amazon VPC User Guide. For more information about how to create a VPC, see Tutorial: Creating a VPC with Public and Private Subnets for Your Compute Environments (p. 147).

By default, AWS Batch managed compute environments use a recent, approved version of the Amazon ECS optimized AMI for compute resources. However, you might want to create your own AMI to use for your managed compute environments for various reasons. For more information, see Compute resource AMIs (p. 82).

Note

AWS Batch doesn't upgrade the AMIs in a compute environment after it's created. For example, it also doesn't update the AMIs in your compute environment when a newer version of the Amazon ECS optimized AMI is available. You're responsible for the management of the guest operating system. This includes any updates and security patches. You're also responsible for any additional application software or utilities that you install on the compute resources. To use a new AMI for your AWS Batch jobs:
1. Create a new compute environment with the new AMI.
2. Add the compute environment to an existing job queue.
3. Remove the earlier compute environment from your job queue.
4. Delete the earlier compute environment.

Unmanaged compute environments

In an unmanaged compute environment, you manage your own compute resources. You must verify that
the AMI you use for your compute resources meets the Amazon ECS container instance AMI specification.
For more information, see Compute resource AMI specification (p. 83) and Creating a compute
resource AMI (p. 83).

**Note**

AWS Fargate resources are not supported in unmanaged compute environments.

After you created your unmanaged compute environment, use the DescribeComputeEnvironments API
operation to view the compute environment details. Find the Amazon ECS cluster that's associated with
the environment and then manually launch your container instances into that Amazon ECS cluster.

The following AWS CLI command also provides the Amazon ECS cluster ARN:

```bash
aws batch describe-compute-environments --compute-environments unmanagedCE --query computeEnvironments[].ecsClusterArn
```

For more information, see Launching an Amazon ECS container instance in the Amazon Elastic Container
Service Developer Guide. When you launch your compute resources, specify the Amazon ECS cluster ARN
that the resources should register with the following Amazon EC2 user data. Replace `ecsClusterArn`
with the cluster ARN you obtained with the previous command.

```bash
#!/bin/bash
echo "ECS_CLUSTER=ecsClusterArn" >> /etc/ecs/ecs.config
```

Compute resource AMIs

By default, AWS Batch managed compute environments use a recent, approved version of the Amazon
ECS optimized AMI for compute resources. However, you might want to consider creating your own AMI
to use for your managed and unmanaged compute environments. You should do this if you also require
the following actions:

- Increase the storage size of your AMI root or data volumes
- Add instance storage volumes for supported Amazon EC2 instance types
- Configure the Amazon ECS container agent with custom options
- Configure Docker to use custom options
- Configure a GPU workload AMI that allows containers to access GPU hardware on supported Amazon
  EC2 instance types

**Topics**

- Compute resource AMI specification (p. 83)
- Creating a compute resource AMI (p. 83)
- Using a GPU workload AMI (p. 85)
Compute resource AMI specification

The basic AWS Batch compute resource AMI specification consists of the following items:

**Required**

- A modern Linux distribution that's running at least version 3.10 of the Linux kernel on an HVM virtualization type AMI.
  
  **Important**
  
  Multi-node parallel jobs can only run on compute resources that were launched on an Amazon Linux instance with the `ecs-init` package installed. We recommend that you use the default Amazon ECS optimized AMI when you create your compute environment. You can do this by not specifying a custom AMI. For more information, see Multi-node Parallel Jobs (p. 28).

- The Amazon ECS container agent. (We recommend that you use the latest version). For more information, see Installing the Amazon ECS Container Agent in the Amazon Elastic Container Service Developer Guide.

- The `awslogs` log driver must be specified as an available log driver with the `ECS_AVAILABLE_LOGGING_DRIVER` environment variable when the Amazon ECS container agent is started. For more information, see Amazon ECS Container Agent Configuration in the Amazon Elastic Container Service Developer Guide.

- A Docker daemon that's running at least version 1.9, and any Docker runtime dependencies. For more information, see Check runtime dependencies in the Docker documentation.
  
  **Note**
  
  For a better experience, we recommend the Docker version that ships with and is tested with the corresponding Amazon ECS agent version that you're using. For more information, see Amazon ECS Container Agent Versions in the Amazon Elastic Container Service Developer Guide.

**Recommended**

- An initialization and nanny process to run and monitor the Amazon ECS agent. The Amazon ECS-optimized AMI uses the `ecs-init` upstart process, and other operating systems might use systemd. To view several example user data configuration scripts that use systemctl to start and monitor the Amazon ECS container agent, see Example container instance User Data Configuration Scripts in the Amazon Elastic Container Service Developer Guide. For more information about `ecs-init`, see the `ecs-init` project on GitHub. At a minimum, managed compute environments require the Amazon ECS agent to start at boot. If the Amazon ECS agent isn't running on your compute resource, then it can't accept jobs from AWS Batch.

The Amazon ECS optimized AMI is preconfigured with these requirements and recommendations. We recommend that you use the Amazon ECS optimized AMI or an Amazon Linux AMI with the `ecs-init` package installed for your compute resources. You should choose another AMI if your application requires a specific operating system or a Docker version that's not yet available in those AMIs. For more information, see Amazon ECS-Optimized AMI in the Amazon Elastic Container Service Developer Guide.

Creating a compute resource AMI

You can create your own custom compute resource AMI to use for your managed and unmanaged compute environments, provided that you follow the Compute resource AMI specification (p. 83). After you have created your custom AMI, you can create a compute environment that uses that AMI, you can associate it with a job queue, and then start submitting jobs to that queue.
To create a custom compute resource AMI

1. Choose a base AMI to start from. The base AMI must use HVM virtualization, and it can't be a Windows AMI.

   **Note**
   The AMI that you choose for a compute environment must match the architecture of the instance types that you intend to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support ARM instances. Amazon ECS vends both x86 and ARM versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

   The Amazon ECS optimized AMI is the default AMI for compute resources in managed compute environments. The Amazon ECS optimized AMI is preconfigured and tested on AWS Batch by AWS engineers. It's the simplest AMI for you to get started and to get your compute resources running on AWS quickly. For more information, see Amazon ECS-Optimized AMI in the Amazon Elastic Container Service Developer Guide.

   Alternatively, you can choose another Amazon Linux variant and install the `ecs-init` package with the following command:

   ```bash
   sudo yum install -y ecs-init
   ```

   For example, if you want to run GPU workloads on your AWS Batch compute resources, you can start with the Amazon Linux Deep Learning AMI and configure it to be able to run AWS Batch jobs. For more information, see Using a GPU workload AMI (p. 85).

   **Important**
   If you choose a base AMI that doesn't support the `ecs-init` package, you must configure a way to start the Amazon ECS agent at boot and keep it running. To view several example user data configuration scripts that use `systemd` to start and monitor the Amazon ECS container agent, see Example container instance user data configuration scripts in the Amazon Elastic Container Service Developer Guide.

2. Launch an instance from your selected base AMI with the appropriate storage options for your AMI. You can configure the size and number of attached Amazon EBS volumes, or instance storage volumes if the instance type you’ve selected supports them. For more information, see Launching an Instance and Amazon EC2 Instance Store in the Amazon EC2 User Guide for Linux Instances.

3. Connect to your instance with SSH and perform any necessary configuration tasks. This might include any or all of the following steps:
   - Installing the Amazon ECS container agent. For more information, see Installing the Amazon ECS Container Agent in the Amazon Elastic Container Service Developer Guide.
   - Configuring a script to format instance store volumes.
   - Adding instance store volume or Amazon EFS file systems to the `/etc/fstab` file so that they're mounted at boot.
   - Configuring Docker options, such as enabling debugging or adjusting base image size.
   - Installing packages or copying files.

   For more information, see Connecting to Your Linux Instance Using SSH in the Amazon EC2 User Guide for Linux Instances.

4. If you started the Amazon ECS container agent on your instance, you must stop it and remove any persistent data checkpoint files before creating your AMI. Otherwise, if you don’t do this, the agent doesn’t start on instances that are launched from your AMI.
   a. Stop the Amazon ECS container agent.
• Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl stop ecs
```

• Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs
```

b. Remove the persistent data checkpoint files. By default, these files are located in the `/var/lib/ecs/data/` directory. Use the following command to remove any such files.

```
sudo rm -rf /var/lib/ecs/data/*
```

5. Create a new AMI from your running instance. For more information, see Creating an Amazon EBS-Backed Linux AMI in the Amazon EC2 User Guide for Linux Instances guide.

To use your new AMI with AWS Batch

1. After the AMI is created, create a compute environment with your new AMI. Make sure that you select Enable user-specified AMI ID and specify your custom AMI ID in Step 5.h.iii (p. 96)). For more information, see Creating a compute environment (p. 93).

   **Note**

   The AMI that you choose for a compute environment must match the architecture of the instance types that you intend to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support ARM instances. Amazon ECS vends both x86 and ARM versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

2. Create a job queue and associate your new compute environment. For more information, see Creating a job queue (p. 76).

   **Note**

   All compute environments that are associated with a job queue must share the same architecture. AWS Batch doesn’t support mixing compute environment architecture types in a single job queue.

3. (Optional) Submit a sample job to your new job queue. For more information, see Example job definitions (p. 73), Creating a job definition (p. 32), and Submitting a Job (p. 14).

Using a GPU workload AMI

To run GPU workloads on your AWS Batch compute resources, you must use an AMI with GPU support. For more information, see Working with GPUs on Amazon ECS and Amazon ECS-optimized AMIs in Amazon Elastic Container Service Developer Guide.

In managed compute environments, if the compute environment specifies any p2, p3, g3, g3s, or g4 instance types or instance families, then AWS Batch uses an Amazon ECS GPU optimized AMI.

In unmanaged compute environments, an Amazon ECS GPU-optimized AMI is recommended. You can use the AWS Command Line Interface or AWS Systems Manager Parameter Store GetParameter, GetParameters, and GetParametersByPath operations to retrieve the metadata for the recommended Amazon ECS GPU-optimized AMIs.

The following examples demonstrate the use of GetParameter.
AWS CLI

```bash
$ aws ssm get-parameter --name /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended \   --region us-east-2 --output json
```

The output includes the AMI information in the Value parameter:

```json
{
    "Parameter": {
        "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended",
        "LastModifiedDate": 1555434128.664,
        "Value": "{\"schema_version\":1,\"image_name\":\"amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs\",\"image_id\":\"ami-083c800fe4211192f\",\"os\":\"Amazon Linux 2\",\"ecs_runtime_version\":\"Docker version 18.06.1-ce\",\"ecs_agent_version\":\"1.27.0\"}",
        "Version": 9,
        "Type": "String",
        "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended"
    }
}
```

Python

```python
from __future__ import print_function
import json
import boto3

ssm = boto3.client('ssm', region_name='us-east-2')

response = ssm.get_parameter(Name='/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended')
jsonVal = json.loads(response['Parameter']['Value'])

print("image_id   = " + jsonVal['image_id'])
print("image_name = " + jsonVal['image_name'])
```

The output only includes the AMI ID and AMI name:

```python
image_id   = ami-083c800fe4211192f
image_name = amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs
```

The following examples demonstrate the use of GetParameters.

AWS CLI

```bash
$ aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name \   /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id \   --region us-east-2 --output json
```

The output includes the full metadata for each of the parameters:

```json
{
    "InvalidParameters": [],
    "Parameters": [
```
Using a GPU workload AMI

```python
from __future__ import print_function
import boto3
ssm = boto3.client('ssm', 'us-east-2')
response = ssm.get_parameters(
    Names=['/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name',
        '/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id'])
for parameter in response['Parameters']:
    print(parameter['Name'] + ' = ' + parameter['Value'])
```

The output includes the AMI ID and AMI name, using the full path for the names:

- `/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id` = `ami-083c800fe4211192f`
- `/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name` = `amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs`

The following examples demonstrate the use of `GetParametersByPath`.

**AWS CLI**

```
$ aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/
gpu/recommended \  
   --region us-east-2 --output json
```

The output includes the full metadata for all of the parameters under the specified path:

```json
{
   "Parameters": [
      {
         "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_agent_version",
```
Python

```python
from __future__ import print_function
import boto3
```
Launch template support

AWS Batch supports using Amazon EC2 launch templates with your EC2 compute environments. With launch templates, you can modify the default configuration of your AWS Batch compute resources without needing to create customized AMIs.

**Note**
Launch templates aren’t supported on AWS Fargate resources.

You must create a launch template before you can associate it with a compute environment. You can create a launch template in the Amazon EC2 console, or you can use the AWS CLI or an AWS SDK. For example, the following JSON file represents a launch template that resizes the Docker data volume for the default AWS Batch compute resource AMI and also sets it to be encrypted.

```json
{
    "LaunchTemplateName": "increase-container-volume-encrypt",
    "LaunchTemplateData": {
        "BlockDeviceMappings": [
            {
                "DeviceName": "/dev/xvdcz",
                "Ebs": {
                    "Encrypted": true,
                    "VolumeSize": 100,
                    "VolumeType": "gp2"
                }
            }
        ]
    }
}
```

You can create the previous launch template by saving the JSON to a file called `lt-data.json` and running the following AWS CLI command:

```
aws ec2 --region <region> create-launch-template --cli-input-json file://lt-data.json
```
For more information about launch templates, see Launching an Instance from a Launch Template in the Amazon EC2 User Guide for Linux Instances.

If you use a launch template to create your compute environment, you can move the following existing compute environment parameters to your launch template:

**Note**
If any of these parameters (with the exception of Amazon EC2 tags) are specified both in the launch template and in the compute environment configuration, the compute environment parameters take precedence. Amazon EC2 tags are merged between the launch template and the compute environment configuration. If there is a collision on the tag's key, then the value in the compute environment configuration takes precedence.

- Amazon EC2 key pair
- Amazon EC2 AMI ID
- Security group IDs
- Amazon EC2 tags

The following launch template parameters are **ignored** by AWS Batch:

- Instance type (specify your desired instance types when you create your compute environment)
- Instance role (specify your desired instance role when you create your compute environment)
- Network interface subnets (specify your desired subnets when you create your compute environment)
- Instance market options (AWS Batch must control Spot Instance configuration)
- Disable API termination (AWS Batch must control instance lifecycle)

AWS Batch doesn't support updating a compute environment with a new launch template version. If you update your launch template, you must create a new compute environment with the new template for the changes to take effect.

### Amazon EC2 user data in launch templates

You can supply Amazon EC2 user data in your launch template that's run by cloud-init when your instances launch. Your user data can perform common configuration scenarios, including but not limited to:

- Including users or groups
- Installing packages
- Creating partitions and file systems

Amazon EC2 user data in launch templates must be in the MIME multi-part archive format. This is because your user data is merged with other AWS Batch user data that's required to configure your compute resources. You can combine multiple user data blocks together into a single MIME multi-part file. For example, you might want to combine a cloud boothook that configures the Docker daemon with a user data shell script that writes configuration information for the Amazon ECS container agent.

If you're using AWS CloudFormation, the AWS::CloudFormation::Init type can be used with the cfn-init helper script to perform common configuration scenarios.

A MIME multi-part file consists of the following components:

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

The following are example MIME multi-part files that you can use to create your own.

Note
If you add user data to a launch template in the Amazon EC2 console, you can paste it in as
plaintext, or upload from a file. If you use the AWS CLI or an AWS SDK, you must first base64
encode the user data and submit that string as the value of the UserData parameter when you
call CreateLaunchTemplate, as shown in this JSON.

```
{
   "LaunchTemplateName": "base64-user-data",
   "LaunchTemplateData": {
      "UserData":
         "ewogICAgIkxhdW5jaFRlbXBsYXRlTmFtZSI6ICJpbmNyZWFzZS1jb250YWluZXItcm9sdW..."
   }
}
```

Examples
• Example: Mount an existing Amazon EFS file system (p. 91)
• Example: Override default Amazon ECS container agent configuration (p. 92)
• Example: Mount an existing Amazon FSx for Lustre file system (p. 92)

Example: Mount an existing Amazon EFS file system

Example
This example MIME multi-part file configures the compute resource to install the amazon-efs-utils
package and mount an existing Amazon EFS file system at /mnt/efs.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"
packages:
  - amazon-efs-utils
runcmd:
  - file_system_id_01=fs-abcdef123
  - efs_directory=/mnt/efs
  - mkdir -p ${efs_directory}
  - echo "${file_system_id_01}:/${efs_directory} efs tls,_netdev" >> /etc/fstab
  - mount -a -t efs defaults

--==MYBOUNDARY==--
```

91
Example: Override default Amazon ECS container agent configuration

Example

This example MIME multi-part file overrides the default Docker image cleanup settings for a compute resource.

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

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
echo ECS_IMAGE_CLEANUP_INTERVAL=60m >> /etc/ecs/ecs.config
echo ECS_IMAGE_MINIMUM_CLEANUP_AGE=60m >> /etc/ecs/ecs.config

--==MYBOUNDARY==--
```

Example: Mount an existing Amazon FSx for Lustre file system

Example

This example MIME multi-part file configures the compute resource to install the lustre2.10 package from the Extras Library and mount an existing Amazon FSx for Lustre file system at /scratch. This example is for Amazon Linux 2. For installation instructions for other Linux distributions, see Installing the Lustre Client in the Amazon FSx for Lustre User Guide.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"
runcmd:
- file_system_id_01=fs-0abcdef1234567890
- region=us-east-2
- fsx_directory=/scratch
- amazon-linux-extras install -y lustre2.10
- mkdir -p ${fsx_directory}
- mount -t lustre ${file_system_id_01}.fsx.${region}.amazonaws.com@tcp:fsx ${fsx_directory}

--==MYBOUNDARY==--
```

In the volumes and mountPoints members of the container properties the mount points must be mapped into the container.

```
{
  "volumes": [
    {
      "host": {
        "sourcePath": "/scratch"
      },
      "name": "Scratch"
    }
  ],
  "mountPoints": [
    {
      "containerPath": "/scratch",
    }
  ]
}
```
Creating a compute environment

Before you can run jobs in AWS Batch, you need to create a compute environment. You can create a managed compute environment where AWS Batch manages the Amazon EC2 instances or AWS Fargate resources within the environment based on your specifications. Or, alternatively, you can create an unmanaged compute environment where you handle the Amazon EC2 instance configuration within the environment.

Contents
- To create a managed compute environment using AWS Fargate resources (p. 93)
- To create a managed compute environment using EC2 resources (p. 94)
- To create an unmanaged compute environment using EC2 resources (p. 97)

To create a managed compute environment using AWS Fargate resources

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Compute environments, Create.
4. Configure the environment.
   a. For Compute environment type, choose Managed.
   b. For Compute environment name, specify a unique name for your compute environment. You can use up to 128 characters. Valid characters are letters (both uppercase and lowercase), numbers, hyphens (-), and underscores (_).
   c. Ensure that Enable compute environment is selected so that your compute environment can accept jobs from the AWS Batch job scheduler.
   d. (Optional) Expand Additional settings: service role, instance role, EC2 key pair.
      i. For Service role, choose to create a new role or use an existing role. The role allows the AWS Batch service to make calls to the required AWS API operations on your behalf. For more information, see AWS Batch Service IAM Role (p. 126). If you choose to create a new role, the required role (AWSBatchServiceRole) is created for you.
      ii. For Instance role, choose to create a new instance profile or use an existing instance profile that has the required IAM permissions attached. This instance profile allows the Amazon ECS container instances that are created for your compute environment to make calls to the required AWS API operations on your behalf. For more information, see Amazon ECS Instance Role (p. 129). If you choose to create a new instance profile, the required role (ecsInstanceRole) is created for you.
      iii. For EC2 key pair choose an existing Amazon EC2 key pair to associate with the instance at launch. This key pair allows you to connect to your instances with SSH (ensure that your security group allows incoming traffic on port 22).
5. Configure your Instance configuration.
   a. For Provisioning model, choose Fargate to launch Fargate On-Demand resources or Fargate Spot to use Fargate Spot resources.
To create a managed compute environment using EC2 resources

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Compute environments, Create.
4. Configure the environment.

   a. For Compute environment type, choose Managed.
   b. For Compute environment name, specify a unique name for your compute environment. You can use up to 128 characters. Valid characters are letters (both uppercase and lowercase), numbers, hyphens (-), and underscores (_).
   c. Ensure that Enable compute environment is selected so that your compute environment can accept jobs from the AWS Batch job scheduler.
   d. (Optional) Expand Additional settings: service role, instance role, EC2 key pair.
      i. For Service role, choose to create a new role or use an existing role. The role allows the AWS Batch service to make calls to the required AWS API operations on your behalf. For more information, see AWS Batch Service IAM Role (p. 126). If you choose to create a new role, the required role (AWSBatchServiceRole) is created for you.
      ii. For Instance role, choose to create a new instance profile or use an existing instance profile that has the required IAM permissions attached. This instance profile allows the Amazon ECS container instances that are created for your compute environment to make calls to
To create a managed compute environment using EC2 resources, AWS Batch will perform the required AWS API operations on your behalf. For more information, see Amazon ECS Instance Role (p. 129). If you choose to create a new instance profile, the required role (ecsInstanceRole) is created for you.

iii. For **EC2 key pair** choose an existing Amazon EC2 key pair to associate with the instance at launch. This key pair allows you to connect to your instances with SSH (verify that your security group allows incoming traffic on port 22).

5. **Configure your Instance configuration.**

a. For **Provisioning model**, choose **On-Demand** to launch Amazon EC2 On-Demand Instances or **Spot** to use Amazon EC2 Spot Instances.

b. If you chose to use Spot Instances:

   - (Optional) For **Maximum % on-demand price**, choose the maximum percentage that a Spot Instance price can be when compared with the On-Demand price for that instance type before instances are launched. For example, if your maximum price is 20%, then the Spot price must be less than 20% of the current On-Demand price for that EC2 instance. You always pay the lowest (market) price and never more than your maximum percentage. If you leave this field empty, the default value is 100% of the On-Demand price.

c. For **Minimum vCPUs**, choose the minimum number of EC2 vCPUs that your compute environment should maintain, regardless of job queue demand.

d. For **Maximum vCPUs**, choose the maximum number of EC2 vCPUs that your compute environment can scale out to, regardless of job queue demand.

e. For **Desired vCPUs**, choose the number of EC2 vCPUs that your compute environment should launch with. As your job queue demand increases, AWS Batch can increase the desired number of vCPUs in your compute environment and add EC2 instances, up to the maximum vCPUs. As demand decreases, AWS Batch can decrease the desired number of vCPUs in your compute environment and remove instances, down to the minimum vCPUs.

f. For **Allowed instance types**, choose the Amazon EC2 instance types that can be launched. You can specify instance families to launch any instance type within those families (for example, c5, c5n, or p3), or you can specify specific sizes within a family (such as c5.8xlarge). Note that metal instance types aren't in the instance families. For example, c5 doesn't include c5.metal. You can also choose **optimal** to select instance types (from the C4, M4, and R4 instance families) as you need that match the demand of your job queues.

   **Note**
   When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can't mix x86 and ARM instances in the same compute environment.

   **Note**
   AWS Batch will scale GPUs based on the required amount in your job queues. In order to use GPU scheduling, the compute environment must include instance types from the p2, p3, g3, g3s, or g4 families.

   **Note**
   Currently, optimal uses instance types from the C4, M4, and R4 instance families. In Regions that don't have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

g. For **Allocation strategy**, choose the allocation strategy to use when selecting instance types from the list of allowed instance types. **BEST_FIT_PROGRESSIVE** is usually the better choice for EC2 On-Demand compute environments, and **SPOT_CAPACITY_OPTIMIZED** for EC2 Spot compute environments. For more information, see the section called “Allocation strategies” (p. 106).

h. (Optional) Expand **Additional settings: launch template, user specified AMI**.
To create a managed compute environment using EC2 resources

i. (Optional) For **Launch template**, select an existing Amazon EC2 launch template to configure your compute resources; the default version of the template is automatically populated. For more information, see Launch template support (p. 89).

ii. (Optional) For **Launch template version**, enter $Default, $Latest, or a specific version number to use.

   **Important**
   After the compute environment is created, the launch template version used will not be changed, even if the $Default or $Latest version for the launch template is updated. To use a new launch template version, create a new compute environment, add the new compute environment to the existing job queue, remove the old compute environment from the job queue, and delete the old compute environment.

iii. (Optional) Check **Enable user-specified AMI ID** to use your own custom AMI. By default, AWS Batch managed compute environments use a recent, approved version of the Amazon ECS-optimized AMI for compute resources. You can create and use your own AMI in your compute environment by following the compute resource AMI specification. For more information, see Compute resource AMIs (p. 82).

   **Note**
   The AMI that you choose for a compute environment must match the architecture of the instance types that you intend to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support ARM instances. Amazon ECS vends both x86 and ARM versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

   - For **AMI ID**, paste your custom AMI ID and choose Validate AMI.

iv. (Optional) For **EC2 configuration** choose **Image type** and **Image ID override** values to provide information for AWS Batch to select Amazon Machine Images (AMIs) for instances in the compute environment. If the **Image ID override** isn't specified for each **Image type**, AWS Batch selects a recent Amazon ECS optimized AMI. If no **Image type** is specified, the default is a Amazon Linux for non-GPU, non-AWS Graviton instance. In the future, this default will change to Amazon Linux 2 for all non-GPU instances.

   **Amazon Linux 2**
   Default for all AWS Graviton-based instance families (for example, C6g, M6g, R6g, and T4g) and can be used for all non-GPU instance types.

   **Amazon Linux 2 (GPU)**
   Default for all GPU instance families (for example P4 and G4) and can be used for all non-AWS Graviton-based instance types.

   **Amazon Linux**
   Default for all non-GPU, non-AWS Graviton instance families. Amazon Linux is reaching the end-of-life of standard support. For more information, see Amazon Linux AMI.

i.

6. **Configure networking.**

   **Important**
   Compute resources need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your compute resources having public IP addresses. For more information about interface VPC endpoints, see Amazon ECS Interface VPC Endpoints (AWS PrivateLink) in the Amazon Elastic Container Service Developer Guide.
If you do not have an interface VPC endpoint configured and your compute resources 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. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Compute Environments (p. 147).

a. For **VPC ID**, choose a VPC into which to launch your instances.

b. For **Subnets**, choose which subnets in the selected VPC should host your instances. By default, all subnets within the selected VPC are chosen.

c. (Optional) Expand **Additional settings: Security groups, EC2 tags**.

i. For **Security groups**, choose a security group to attach to your instances. By default, the default security group for your VPC is chosen.

ii. (Optional) In the **EC2 tags**, you can tag the Amazon EC2 instances used by your On-Demand Instances. For example, you can specify "Name": "AWS Batch Instance - C4OnDemand" as a tag so that each instance in your compute environment has that name. This is helpful for recognizing your AWS Batch instances in the Amazon EC2 console.

    **Note**

    EC2 tags isn't available when using either Spot or Fargate provisioning models.

7. (Optional) In the **Tags** section, you can specify the key and value for each tag to associate with the compute environment. For more information, see Tagging your AWS Batch resources (p. 163).

8. Choose **Create compute environment** to finish.

---

**To create an unmanaged compute environment using EC2 resources**

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Compute environments, Create environment.
4. For **Compute environment type**, choose Unmanaged.
5. For **Compute environment name**, specify a unique name for your compute environment. You can use up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores.
6. For **Service role**, choose to create a new role or use an existing role that allows the AWS Batch service to make calls to the required AWS API operations on your behalf. For more information, see AWS Batch Service IAM Role (p. 126). If you choose to create a new role, the required role (AWSBatchServiceRole) is created for you.
7. Ensure that **Enable compute environment** is selected so that your compute environment can accept jobs from the AWS Batch job scheduler.
8. Choose **Create** to finish.
9. (Optional) Retrieve the Amazon ECS cluster ARN for the associated cluster. The following AWS CLI command provides the Amazon ECS cluster ARN for a compute environment:

   ```bash
   aws batch describe-compute-environments --compute-environments unmanagedCE --query computeEnvironments[].ecsClusterArn
   ```

10. (Optional) Launch container instances into the associated Amazon ECS cluster. For more information, see Launching an Amazon ECS container instance in the Amazon Elastic Container Service Developer Guide. When you launch your compute resources, specify the Amazon ECS cluster ARN that the resources should register with the following Amazon EC2 user data. Replace `ecsClusterArn` with the cluster ARN you obtained with the previous command.
Note
Your unmanaged compute environment does not have any compute resources until you launch them manually.

Compute environment template

The following example shows an empty compute environment template. You can use this template to create your compute environment that can then be saved to a file and used with the AWS CLI --cli-input-json option. For more information about these parameters, see CreateComputeEnvironment in the AWS Batch API Reference.

```json
{
    "computeEnvironmentName": "",
    "type": "UNMANAGED",
    "state": "ENABLED",
    "computeResources": {
        "type": "SPOT",
        "allocationStrategy": "SPOT_CAPACITY_OPTIMIZED",
        "minvCpus": 0,
        "maxvCpus": 0,
        "desiredvCpus": 0,
        "instanceTypes": [
            ""
        ],
        "imageId": "",
        "subnets": [
            ""
        ],
        "securityGroupIds": [
            ""
        ],
        "ec2KeyPair": "",
        "instanceRole": "",
        "tags": {
            "KeyName": ""
        },
        "placementGroup": "",
        "bidPercentage": 0,
        "spotIamFleetRole": "",
        "launchTemplate": {
            "launchTemplateId": "",
            "launchTemplateName": "",
            "version": ""
        },
        "ec2Configuration": [
            {
                "imageType": "",
                "imageIdOverride": ""
            }
        ],
        "serviceRole": "",
        "tags": {
            "KeyName": ""
        }
    }
}
```
Note
You can generate the preceding compute environment template with the following AWS CLI command.

```
$ aws batch create-compute-environment --generate-cli-skeleton
```

Compute environment parameters

Compute environments are split into five basic components: the name, type, and state of the compute environment, the compute resource definition (if it’s a managed compute environment), and the service role to use to provide IAM permissions to AWS Batch.

Topics
- Compute environment name (p. 99)
- Type (p. 99)
- State (p. 99)
- Compute resources (p. 100)
- Service role (p. 105)
- Tags (p. 106)

Compute environment name

```
computeEnvironmentName
```

The name for your compute environment. You can use up to 128 characters. Valid characters are letters (uppercase and lowercase), numbers, hyphens (-), and underscores (_).

Type: String
Required: Yes

Type

```
type
```

The type of the compute environment. Choose MANAGED to have AWS Batch manage the EC2 or Fargate compute resources that you define. For more information, see Compute resources (p. 100). Choose UNMANAGED to manage your own EC2 compute resources.

Type: String
Valid values: MANAGED | UNMANAGED
Required: Yes

State

```
state
```

The state of the compute environment.
If the state is ENABLED, the AWS Batch scheduler attempts to place jobs within the environment. These jobs are from an associated job queue on the compute resources. If the compute environment is managed, it can scale its instances out or in automatically based on job queue demand.

If the state is DISABLED, the AWS Batch scheduler doesn't attempt to place jobs within the environment. Jobs in a STARTING or RUNNING state continue to progress normally. Managed compute environments in the DISABLED state don't scale out. However, after instances go idle, they scale in to the smallest number of instances that satisfies the minvCpus value.

Type: String

Valid values: ENABLED | DISABLED

Required: No

**Compute resources**

*computeResources*

Details of the compute resources managed by the compute environment. For more information, see Compute Environments.

Type: ComputeResource object

Required: This parameter is required for managed compute environments
type

The type of compute environment. You can choose either to use EC2 On-Demand Instances (EC2) and EC2 Spot Instances (SPOT), or to use Fargate capacity (FARGATE) and Fargate Spot capacity (FARGATE_SPOT) in your managed compute environment. If you choose SPOT, you must also specify an Amazon EC2 Spot Fleet role with the spotIamFleetRole parameter. For more information, see Amazon EC2 Spot Fleet Role (p. 130).

Valid values: EC2 | SPOT | FARGATE | FARGATE_SPOT

Required: Yes

allocationStrategy

The allocation strategy to use for the compute resource if not enough instances of the best fitting EC2 instance type can be allocated. This might be due to availability of the instance type in the Region or Amazon EC2 service limits. For more information, see Allocation strategies (p. 106).

**Note**

This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified.

BEST_FIT (default)

AWS Batch selects an instance type that best fits the needs of the jobs with a preference for the lowest cost instance type. If additional instances of the selected instance type aren't available, AWS Batch waits for the additional instances to be available. If there aren't enough instances available, or if you're hitting Amazon EC2 service limits then additional jobs don't run until currently running jobs have completed. This allocation strategy keeps costs lower but can limit scaling. If you're using Spot Fleets with BEST_FIT then the Spot Fleet IAM Role must be specified.

BEST_FIT_PROGRESSIVE

Use additional instance types that are large enough to meet the requirements of the jobs in the queue, with a preference for instance types with a lower cost for each unit vCPU. If
additional instances of the previously selected instance types aren't available, AWS Batch selects new instance types.

**SPOT_CAPACITY_OPTIMIZED**

(Only available for Spot Instance compute resources) Use additional instance types that are large enough to meet the requirements of the jobs in the queue, with a preference for instance types that are less likely to be interrupted.

With both `BEST_FIT_PROGRESSIVE` and `SPOT_CAPACITY_OPTIMIZED` strategies, AWS Batch might need to exceed `maxvCpus` to meet your capacity requirements. In this event, AWS Batch never exceeds `maxvCpus` by more than a single instance.

Valid values: `BEST_FIT | BEST_FIT_PROGRESSIVE | SPOT_CAPACITY_OPTIMIZED`

Required: No

**minvCpus**

The minimum number of Amazon EC2 vCPUs that an environment should maintain (even if a compute environment is DISABLED).

**Note**

This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified.

Type: Integer

Required: Yes

**maxvCpus**

The maximum number of Amazon EC2 vCPUs that an environment can reach.

**Note**

With both `BEST_FIT_PROGRESSIVE` and `SPOT_CAPACITY_OPTIMIZED` allocation strategies, AWS Batch might need to exceed `maxvCpus` to meet your capacity requirements. In this event, AWS Batch never exceeds `maxvCpus` by more than a single instance. For example, AWS Batch uses no more than a single instance from among those specified in your compute environment.

Type: Integer

Required: Yes

**desiredvCpus**

The desired number of Amazon EC2 vCPUs in the compute environment. AWS Batch modifies this value between the minimum and maximum values based on job queue demand.

**Note**

This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified.

Type: Integer

Required: No

**instanceTypes**

The instance types that can be launched. This parameter isn't applicable to jobs that are running on Fargate resources, and shouldn't be specified. You can specify instance families to launch any instance type within those families (for example, `c5`, `c5n`, or `p3`). Or, you can specify specific sizes within a family (such as `c5.8xlarge`). Note that metal instance types aren't in the instance families (for example `c5` does not include `c5.metal`). You can also choose `optimal` to
select instance types (from the C4, M4, and R4 instance families) that match the demand of your job queues.

**Note**
When you create a compute environment, the instance types that you select for the compute environment must share the same architecture. For example, you can't mix x86 and ARM instances in the same compute environment.

**Note**
Currently, optimal uses instance types from the C4, M4, and R4 instance families. In Regions that don't have instance types from those instance families, instance types from the C5, M5, and R5 instance families are used.

Type: Array of strings

Required: yes

imageId

*This parameter is deprecated.*

The Amazon Machine Image (AMI) ID used for instances launched in the compute environment. This parameter is overridden by the `imageIdOverride` member of the `Ec2Configuration` structure.

**Note**
This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified.

**Note**
The AMI that you choose for a compute environment must match the architecture of the instance types that you intend to use for that compute environment. For example, if your compute environment uses A1 instance types, the compute resource AMI that you choose must support ARM instances. Amazon ECS vends both x86 and ARM versions of the Amazon ECS optimized Amazon Linux 2 AMI. For more information, see Amazon ECS optimized Amazon Linux 2 AMI in the Amazon Elastic Container Service Developer Guide.

Type: String

Required: No

subnets

The VPC subnets into which the compute resources are launched. These subnets must be within the same VPC. This parameter is required for jobs running on Fargate resources, where it can contain a maximum of 16 subnets. For more information, see VPCs and Subnets in the Amazon VPC User Guide.

Type: Array of strings

Required: Yes

securityGroupIds

The Amazon EC2 security groups associated with instances launched in the compute environment. One or more security groups must be specified, either in securityGroupIds or using a launch template referenced in launchTemplate. This parameter is required for jobs running on Fargate resources and must contain at least one security group. (Fargate doesn't support launch templates.) If security groups are specified using both securityGroupIds and launchTemplate, the values in securityGroupIds will be used.

Type: Array of strings
Required: Yes

ect2KeyPair

The EC2 key pair that's used for instances launched in the compute environment. You can use this key pair to log in to your instances with SSH.

Note
This parameter isn't applicable to jobs that are running on Fargate resources, and shouldn't be specified.

Type: String
Required: No

instanceRole

The Amazon ECS instance profile to attach to Amazon EC2 instances in a compute environment. This parameter isn't applicable to jobs that are running on Fargate resources, and shouldn't be specified. You can specify the short name or full Amazon Resource Name (ARN) of an instance profile. For example, ecsInstanceRole or arn:aws:iam::{aws_account_id}:instance-profile/ecsInstanceRole. For more information, see Amazon ECS Instance Role (p. 129).

Type: String
Required: No

tags

Key-value pair tags to be applied to EC2 instances that are launched in the compute environment. For example, you can specify "Name": "AWS Batch Instance - C4OnDemand" as a tag so that each instance in your compute environment has that name. This is helpful for recognizing your AWS Batch instances in the Amazon EC2 console. These tags can't be updated or removed after the compute environment has been created. Any changes require creating a new compute environment and removing the previous compute environment. These tags aren't seen when using the AWS Batch ListTagsForResource API operation.

Note
This parameter isn't applicable to jobs that are running on Fargate resources, and shouldn't be specified.

Type: String to string map
Required: No

placementGroup

The Amazon EC2 placement group to associate with your compute resources. This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified. If you intend to submit multi-node parallel jobs to your compute environment, you should consider creating a cluster placement group and associate it with your compute resources. This keeps your multi-node parallel job on a logical grouping of instances within a single Availability Zone with high network flow potential. For more information, see Placement Groups in the Amazon EC2 User Guide for Linux Instances.

Type: String
Required: No

bidPercentage

The maximum percentage that an EC2 Spot Instance price can be when compared with the On-Demand price for that instance type before instances are launched. For example, if your maximum percentage is 20%, then the Spot price must be less than 20% of the current On-Demand price for that EC2 instance. You always pay the lowest (market) price and never more
than your maximum percentage. If you leave this field empty, the default value is 100% of the On-Demand price.

**Note**
This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified.

Required: No

**spotIamFleetRole**

The Amazon Resource Name (ARN) of the Amazon EC2 Spot Fleet IAM role applied to a SPOT compute environment. This role is required if the allocation strategy set to BEST_FIT or if the allocation strategy isn't specified. For more information, see Amazon EC2 Spot Fleet Role (p. 130).

**Note**
This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified.

**Important**
To tag your Spot Instances on creation, the Spot Fleet IAM role specified here must use the newer AmazonEC2SpotFleetTaggingRole managed policy. The previously recommended AmazonEC2SpotFleetRole managed policy doesn't have the required permissions to tag Spot Instances. For more information, see Spot Instances Not Tagged on Creation (p. 170).

Type: String

Required: This parameter is required for SPOT compute environments.

**launchTemplate**

An optional launch template to associate with your compute resources. This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified. Any other compute resource parameters that you specify in a CreateComputeEnvironment API operation override the same parameters in the launch template. To use a launch template, you must specify either the launch template ID or launch template name in the request, but not both. For more information, see Launch template support (p. 89).

Type: LaunchTemplateSpecification

**launchTemplateName**

The name of the launch template.

Type: String

Required: No

**version**

The version number of the launch template, $Latest, or $Default.
If the value is $Latest, the latest version of the launch template is used. If the value is $Default, the default version of the launch template is used.

Default: $Default.

Type: String

Required: No

`ec2Configuration`

Provides information used to select Amazon Machine Images (AMIs) for instances in the EC2 compute environment. If `Ec2Configuration` isn't specified, the default is currently Amazon Linux (`ECS_AL1`) for non-GPU instances. In the future, this default will be changing to Amazon Linux 2 (`ECS_AL2`).

**Note**

This parameter isn't applicable to jobs that are running on Fargate resources, and shouldn't be specified.

Type: Array of `Ec2Configuration` objects

Required: No

`imageIdOverride`

The AMI ID used for instances launched in the compute environment that matches the image type. This setting overrides the `imageId` set in the `computeResource` object.

Type: String

Required: No

`imageType`

The image type to match with the instance type to select an AMI. If the `imageIdOverride` parameter isn't specified, then a recent Amazon ECS-optimized AMI is used.

- **Amazon Linux 2 (ECS_AL2)**
  
  Default for all AWS Graviton based instance families (for example, C6g, M6g, R6g, and T4g) and can be used for all non-GPU instance types.

- **Amazon Linux 2 (GPU) (ECS_AL2_NVIDIA)**
  
  Default for all GPU instance families (for example P4 and G4) and can be used for all non-AWS Graviton based instance types.

- **Amazon Linux (ECS_AL1)**
  
  Default for all non-GPU, non-AWS Graviton instance families. Amazon Linux is reaching the end-of-life of standard support. For more information, see Amazon Linux AMI.

Type: String

Required: Yes

**Service role**

`serviceRole`

The full Amazon Resource Name (ARN) of the IAM role that allows AWS Batch to make calls to other AWS services on your behalf. For more information, see AWS Batch Service IAM Role (p. 126).
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.

**Note**
Depending on how you created your AWS Batch service role, its ARN might contain the service-role path prefix. When you only specify the name of the service role, AWS Batch assumes that your ARN doesn't use the service-role path prefix. Because of this, we recommend that you specify the full ARN of your service role when you create compute environments.

Type: String
Required: Yes

**Tags**

Key-value pair tags to associate with the compute environment. For more information, see Tagging your AWS Batch resources (p. 163).

Type: String to string map
Required: No

### Allocation strategies

When a managed compute environment is created, AWS Batch selects instance types from the instanceTypes specified that best fit the needs of the jobs. The allocation strategy defines behavior when AWS Batch needs additional capacity. This parameter isn't applicable to jobs running on Fargate resources, and shouldn't be specified. For more information, see Allocation strategies (p. 106).

**BEST_FIT** (default)

AWS Batch selects an instance type that best fits the needs of the jobs with a preference for the lowest-cost instance type. If additional instances of the selected instance type are not available, AWS Batch waits for the additional instances to be available. If there aren't enough instances available, or if the user is hitting Amazon EC2 service limits then additional jobs don't run until currently running jobs have completed. This allocation strategy keeps costs lower but can limit scaling. If you're using Spot Fleets with BEST_FIT then the Spot Fleet IAM Role must be specified.

**BEST_FIT_PROGRESSIVE**

AWS Batch selects additional instance types that are large enough to meet the requirements of the jobs in the queue, with a preference for instance types with a lower cost for each unit vCPU. If additional instances of the previously selected instance types aren't available, AWS Batch selects new instance types.

**SPOT_CAPACITY_OPTIMIZED**

AWS Batch selects one or more instance types that are large enough to meet the requirements of the jobs in the queue, with a preference for instance types that are less likely to be interrupted. This allocation strategy is only available for Spot Instance compute resources.

With both BEST_FIT_PROGRESSIVE and SPOT_CAPACITY_OPTIMIZED strategies, AWS Batch might need to exceed maxvCpus to meet your capacity requirements. In this event, AWS Batch never exceeds maxvCpus by more than a single instance.
Compute Resource Memory Management

When the Amazon ECS container agent registers a compute resource into a compute environment, the agent must determine how much memory the compute resource has available to reserve for your jobs. Because of platform memory overhead and memory occupied by the system kernel, this number is different than the installed memory amount that is advertised for Amazon EC2 instances. For example, an m4.large instance has 8 GiB of installed memory. However, this does not always translate to exactly 8192 MiB of memory available for jobs when the compute resource registers.

If you specify 8192 MiB for the job, and none of your compute resources have 8192 MiB or greater of memory available to satisfy this requirement, then the job cannot be placed in your compute environment. If you are using a managed compute environment, then AWS Batch must launch a larger instance type to accommodate the request.

The default AWS Batch compute resource AMI also reserves 32 MiB of memory for the Amazon ECS container agent and other critical system processes. This memory is not available for job allocation. For more information, see Reserving System Memory (p. 107).

The Amazon ECS container agent uses the Docker `ReadMemInfo()` function to query the total memory available to the operating system. Linux provides 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.

```
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.

Reserving System Memory

If you occupy all of the memory on a compute resource with your jobs, then it is possible that your jobs 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 jobs. This effectively reserves that memory for critical system processes.

The default AWS Batch compute resource AMI reserves 32 MiB of memory for the Amazon ECS container agent and other critical system processes.

Viewing Compute Resource Memory

You can view how much memory a compute resource 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 jobs as much memory as possible for a particular instance type, you can observe the memory available for that compute resource and then assign your jobs that much memory.

To view compute resource memory

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster that hosts your compute resources to view. The cluster name for your compute environment begins with your compute environment name.
3. Choose **ECS Instances**, and select a compute resource from the **Container Instance** column to view.
4. The **Resources** section shows the registered and available memory for the compute resource.

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

The **Registered** memory value is what the compute resource registered with Amazon ECS when it was first launched, and the **Available** memory value is what has not already been allocated to jobs.
AWS Batch on AWS Fargate

AWS Fargate is a technology that you can use with AWS Batch 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 jobs with Fargate resources, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate job has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another job.

Contents

- Job definitions on Fargate (p. 109)
- Job queues on Fargate (p. 110)
- Compute environments on Fargate (p. 110)

Job definitions on Fargate

AWS Batch jobs on Fargate don't support all of the job definition parameters that are available. Some parameters are not supported at all, and others behave differently for Fargate jobs.

The following list describes job definition parameters that are not valid or otherwise restricted in Fargate jobs.

platformCapabilities

Must be specified as FARGATE.

```
"platformCapabilities": [ "FARGATE" ]
```

type

Must be specified as container.

```
"type": "container"
```

Parameters in containerProperties

executionRoleArn

Must be specified for jobs running on Fargate resources. For more information, see IAM Roles for Tasks in the Amazon Elastic Container Service Developer Guide.

```
"executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
```

fargatePlatformConfiguration

(Optional, only for Fargate job definitions). Specifies the Fargate platform version, or LATEST for a recent platform version. Possible values for platformVersion are 1.3.0, 1.4.0, and LATEST (default).
"fargatePlatformConfiguration": { "platformVersion": "1.4.0" }

instanceType, ulimits

Not applicable for jobs running on Fargate resources.

memory, vcpus

These settings must be specified in resourceRequirements

privileged

Either don't specify this parameter, or specify false.

"privileged": false

resourceRequirements

Both memory and vCPU requirements must be specified, using supported values (p. 55). GPU resources are not supported for jobs running on Fargate resources.

"resourceRequirements": [
  { "type": "MEMORY", "value": "512"},
  { "type": "VCPU", "value": "0.25"}
]

Parameters in linuxParameters
devices, maxSwap, sharedMemorySize, swappiness, tmpfs

Not applicable for jobs running on Fargate resources.

Parameters in logConfiguration

logDriver

Only awslogs and fluentd are supported. For more information, see Using the awslogs log driver (p. 61).

Members in networkConfiguration

assignPublicIp

If the private subnet does not have a NAT gateway attached to send traffic to the Internet, assignPublicIp must be "ENABLED". For more information, see For more information, see AWS Batch execution IAM role (p. 157).

Job queues on Fargate

AWS Batch job queues on Fargate are essentially unchanged. The only restriction is that the compute environments listed in computeEnvironmentOrder must all be Fargate compute environments (FARGATE or FARGATE_SPOT). EC2 and Fargate compute environments can't be mixed.

Compute environments on Fargate

AWS Batch compute environments on Fargate don't support all of the compute environment parameters that are available. Some parameters are not supported at all, and others have specific requirements for Fargate.
The following list describes compute environment parameters that are not valid or otherwise restricted in Fargate jobs.

**type**

This parameter must be **MANAGED**.

```
"type": "MANAGED"
```

**Parameters in the computeResources object**

- allocationStrategy
- bidPercentage
- desiredvCpus
- imageId
- instanceTypes
- ec2Configuration
- ec2KeyPair
- instanceRole
- launchTemplate
- minvCpus
- placementGroup
- spotIamFleetRole

These aren't applicable for Fargate compute environments and shouldn't be provided.

**subnets**

If the subnets listed in this parameter don't have NAT gateways attached, the `assignPublicIp` parameter in the job definition must be set to **ENABLED**.

**tags**

This isn't applicable for Fargate compute environments and shouldn't be provided. To specify tags for Fargate compute environments, use the `tags` parameter that's not in the `computeResources` object.

**type**

This must be either **FARGATE** or **FARGATE_SPOT**.

```
"type": "FARGATE_SPOT"
```
Elastic Fabric Adapter

An Elastic Fabric Adapter (EFA) is a network device to accelerate High Performance Computing (HPC) applications. AWS Batch supports applications that use EFA if the following conditions are met.

- Compute environment contains only supported instance types (c5n.18xlarge, c5n.metal, i3en.24xlarge, m5dn.24xlarge, m5n.24xlarge, r5dn.24xlarge, r5n.24xlarge, and p3dn.24xlarge).
- The OS in the AMI supports EFA: Amazon Linux, Amazon Linux 2, Red Hat Enterprise Linux 7.6, CentOS 7.6, Ubuntu 16.04, Ubuntu 18.04.
- The AMI has the EFA driver loaded.
- The security group for the EFA must allow all inbound and outbound traffic to and from the security group itself.
- All instances that use an EFA should be in the same cluster placement group.
- The job definition must include a devices member with hostPath set to /dev/infiniband/uverbs0 to allow the EFA device to be passed through to the container. If containerPath is specified it must also be set to /dev/infiniband/uverbs0. If permissions is set it must be set to READ | WRITE | MKNOD.

The location of the LinuxParameters member will be different for multi-node parallel jobs and single-node container jobs. The examples below demonstrate the differences but are missing required values.

**Example Example for multi-node parallel job**

```json
{
  "jobDefinitionName": "EFA-MNP-JobDef",
  "type": "multinode",
  "nodeProperties": {
    ...
    "nodeRangeProperties": [
      {
        ...
        "container": {
          ...
          "linuxParameters": {
            "devices": [
              {
                "hostPath": "/dev/infiniband/uverbs0",
                "containerPath": "/dev/infiniband/uverbs0",
                "permissions": [
                  "READ", "WRITE", "MKNOD"
                ]
              }
            ],
            ...
          }
        },
        ...
      }
    ],
  }
}
```

**Example Example for single-node container job**

```json
{
  "jobDefinitionName": "EFA-Container-JobDef",
  "type": "single-node",
  "containerProperties": {
    ...
    "linuxParameters": {
      "devices": [
        {
          "hostPath": "/dev/infiniband/uverbs0",
          "containerPath": "/dev/infiniband/uverbs0",
          "permissions": [
            "READ", "WRITE", "MKNOD"
          ]
        }
      ],
      ...
    }
  }
}
```
"type": "container",
...
"containerProperties": {
  ...
  "linuxParameters": {
    "devices": [
      {
        "hostPath": "/dev/infiniband/uverbs0",
      },
      ],
    },
  },
}

For more information about EFA, see Elastic Fabric Adapter in Amazon EC2 User Guide for Linux Instances.
AWS Batch IAM Policies, Roles, and Permissions

By default, IAM users don't have permission to create or modify AWS Batch resources, or perform tasks using the AWS Batch API. This means that they also can't do so using the AWS Batch console or the AWS CLI. To allow IAM users to create or modify resources and submit jobs, you must create IAM policies that grant IAM users permission to use the specific resources and API operations they need. Then, attach those policies to the IAM users or groups that require those permissions.

When you attach a policy to a user or group of users, it allows or denies the users permissions to perform the specified tasks on the specified resources. For more information, see Permissions and Policies in theIAM User Guide. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

Likewise, AWS Batch makes calls to other AWS services on your behalf, so the service must authenticate with your credentials. This authentication is accomplished by creating an IAM role and policy that can provide these permissions and then associating that role with your compute environments when you create them. For more information, see Amazon ECS Instance Role (p. 129), IAM Roles, Using Service-Linked Roles, and Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

Getting Started

An IAM policy must grant or deny permissions to use one or more AWS Batch actions.

Topics
- Policy Structure (p. 114)
- Supported Resource-Level Permissions for AWS Batch API Actions (p. 117)
- Example Policies (p. 122)
- AWS Batch managed policy (p. 125)
- Creating AWS Batch IAM Policies (p. 126)
- AWS Batch Service IAM Role (p. 126)
- Amazon ECS Instance Role (p. 129)
- Amazon EC2 Spot Fleet Role (p. 130)
- CloudWatch Events IAM Role (p. 131)

Policy Structure

The following topics explain the structure of an IAM policy.

Topics
- Policy Syntax (p. 115)
- Actions for AWS Batch (p. 115)
- Amazon Resource Names for AWS Batch (p. 116)
- Checking That Users Have the Required Permissions (p. 116)
Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```json
{
    "Statement": [{
        "Effect": "effect",
        "Action": "action",
        "Resource": "arn",
        "Condition": {
            "condition": {
                "key": "value"
            }
        }
    }]
}
```

There are various elements that make up a statement:

- **Effect**: The effect can be Allow or Deny. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.

- **Action**: The action is the specific API action for which you are granting or denying permission. To learn about specifying action, see Actions for AWS Batch (p. 115).

- **Resource**: The resource that's affected by the action. Some AWS Batch API actions allow you to include specific resources in your policy that can be created or modified by the action. To specify a resource in the statement, use its Amazon Resource Name (ARN). For more information, see Supported Resource-Level Permissions for AWS Batch API Actions (p. 117) and Amazon Resource Names for AWS Batch (p. 116). If the AWS Batch API operation currently does not support resource-level permissions, you must use the * wildcard to specify that all resources can be affected by the action.

- **Condition**: Conditions are optional. They can be used to control when your policy is in effect.

For more information about example IAM policy statements for AWS Batch, see Creating AWS Batch IAM Policies (p. 126).

Actions for AWS Batch

In an IAM policy statement, you can specify any API action from any service that supports IAM. For AWS Batch, use the following prefix with the name of the API action: `batch:`. For example: `batch:SubmitJob` and `batch:CreateComputeEnvironment`.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["batch:action1", "batch:action2"]
```

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Describe" as follows:

```
"Action": "batch:Describe*"
```

To specify all AWS Batch API actions, use the * wildcard as follows:

```
"Action": "batch:*"
```
For a list of AWS Batch actions, see Actions in the AWS Batch API Reference.

**Amazon Resource Names for AWS Batch**

Each IAM policy statement applies to the resources that you specify using their ARNs.

An ARN has the following general syntax:

```
arn:aws:[service]:[region]:[account]:resourceType/resourcePath
```

- **service**
  - The service (for example, batch).

- **region**
  - The Region for the resource (for example, us-east-2).

- **account**
  - The AWS account ID, with no hyphens (for example, 123456789012).

- **resourceType**
  - The type of resource (for example, compute-environment).

- **resourcePath**
  - A path that identifies the resource. You can use the * wildcard in your paths.

AWS Batch API operations currently supports resource-level permissions on several API operations. For more information, see Supported Resource-Level Permissions for AWS Batch API Actions (p. 117). To specify all resources, or if a specific API action does not support ARNs, use the * wildcard in the Resource element as follows:

```
"Resource": "*
```

**Checking That Users Have the Required Permissions**

Before you put an IAM policy into production, we recommend that you check whether it grants users the permissions to use the particular API actions and resources they need.

First, create an IAM user for testing purposes and attach the IAM policy to the test user. Then, make a request as the test user. You can make test requests in the console or with the AWS CLI.

**Note**

You can also test your policies with the IAM Policy Simulator. For more information about the policy simulator, see Working with the IAM Policy Simulator in the IAM User Guide.

If the policy doesn't grant the user the permissions that you expected, or is overly permissive, you can adjust the policy as needed. Retest until you get the desired results.

**Important**

It can take several minutes for policy changes to propagate before they take effect. Therefore, we recommend that you allow five minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the DecodeAuthorizationMessage action. For more information, see DecodeAuthorizationMessage in the AWS Security Token Service API Reference, and decode-authorization-message in the AWS CLI Command Reference.
Supported Resource-Level Permissions for AWS Batch API Actions

The term *resource-level permissions* refers to the ability to specify the resources on which users are allowed to perform actions. AWS Batch has partial support for resource-level permissions. For certain AWS Batch 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 permissions to submit jobs, but only to a specific job queue and only with a specific job definition.

The following list describes the AWS Batch API actions that currently support resource-level permissions, as well as the supported resources, resource ARNs, and condition keys for each action.

**Important**
If an AWS Batch API action isn't listed in this list, then it does not support resource-level permissions. If an AWS Batch API action does not support resource-level permissions, you can grant users permission to use the action, but you have to specify a * wildcard for the resource element of your policy statement.

**Actions**

- CancelJob (p. 117), CreateComputeEnvironment (p. 117), CreateJobQueue (p. 118), DeleteComputeEnvironment (p. 118), DeleteJobQueue (p. 118), DeregisterJobDefinition (p. 118), ListTagsForResource (p. 119), RegisterJobDefinition (p. 119), SubmitJob (p. 120), TagResource (p. 120), TerminateJob (p. 121), UntagResource (p. 121), UpdateComputeEnvironment (p. 122), UpdateJobQueue (p. 122)

**CancelJob**

Cancels a job in an AWS Batch queue.

**Resource**

- Job

  arn:aws:batch:::job/jobId

  **Condition keys**

  - aws:ResourceTag/${TagKey} (String)

  Filters actions based on the tags associated with the resource.

**CreateComputeEnvironment**

Creates an AWS Batch compute environment.

**Resource**

- Compute Environment

  arn:aws:batch:::compute-environment/compute-environment-name

  **Condition keys**

  - aws:ResourceTag/${TagKey} (String)

  Filters actions based on the tags associated with the resource.

  **Condition keys**

  - aws:RequestTag/${TagKey} (String)

  Filters actions based on the tags that are passed in the request.

  - aws:TagKeys (String)

  Filters actions based on the tag keys that are passed in the request.
CreateJobQueue

Creates an AWS Batch job queue.

Resource

Compute Environment


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

aws:RequestTag/${TagKey} (String)

Filters actions based on the tags that are passed in the request.

aws:TagKeys (String)

Filters actions based on the tag keys that are passed in the request.

DeleteComputeEnvironment

Deletes an AWS Batch compute environment.

Resource

Compute Environment


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

DeleteJobQueue

Deletes the specified job queue. Deleting the job queue eventually deletes all of the jobs in the queue. Jobs are deleted at a rate of about 16 jobs each second.

Resource

Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

DeregisterJobDefinition

Deregisters an AWS Batch job definition.

Resource

Job Definition

arn:aws:batch:region:account:job-definition/definition-name:revision
Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

ListTagsForResource

Lists the tags for the specified resource.

Resource

Compute Environment


Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

Job

arn:aws:batch:region:account:job/jobId

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

Job Definition

arn:aws:batch:region:account:job-definition/definition-name:revision

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

Job Queue

arn:aws:batch:region:account:job-queue/queue-name

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

RegisterJobDefinition

Registers an AWS Batch definition.

Resource

Job Definition

arn:aws:batch:region:account:job-definition/definition-name:revision

Condition keys

aws:ResourceTag/${TagKey} (String)

Filters actions based on the tags associated with the resource.

Condition keys

batch:AWSLogsCreateGroup (Boolean)

When this parameter is true, the awslogs-group will be created for the logs.

batch:AWSLogsGroup (String)

The awslogs group where the logs are located.
Supported Resource-Level Permissions

**batch:AWSLogsRegion (String)**

The Region where the logs are sent to.

**batch:AWSLogsStreamPrefix (String)**

The awslogs log stream prefix.

**batch:Image (String)**

The Docker image used to start a job.

**batch:LogDriver (String)**

The log driver used for the job.

**batch:Privileged (Boolean)**

When this parameter is true, the container for the job is given elevated permissions on the host container instance (similar to the root user).

**batch:User (String)**

The user name or numeric uid to use inside the container for the job.

**aws:RequestTag/${TagKey} (String)**

Filters actions based on the tags that are passed in the request.

**aws:TagKeys (String)**

Filters actions based on the tag keys that are passed in the request.

**SubmitJob**

Submits an AWS Batch job from a job definition.

**Job**

**Condition keys**

**aws:ResourceTag/${TagKey} (String)**

Filters actions based on the tags associated with the resource.

**Job Definition**

**Condition keys**

**aws:ResourceTag/${TagKey} (String)**

Filters actions based on the tags associated with the resource.

**Job Queue**

**Condition keys**

**aws:ResourceTag/${TagKey} (String)**

Filters actions based on the tags associated with the resource.

**TagResource**

Tags the specified resource.

**Resource**

**Compute Environment**

**Condition keys**

**aws:ResourceTag/${TagKey} (String)**

Filters actions based on the tags associated with the resource.
**Condition keys**

`aws:ResourceTag/${TagKey} (String)`

Filters actions based on the tags associated with the resource.

**Job**

`arn:aws:batch:region:account:job/jobId`

**Condition keys**

`aws:ResourceTag/${TagKey} (String)`

Filters actions based on the tags associated with the resource.

**Job Definition**

`arn:aws:batch:region:account:job-definition/definition-name:revision`

**Condition keys**

`aws:ResourceTag/${TagKey} (String)`

Filters actions based on the tags associated with the resource.

**Job Queue**

`arn:aws:batch:region:account:job-queue/queue-name`

**Condition keys**

`aws:ResourceTag/${TagKey} (String)`

Filters actions based on the tags associated with the resource.

Filters actions based on the tags that are passed in the request.

`aws:TagKeys (String)`

Filters actions based on the tag keys that are passed in the request.

**TerminateJob**

Terminates a job in an AWS Batch job queue.

**Resource**

**Job**

`arn:aws:batch:region:account:job/jobId`

**Condition keys**

`aws:ResourceTag/${TagKey} (String)`

Filters actions based on the tags associated with the resource.

**UntagResource**

Untags the specified resource.

**Resource**

**Compute Environment**


**Condition keys**

`aws:ResourceTag/${TagKey} (String)`

Filters actions based on the tags associated with the resource.
Example Policies

The following examples show policy statements that you could use to control the permissions that IAM users have to AWS Batch.

**Job**

```
arn:aws:batch:region:account:job/jobId
```

**Condition keys**

```
aws:ResourceTag/${TagKey} (String)
```

Filters actions based on the tags associated with the resource.

**Job Definition**

```
arn:aws:batch:region:account:job-definition/definition-name:revision
```

**Condition keys**

```
aws:ResourceTag/${TagKey} (String)
```

Filters actions based on the tags associated with the resource.

**Job Queue**

```
arn:aws:batch:region:account:job-queue/queue-name
```

**Condition keys**

```
aws:ResourceTag/${TagKey} (String)
```

Filters actions based on the tags associated with the resource.

**Condition keys**

```
aws:TagKeys (String)
```

Filters actions based on the tag keys that are passed in the request.

**UpdateComputeEnvironment**

Updates an AWS Batch compute environment.

**Resource**

**Compute Environment**

```
```

**Condition keys**

```
aws:ResourceTag/${TagKey} (String)
```

Filters actions based on the tags associated with the resource.

**UpdateJobQueue**

Updates a job queue.

**Resource**

**Job Queue**

```
arn:aws:batch:region:account:job-queue/queue-name
```

**Condition keys**

```
aws:ResourceTag/${TagKey} (String)
```

Filters actions based on the tags associated with the resource.
Examples

- Example: Read-Only Access (p. 123)
- Example: Restricting to POSIX User, Docker Image, Privilege Level, and Role on Job Submission (p. 123)
- Example: Restrict to Job Definition Prefix on Job Submission (p. 124)
- Example: Restrict to Job Queue (p. 125)

Example: Read-Only Access

The following policy grants users permissions to use all AWS Batch API actions whose names begin with `Describe` and `List`.

Users don't have permission to perform any actions on the resources (unless another statement grants them permission to do so) because they're denied permission to use API actions by default.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "batch:Describe*",
            "batch:List*"
         ],
         "Resource": "*"
      }
   ]
}
```

Example: Restricting to POSIX User, Docker Image, Privilege Level, and Role on Job Submission

The following policy allows a user to manage their own set of restricted job definitions.

The first and second statements allow a user to register and deregister any job definition name whose name is prefixed with `JobDefA_`.

The first statement also uses conditional context keys to restrict the POSIX user, privileged status, and container image values within the `containerProperties` of a job definition. For more information, see RegisterJobDefinition in the AWS Batch API Reference. In this example, job definitions can only be registered when the POSIX user is set to `nobody`, the privileged flag is set to `false`, and the image is set to `myImage` in an Amazon ECR repository.

```
Important
Docker resolves the user parameter to that user's uid from within the container image. In most cases this is found in the `/etc/passwd` file within the container image. This name resolution can be avoided by using direct uid values in both the job definition and any associated IAM policies. Both the AWS Batch APIs and the `batch:User` IAM conditional keys support numeric values.
```

The third statement restricts a user to passing only a specific role to a job definition.

```json
{
   "Version": "2012-10-17",
}
```
Example: Restrict to Job Definition Prefix on Job Submission

The following policy allows a user to submit jobs to any job queue with any job definition name that begins with `JobDefA_`.

**Important**
When scoping resource-level access for job submission, you must provide both job queue and job definition resource types.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:RegisterJobDefinition"
            ],
            "Resource": [
            ],
            "Condition": {
                "StringEquals": {
                    "batch:User": ["nobody"],
                    "batch:Image": ["<aws_account_id>.dkr.ecr.<aws_region>.amazonaws.com/myImage"]
            },
            "Bool": {
                "batch:Privileged": "false"
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "batch:DeregisterJobDefinition"
            ],
            "Resource": [
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::<aws_account_id>:role/MyBatchJobRole"
            ]
        }
    ]
}
```
"Resource": [  
],
]
]
]

Example: Restrict to Job Queue

The following policy allows a user to submit jobs to a specific job queue, named queue1, with any job definition name.

Important
When scoping resource-level access for job submission, you must provide both job queue and job definition resource types.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:SubmitJob"
      ],
      "Resource": [
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/*",
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/queue1"
      ]
    }
  ]
}
```

AWS Batch managed policy

AWS Batch provides a managed policy that you can attach to IAM users that provides permission to use AWS Batch resources and API operations. You can apply this policy directly, or you can use it as a starting point for creating your own policies. For more information about each API operation mentioned in these policies, see Actions in the AWS Batch API Reference.

AWSBatchFullAccess

This policy allows full administrator access to AWS Batch.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:*",
        "cloudwatch:GetMetricStatistics",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeVpcs",
        "ec2:DescribeImages",
```
Creating AWS Batch IAM Policies

You can create specific IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information, see Permissions and Policies in the IAM User Guide. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

AWS Batch Service IAM Role

AWS Batch makes calls to other AWS services on your behalf to manage the resources that you use with the service. Before you can use the service, you must have an IAM policy and role that provides the necessary permissions to AWS Batch.

In most cases, the AWS Batch service role is created for you automatically in the console first-run experience. You can use the following procedure to check if your account already has the AWS Batch service role.

The AWSBatchServiceRole policy is shown below.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/AWSBatchServiceRole",
                "arn:aws:iam::*:role/service-role/AWSBatchServiceRole",
                "arn:aws:iam::*:role/ecsInstanceRole",
                "arn:aws:iam::*:instance-profile/ecsInstanceRole",
                "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
                "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
                "arn:aws:iam::*:role/AWSBatchJobRole"
            ]
        }
    ]
}
```
```json
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceAttribute",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeImages",
        "ec2:DescribeImageAttribute",
        "ec2:DescribeSpotInstanceRequests",
        "ec2:DescribeSpotFleetInstances",
        "ec2:DescribeSpotFleetRequests",
        "ec2:DescribeSpotPriceHistory",
        "ec2:DescribeVpcClassicLink",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:CreateLaunchTemplate",
        "ec2:DeleteLaunchTemplate",
        "ec2:RequestSpotFleet",
        "ec2:CancelSpotFleetRequests",
        "ec2:ModifySpotFleetRequest",
        "ec2:TerminateInstances",
        "ec2:RunInstances",
        "autoscaling:DescribeAccountLimits",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:CreateLaunchConfiguration",
        "autoscaling:CreateAutoScalingGroup",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:SetDesiredCapacity",
        "autoscaling:DeleteLaunchConfiguration",
        "autoscaling:DeleteAutoScalingGroup",
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:SuspendProcesses",
        "autoscaling:PutNotificationConfiguration",
        "autoscaling:TerminateInstanceInAutoScalingGroup",
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListTaskDefinitionFamilies",
        "ecs:ListTaskDefinitions",
        "ecs:ListTasks",
        "ecs:CreateCluster",
        "ecs:DeleteCluster",
        "ecs:RegisterTaskDefinition",
        "ecs:DeregisterTaskDefinition",
        "ecs:RunTask",
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:UpdateContainerAgent",
        "ecs:DeregisterContainerInstance",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "iam:GetInstanceProfile",
        "iam:GetRole"
    ],
    "Resource": "*
}
}```
You can use the following procedure to see if your account already has the AWS Batch service role and attach the managed IAM policy if needed.

**To check for the AWSBatchServiceRole 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 AWSBatchServiceRole. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.
4. Choose Permissions.
5. Ensure that the AWSBatchServiceRole managed policy is attached to the role. If the policy is attached, your AWS Batch service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose Attach Policy.
To narrow the list of available policies to attach, for Filter, type `AWSBatchServiceRole`.

Select the `AWSBatchServiceRole` policy and choose Attach Policy.

Choose Trust Relationships, Edit Trust Relationship.

Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose Cancel. If the trust relationship does not match, copy the policy into the Policy Document window and choose Update Trust Policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "batch.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }]
}
```

To create the `AWSBatchServiceRole` IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create New Role.
3. For Select type of trusted entity, choose AWS service. For Choose the service that will use this role, choose Batch.
5. For Role Name, type `AWSBatchServiceRole` and choose Create Role.

Amazon ECS Instance Role

AWS Batch compute environments are populated with Amazon ECS container instances, and they run the Amazon ECS container agent locally. The Amazon ECS container agent makes calls to various AWS APIs on your behalf, so container instances that run the agent require an IAM policy and role for these services to know that the agent belongs to you. Before you can create a compute environment and launch container instances into it, you must create an IAM role and an instance profile for those container instances to use when they are launched. This requirement applies to container instances launched with or without the Amazon ECS-optimized AMI provided by Amazon.

The Amazon ECS instance role and instance profile are automatically created for you in the console first-run experience. However, you can use the following procedure to check and see if your account already has the Amazon ECS instance role and instance profile 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 steps below to create the role.
   a. Choose Create Role.
   b. For Select type of trusted entity, choose AWS service. For Choose the service that will use this role, choose Elastic Container Service. For Select your use case, choose EC2 Role for Elastic Container Service.
   c. Choose Next: Permissions, Next: Tags, and Next: Review.
   d. For Role Name, type `ecsInstanceRole` and choose Create Role.
Amazon EC2 Spot Fleet Role

If you create a managed compute environment that uses Amazon EC2 Spot Fleet Instances, you must create a role that grants the Spot Fleet permission to launch, tag, and terminate instances on your behalf. Specify the role in your Spot Fleet request. You must also have the \texttt{AWSServiceRoleForEC2Spot} and \texttt{AWSServiceRoleForEC2SpotFleet} service-linked roles for Amazon EC2 Spot and Spot Fleet. Use the procedures below to create all of these roles. For more information, see Using Service-Linked Roles and Creating a Role to Delegate Permissions to an AWS Service in the \textit{IAM User Guide}.

Topics

- Create Amazon EC2 Spot Fleet Roles in the AWS Management Console (p. 130)
- Create Amazon EC2 Spot Fleet Roles with the AWS CLI (p. 131)

Create Amazon EC2 Spot Fleet Roles in the AWS Management Console

To create the \texttt{AWSServiceRoleForEC2Spot} IAM service-linked role for Amazon EC2 Spot

1. Open the IAM console at \texttt{https://console.aws.amazon.com/iam/}.
2. In the navigation pane, choose \texttt{Roles}, \texttt{Create role}.
3. For \texttt{Select type of trusted entity}, choose \texttt{AWS service}. For \texttt{Choose the service that will use this role}, choose \texttt{EC2}.
4. In the \texttt{Select your use case} section, choose \texttt{EC2 - Spot Instances}.
5. Choose \texttt{Next: Permissions}, \texttt{Next: Tags}, and \texttt{Next: Review}.
6. For \texttt{Role Name}, type \texttt{AmazonEC2SpotFleetRole}. Choose \texttt{Create Role}.

To create the \texttt{AWSServiceRoleForEC2SpotFleet} IAM service-linked role for Amazon EC2 Spot Fleet

1. Open the IAM console at \texttt{https://console.aws.amazon.com/iam/}.
2. In the navigation pane, choose \texttt{Roles}, \texttt{Create role}.
3. For \texttt{Select type of trusted entity}, choose \texttt{AWS service}. For \texttt{Choose the service that will use this role}, choose \texttt{EC2}.
4. In the \texttt{Select your use case} section, choose \texttt{EC2 - Spot Fleet}.
5. Choose \texttt{Next: Permissions}, \texttt{Next: Tags}, and \texttt{Next: Review}.
6. For \texttt{Role Name}, type \texttt{AWSServiceRoleForEC2SpotFleet}. Choose \texttt{Create role}.

Note

Historically, there have been two managed policies for the Amazon EC2 Spot Fleet role.

- \texttt{AmazonEC2SpotFleetRole}: This was the original managed policy for the Spot Fleet role. It has tighter IAM permissions, but it does not support Spot Instance tagging in compute environments. If you've previously created a Spot Fleet role with this policy, see Spot Instances Not Tagged on Creation (p. 170) to apply the new recommended policy to that role.

- \texttt{AmazonEC2SpotFleetTaggingRole}: This role provides all of the necessary permissions to tag Amazon EC2 Spot Instances. Use this role to allow Spot Instance tagging on your AWS Batch compute environments.
Create Amazon EC2 Spot Fleet Roles with the AWS CLI

To create the AmazonEC2SpotFleetRole IAM role for your Spot Fleet compute environments

1. Run the following command with the AWS CLI:

```bash
aws iam create-role --role-name AmazonEC2SpotFleetRole \
  --assume-role-policy-document '{"Version":"2012-10-17","Statement":
  [{"Sid":"","Effect":"Allow","Principal":
    {"Service":"spotfleet.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
```

2. To attach the AmazonEC2SpotFleetTaggingRole managed IAM policy to your AmazonEC2SpotFleetRole role, run the following command with the AWS CLI:

```bash
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole \
  --role-name AmazonEC2SpotFleetRole
```

To create the AWSServiceRoleForEC2Spot IAM service-linked role for Amazon EC2 Spot

- Run the following command with the AWS CLI:

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

To create the AWSServiceRoleForEC2SpotFleet IAM service-linked role for Amazon EC2 Spot Fleet

- Run the following command with the AWS CLI:

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

CloudWatch Events IAM Role

Amazon CloudWatch Events delivers a near-real time stream of system events that describe changes in Amazon Web Services resources. AWS Batch jobs are available as CloudWatch Events targets. Using simple rules that you can quickly set up, you can match events and submit AWS Batch jobs in response to them. Before you can submit AWS Batch jobs with CloudWatch Events rules and targets, CloudWatch Events must have permissions to run AWS Batch jobs on your behalf.

**Note**

When you create a rule in the CloudWatch Events console that specifies an AWS Batch queue as a target, you are provided with an opportunity to create this role. For an example walkthrough, see AWS Batch Jobs as CloudWatch Events Targets (p. 134).

The trust relationship for your CloudWatch Events IAM role must provide the events.amazonaws.com service principal the ability to assume the role, as shown below.

```json
{
  "Version": "2012-10-17",
  "Statement":
```
The policy attached to your CloudWatch Events IAM role should allow `batch:SubmitJob` permissions on your resources. AWS Batch provides the `AWSBatchServiceEventTargetRole` managed policy to provide these permissions, which are shown below.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob"
            ],
            "Resource": "*"
        }
    ]
}
```
AWS Batch Event Stream for CloudWatch Events

You can use the AWS Batch event stream for CloudWatch Events to receive near real-time notifications regarding the current state of jobs that have been submitted to your job queues.

Using CloudWatch Events, you can monitor the progress of jobs, build AWS Batch custom workflows with complex dependencies, generate usage reports or metrics around job execution, or build your own custom dashboards. With AWS Batch and CloudWatch Events, you can eliminate scheduling and monitoring code that continuously polls AWS Batch for job status changes. Instead, handle AWS Batch job state changes asynchronously using any CloudWatch Events target, such as AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, or Amazon Kinesis Data Streams.

Events from the AWS Batch event stream are ensured to be delivered at least one time. In the event that duplicate events are sent, the event provides enough information to identify duplicates (you can compare the time stamp of the event and the job status).

AWS Batch jobs are available as CloudWatch Events targets. Using simple rules that you can quickly set up, you can match events and submit AWS Batch jobs in response to them. For more information, see What is Amazon CloudWatch Events? in the Amazon CloudWatch Events User Guide. You can also use CloudWatch Events to schedule automated actions that self-trigger at certain times using cron or rate expressions. For more information, see Schedule Expressions for Rules in the Amazon CloudWatch Events User Guide. For an example walkthrough, see AWS Batch Jobs as CloudWatch Events Targets (p. 134).

Topics
- AWS Batch Events (p. 133)
- AWS Batch Jobs as CloudWatch Events Targets (p. 134)
- Tutorial: Listening for AWS Batch CloudWatch Events (p. 137)
- Tutorial: Sending Amazon Simple Notification Service Alerts for Failed Job Events (p. 139)

AWS Batch Events

AWS Batch sends job status change events to CloudWatch Events. AWS Batch tracks the state of your jobs. If a previously submitted job's status changes, an event is triggered, for example, if a job in the RUNNING status moves to the FAILED status. These events are classified as job state change events.

Note
AWS Batch may add other event types, sources, and details in the future. If you are programmatical 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.

Job State Change Events

Any time that an existing (previously submitted) job changes states, an event is created. For more information about AWS Batch job states, see Job States (p. 16).

Note
Events are not created for the initial job submission.
Example Job State Change Event

Job state change events are delivered in the following format (the detail section below resembles the JobDetail object that is returned from a DescribeJobs API operation in the AWS Batch API Reference). For more information about CloudWatch Events parameters, see Events and Event Patterns in the Amazon CloudWatch Events User Guide.

```json
{
  "version": "0",
  "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
  "detail-type": "Batch Job State Change",
  "source": "aws.batch",
  "account": "aws_account_id",
  "time": "2017-10-23T17:56:03Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:batch:us-east-1:aws_account_id:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8"
  ],
  "detail": {
    "jobName": "event-test",
    "jobId": "4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
    "status": "RUNNABLE",
    "attempts": [],
    "createdAt": 1508781340401,
    "retryStrategy": {
      "attempts": 1
    },
    "dependsOn": [],
    "parameters": {},
    "container": {
      "image": "busybox",
      "vcpus": 2,
      "memory": 2000,
      "command": [
        "echo",
        "'hello world'"
      ],
      "volumes": [],
      "environment": [],
      "mountPoints": [],
      "ulimits": []
    }
  }
}
```

AWS Batch Jobs as CloudWatch Events Targets

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services resources. AWS Batch jobs are available as CloudWatch Events targets. Using simple rules that you can quickly set up, you can match events and submit AWS Batch jobs in response to them. For more information, see What is Amazon CloudWatch Events? in the Amazon CloudWatch Events User Guide.

You can also use CloudWatch Events to schedule automated actions that self-trigger at certain times using cron or rate expressions. For more information, see Schedule Expressions for Rules in the Amazon CloudWatch Events User Guide.
Some common use cases for AWS Batch jobs as a CloudWatch Events target are:

- Create a scheduled job that occurs at regular time intervals, such as a cron job that happens in low-usage hours when Amazon EC2 Spot Instances are less expensive.
- Run an AWS Batch job in response to an API operation logged in CloudTrail, such as automatically submitting a job when an object is uploaded to a specified Amazon S3 bucket, and passing the bucket and key name of the object to AWS Batch parameters using the CloudWatch Events input transformer.

**Note**
In this scenario, all of the AWS resources (Amazon S3 bucket, CloudWatch Events rule, CloudTrail logs, etc.) must be in the same region.

Before you can submit AWS Batch jobs with CloudWatch Events rules and targets, the CloudWatch Events service needs permissions to run AWS Batch jobs on your behalf. When you create a rule in the CloudWatch Events console that specifies an AWS Batch job as a target, you are provided with an opportunity to create this role. For more information about the required service principal and IAM permissions for this role, see CloudWatch Events IAM Role (p. 131).

### Creating a Scheduled AWS Batch Job

The procedure below shows how to create a scheduled AWS Batch job and the required CloudWatch Events IAM role.

#### To create a scheduled AWS Batch job with CloudWatch Events

2. In the left navigation, choose **Events**, **Create rule**.
3. For **Event source**, choose **Schedule**, and then choose whether to use a fixed interval schedule or a cron expression for your schedule rule. For more information, see Schedule Expressions for Rules in the Amazon CloudWatch Events User Guide.
   - For **Fixed rate of**, enter the interval and unit for your schedule.
   - For **Cron expression**, enter the cron expression for your task schedule. These expressions have six required fields, and fields are separated by white space. For more information, and examples of cron expressions, see Cron Expressions in the Amazon CloudWatch Events User Guide.
4. For **Targets**, choose **Add target**.
5. Choose **Batch job queue** and fill in the following fields appropriately:
   - **Job queue**: Enter the Amazon Resource Name (ARN) of the job queue in which to schedule your job.
   - **Job definition**: Enter the name and revision or full ARN of the job definition to use for your job.
   - **Job name**: Enter a name for your job.
   - **Array size**: (Optional) Enter an array size for your job to run more than one copy. For more information, see Array Jobs (p. 20).
   - **Job attempts**: (Optional) Enter the number of times to retry your job if it fails. For more information, see Automated Job Retries (p. 18).
6. Choose an existing CloudWatch Events IAM role to use for your job, or Create a new role for this specific resource to create a new one. For more information, see CloudWatch Events IAM Role (p. 131).
7. For **Rule definition**, fill in the following fields appropriately, and then choose **Create rule**.
   - **Name**: Enter a name for your rule.
   - **Description**: (Optional) Enter a description for your rule.
• **State:** Choose whether to enable your rule so that it begins scheduling at the next interval, or disable it until a later date.

### Passing Event Information to an AWS Batch Target using the CloudWatch Events Input Transformer

You can use the CloudWatch Events input transformer to pass event information to AWS Batch in a job submission. This can be especially valuable if you are triggering jobs as a result of other AWS event information, such as uploading an object to an Amazon S3 bucket. You could use a job definition with parameter substitution values in the container's command, and the CloudWatch Events input transformer can provide the parameter values based on the event data. For example, the job definition below expects to see parameter values called `S3bucket` and `S3key`.

```json
{
    "jobDefinitionName": "echo-parameters",
    "containerProperties": {
        "image": "busybox",
        "vcpus": 2,
        "memory": 2000,
        "command": [
            "echo",
            "Ref::S3bucket",
            "Ref::S3key"
        ]
    }
}
```

Then, you simply create an AWS Batch event target that parses information from the event that triggers it and transforms it into a `parameters` object. When the job runs, the parameters from the trigger event are passed to the job container's command.

**Note**

In this scenario, all of the AWS resources (Amazon S3 bucket, CloudWatch Events rule, CloudTrail logs, etc.) must be in the same region.

**To create an AWS Batch target that uses the input transformer**

2. In the left navigation, choose **Events**, **Create rule**.
3. For **Event source**, choose **Event Pattern**, and then construct the rule as desired to match your application needs.
4. For **Targets**, choose **Batch job queue** and then specify the job queue, job definition, and job name to use for the jobs that are triggered by this rule.
5. Choose **Configure input** and then choose **Input Transformer**.
6. For the upper input transformer text box, specify the values to parse from the triggering event. For example, to parse the bucket and key name from an Amazon S3 event, use the following JSON.

   ```json
   {"S3BucketValue": ".detail.requestParameters.bucketName",
   "S3KeyValue": ".detail.requestParameters.key"
   }
   ```

7. For the lower input transformer text box, create the `Parameters` structure to pass to the AWS Batch job. These parameters are substituted for the `Ref::S3bucket` and `Ref::S3key` placeholders in the job container's command when the job runs.

   ```json
   {"Parameters": {"S3bucket": <S3BucketValue>, "S3key": <S3KeyValue>}}
   ```
8. Choose an existing CloudWatch Events IAM role to use for your job, or **Create a new role for this specific resource** to create a new one. For more information, see CloudWatch Events IAM Role (p. 131).

9. Choose **Configure details** and then for **Rule definition**, fill in the following fields appropriately, and then choose **Create rule**.
   - **Name**: Enter a name for your rule.
   - **Description**: (Optional) Enter a description for your rule.
   - **State**: Choose whether to enable your rule now, or disable it until a later date.

---

**Tutorial: Listening for AWS Batch CloudWatch Events**

In this tutorial, you set up a simple AWS Lambda function that listens for AWS Batch job events and writes them out to a CloudWatch Logs log stream.

**Prerequisites**

This tutorial assumes that you have a working compute environment and job queue that are ready to accept jobs. If you do not have a running compute environment and job queue to capture events from, follow the steps in Getting Started with AWS Batch (p. 9) to create one. At the end of this tutorial, you can submit a job to this job queue 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 AWS Batch event stream messages.

**To create a target Lambda function**

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create a Lambda function**, **Author from scratch**.
3. For **Function name**, enter **batch-event-stream-handler**.
4. For **Runtime**, choose **Python 3.8**.
5. Choose **Create function**.
6. In the **Function code** section, edit the sample code to match the following example:

   ```python
   import json

   def lambda_handler(event, _context):
       # _context is not used
       del _context
       if event['source'] != "aws.batch":
           raise ValueError("Function only supports input from events with a source type of: aws.batch")
       print(json.dumps(event))
   ```

   This is a simple Python 3.8 function that prints the events sent by AWS Batch. 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.
7. Choose **Deploy**.

**Step 2: Register Event Rule**

Next, you create a CloudWatch Events event rule that captures job events coming from your AWS Batch resources. This rule captures all events coming from AWS Batch within the account where it is defined. The job messages themselves contain information about the event source, including the job queue to which it was submitted, 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 permissions to call your Lambda function. If you are creating an event rule using the AWS CLI, you must grant permissions explicitly. For more information, see Events and Event Patterns in the *Amazon CloudWatch User Guide*.

To create your CloudWatch Events rule

2. On the navigation pane, choose **Events**, **Create rule**.
3. For **Event source**, select **Event Pattern** as the event source, and then select **Build custom event pattern**.
4. Paste the following event pattern into the text area.

   ```json
   {
     "source": [
       "aws.batch"
     ]
   }
   ```

   This rule applies to all AWS Batch events for all of your AWS Batch groups. Alternatively, you can create a more specific rule to filter out some results.
5. For **Targets**, choose **Add target**. For **Target type**, choose **Lambda function**, and select your Lambda function.
6. Choose **Configure details**.
7. For **Rule definition**, type a name and description for your rule and choose **Create rule**.

**Step 3: Test Your Configuration**

Finally, you can test your CloudWatch Events configuration by submitting a job to your job queue. If everything is configured properly, your Lambda function is triggered and it writes the event data to a CloudWatch Logs log stream for the function.

To test your configuration

2. Submit a new AWS Batch job. For more information, see Submitting a Job (p. 14).
4. On the navigation pane, choose **Logs** and select the log group for your Lambda function (for example, `/aws/lambda/my-function`).
5. Select a log stream to view the event data.
Tutorial: Sending Amazon Simple Notification Service Alerts for Failed Job Events

In this tutorial, you configure a CloudWatch Events event rule that only captures job events where the job has moved to a FAILED status. At the end of this tutorial, you can submit a job to this job queue to test that you have configured your Amazon SNS alerts correctly.

Prerequisites

This tutorial assumes that you have a working compute environment and job queue that are ready to accept jobs. If you do not have a running compute environment and job queue to capture events from, follow the steps in Getting Started with AWS Batch (p. 9) to create one.

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

2. Choose Topics, Create topic.
3. For Topic name, enter JobFailedAlert and choose Create topic.
4. Select the topic that you just created. On the Topic details: JobFailedAlert screen, choose Create subscription.
5. For Protocol, choose Email. For Endpoint, enter an email address to which you currently have access and choose Create subscription.
6. Check your email account, and wait to receive a subscription confirmation email message. When you receive it, choose Confirm subscription.

Step 2: Register Event Rule

Next, register an event rule that captures only job-failed events.

To create an event rule

2. In the navigation pane, choose Events, Create rule.
3. Choose Show advanced options, edit.
4. For Build a pattern that selects events for processing by your targets, replace the existing text with the following text:

```json
{
    "detail-type": [
        "Batch Job State Change"
    ],
    "source": [
        "aws.batch"
    ],
    "detail": {
        "status": ["FAILED"
    ]
}
```
This code defines a CloudWatch Events rule that matches any event where the job status is FAILED. For more information about event patterns, see Events and Event Patterns in the Amazon CloudWatch User Guide.

5. For Targets, choose Add target. For Target type, choose SNS topic, JobFailedAlert.
6. Choose Configure details.
7. For Rule definition, type a name and description for your rule and then choose Create rule.

**Step 3: Test Your Rule**

To test your rule, submit a job that exits shortly after it starts with a non-zero exit code. If your event rule is configured correctly, you receive an email message within a few minutes with the event text.

**To test a rule**

2. Submit a new AWS Batch job. For more information, see Submitting a Job (p. 14). For the job's command, substitute this command to exit the container with an exit code of 1.
   
   ```
   /bin/sh, -c, 'exit 1'
   ```
3. Check your email to confirm that you have received an email alert for the failed job notification.
Using CloudWatch Logs with AWS Batch

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

For information about sending logs from your jobs to CloudWatch Logs, see Using the awslogs log driver (p. 61). For more information about CloudWatch Logs, see Monitoring Log Files in the Amazon CloudWatch User Guide.

Topics
- CloudWatch Logs IAM Policy (p. 141)
- Installing and configuring the CloudWatch agent (p. 142)
- Viewing CloudWatch Logs (p. 142)

CloudWatch Logs IAM Policy

Before your jobs 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 ec2InstanceRole.

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:

   ```json
   {
   "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 Instance Role (p. 129) to create the role.
4. Choose Permissions, Attach policies.
5. To narrow the available policies to attach, for Filter, type ECS-CloudWatchLogs.
6. Select the ECS-CloudWatchLogs policy and choose Attach policy.

Installing and configuring the CloudWatch agent

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

For more information, see Download and configure the CloudWatch agent using the command line in the Amazon CloudWatch User Guide.

Viewing CloudWatch Logs

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

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

To view your CloudWatch Logs data

2. In the left navigation pane, choose Logs, Log groups.
3. Choose a log group to view.
4. Choose a log stream to view. By default, the streams are identified by the first 200 characters of the job name and the Amazon ECS task ID.
Logging AWS Batch API Calls with AWS CloudTrail

AWS Batch is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Batch. CloudTrail captures all API calls for AWS Batch as events. The calls captured include calls from the AWS Batch console and code calls to the AWS Batch API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Batch. 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 AWS Batch, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS Batch Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Batch, 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 AWS Batch, 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 AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- 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 AWS Batch actions are logged by CloudTrail and are documented in the https://docs.aws.amazon.com/batch/latest/APIReference/. For example, calls to the SubmitJob, ListJobs and DescribeJobs 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 AWS Identity and Access Management (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 AWS Batch 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 aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateComputeEnvironment` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2017-12-20T00:48:46Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      }
    },
    "eventTime": "2017-12-20T00:48:46Z",
    "eventSource": "batch.amazonaws.com",
    "eventName": "CreateComputeEnvironment",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.1",
    "userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
    "requestParameters": {
      "computeResources": {
        "subnets": [
          "subnet-5eda8e04"
        ],
        "tags": {
          "testBatchTags": "CLI testing CE"
        },
        "desiredVCpus": 0,
        "minVCpus": 0,
        "instanceTypes": [
          "optimal"
        ],
        "securityGroupIds": [
          "sg-aba9e8db"
        ],
        "instanceRole": "ecsInstanceRole",
        "maxVCpus": 128,
        "type": "EC2"
      },
      "state": "ENABLED",
      "type": "MANAGED",
      "serviceRole": "service-role/AWSBatchServiceRole",
      "computeEnvironmentName": "Test"
    }
  }
}```
"responseElements": {
  "computeEnvironmentName": "Test",
  "computeEnvironmentArn": "arn:aws:batch:us-east-1:012345678910:compute-environment/Test"
},
"requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
"eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
"readOnly": false,
"eventType": "AwsApiCall",
"recipientAccountId": "012345678910"}
Tutorial: Creating a VPC with Public and Private Subnets for Your Compute Environments

Compute resources in your compute environments need external network access to communicate with the Amazon ECS service endpoint. However, you might have jobs that you would like to run in private subnets. Creating a VPC with both public and private subnets provides you the flexibility to run jobs in either a public or private subnet. Jobs in the private subnets can access the internet through a NAT gateway.

This tutorial guides you through creating a VPC with two public subnets and two private subnets, which are provided with internet access through a NAT gateway.

Step 1: Create an Elastic IP Address for Your NAT Gateway

A NAT gateway requires an Elastic IP address in your public subnet, but the VPC wizard does not create one for you. Create the Elastic IP address before running the VPC wizard.

To create an Elastic IP address
1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose Elastic IPs.
3. Choose Allocate new address, Allocate, Close.
4. Note the Allocation ID for your newly created Elastic IP address; you enter this later in the VPC wizard.

Step 2: Run the VPC Wizard

The VPC wizard automatically creates and configures most of your VPC resources for you.

To run the VPC wizard
1. In the left navigation pane, choose VPC Dashboard.
2. Choose Launch VPC Wizard, VPC with Public and Private Subnets, Select.
3. For VPC name, give your VPC a unique name.
4. For Elastic IP Allocation ID, choose the ID of the Elastic IP address that you created earlier.
5. Choose Create VPC.
6. When the wizard is finished, choose OK. Note the Availability Zone in which your VPC subnets were created. Your additional subnets should be created in a different Availability Zone.

Non-default subnets, such as those created by the VPC wizard, are not auto-assigned public IPv4 addresses. Instances launched in the public subnet must be assigned a public IPv4 address to communicate with the Amazon ECS service endpoint.
To modify your public subnet's IPv4 addressing behavior

1. In the left navigation pane, choose Subnets.
2. Select the public subnet for your VPC. By default, the name created by the VPC wizard is Public subnet.
3. Choose Actions, Modify auto-assign IP settings.
4. Select the Enable auto-assign public IPv4 address check box, and then choose Save.

Step 3: Create Additional Subnets

The wizard creates a VPC with a single public and a single private subnet in a single Availability Zone. For greater availability, you should create at least one more of each subnet type in a different Availability Zone so that your VPC has both public and private subnets across two Availability Zones.

To create an additional private subnet

1. In the left navigation pane, choose Subnets.
2. Choose Create Subnet.
3. For Name tag, enter a name for your subnet, such as Private subnet.
4. For VPC, choose the VPC that you created earlier.
5. For Availability Zone, choose a different Availability Zone than your original subnets in the VPC.
6. For IPv4 CIDR block, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use 10.0.3.0/24 for your second private subnet.
7. Choose Yes, Create.

To create an additional public subnet

1. In the left navigation pane, choose Subnets and then Create Subnet.
2. For Name tag, enter a name for your subnet, such as Public subnet.
3. For VPC, choose the VPC that you created earlier.
4. For Availability Zone, choose the same Availability Zone as the additional private subnet that you created in the previous procedure.
5. For IPv4 CIDR block, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use 10.0.2.0/24 for your second public subnet.
6. Choose Yes, Create.
7. Select the public subnet that you just created and choose Route Table, Edit.
8. By default, the private route table is selected. Choose the other available route table so that the 0.0.0.0/0 destination is routed to the internet gateway (igw-xxxxxxxx) and choose Save.
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 an AWS Batch managed compute environment that launches compute resources into your new VPC. For more information, see Creating a compute environment (p. 93). If you use the compute environment creation wizard in the AWS Batch console, you can specify the VPC that you just created and the public or private subnets into which to launch your instances, depending on your use case.

• Create an AWS Batch job queue that is mapped to your new compute environment. For more information, see Creating a job queue (p. 76).

• Create a job definition to run your jobs with. For more information, see Creating a job definition (p. 32).

• Submit a job with your job definition to your new job queue. This job will land in the compute environment you created with your new VPC and subnets. For more information, see Submitting a Job (p. 14).
Security in AWS Batch

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 AWS Batch, 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 AWS Batch. The following topics show you how to configure AWS Batch to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Batch resources.

**Topics**
- Identity and Access Management for AWS Batch (p. 150)
- Compliance Validation for AWS Batch (p. 161)
- Infrastructure Security in AWS Batch (p. 162)

Identity and Access Management for AWS Batch

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 AWS Batch resources. IAM is an AWS service that you can use with no additional charge.

**Topics**
- Audience (p. 150)
- Authenticating with identities (p. 151)
- Managing access using policies (p. 152)
- How AWS Batch works with IAM (p. 154)
- AWS Batch execution IAM role (p. 157)
- AWS Batch identity-based policy examples (p. 158)
- Troubleshooting AWS Batch identity and access (p. 160)

**Audience**

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in AWS Batch.
Service user – If you use the AWS Batch service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Batch 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 AWS Batch, see Troubleshooting AWS Batch identity and access (p. 160).

Service administrator – If you're in charge of AWS Batch resources at your company, you probably have full access to AWS Batch. It's your job to determine which AWS Batch 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 AWS Batch, see How AWS Batch works with IAM (p. 154).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Batch. To view example AWS Batch identity-based policies that you can use in IAM, see Identity-Based Policy Examples (p. 158).

### Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.

### AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks, even the administrative ones. Instead, adhere to a 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 user 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 isn't associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations.

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the IAM User Guide.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions on your behalf. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. 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.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the IAM User Guide.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the IAM User Guide.

**Managing access using policies**

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions.
AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the IAM User Guide.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

**Identity-based policies**

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the IAM User Guide.

Identity-based policies can be further categorized as inline policies or managed policies. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the IAM User Guide.

**Resource-based policies**

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

**Access Control Lists (ACLs)**

Access control policies (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they are the only policy type that doesn't 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 aren't 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

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.

For more information about identity and access management for AWS Batch, continue to the following pages.

• How AWS Batch works with IAM (p. 154)
• Troubleshooting AWS Batch identity and access (p. 160)

How AWS Batch works with IAM

Before you use IAM to manage access to AWS Batch, you should understand what IAM features are available to use with AWS Batch. To get a high-level view of how AWS Batch and other AWS services work with IAM, see AWS Services That Work with IAM in the IAM User Guide.

Topics
• AWS Batch identity-based policies (p. 154)
• Authorization based on AWS Batch tags (p. 156)
• AWS Batch IAM roles (p. 156)

AWS Batch 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. AWS Batch supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the IAM User Guide.

Actions

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in AWS Batch use the following prefix before the action: batch:. For example, to grant someone permission to submit an AWS Batch job with the AWS Batch SubmitJob API operation, you include the batch:SubmitJob action in their policy. Policy statements must include either an Action
or `NotAction` element. AWS Batch 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": [
    "batch:action1",
    "batch:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, for example, include the following action.

```
"Action": "batch:Describe*"
```

To see a list of AWS Batch actions, see Actions Defined by AWS Batch.

**Resources**

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The AWS Batch job-definition resource has the following ARN.

```
arn:${Partition}:batch:${Region}:${Account}:job-definition/${JobDefinition}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `JobDefinitionAlpha` job definition in your statement, use the following ARN.

```
"Resource": "arn:aws:batch:us-east-1:123456789012:job-definition/JobDefinitionAlpha"
```

To specify all instances that belong to a specific account, use the wildcard (*).

```
```

Some AWS Batch actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Several AWS Batch API operation involves multiple resources. For example, `SubmitJob` submits a job to a job queue so an IAM user must have permissions to use the job definition and the job queue. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
    "resource1",
    "resource2"
]
To see a list of AWS Batch resource types and their ARNs, see Resources Defined by AWS Batch. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS Batch.

Condition keys

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the IAM User Guide.

AWS Batch 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.

The AWS Batch RegisterJobDefinition action supports the batch:User, batch:Privileged, and batch:Image condition keys. For more information, see the section called “Supported Resource-Level Permissions” (p. 117).

To see a list of AWS Batch condition keys, see Condition Keys for AWS Batch. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS Batch.

Examples

To view examples of AWS Batch identity-based policies, see Identity-Based Policy Examples (p. 158).

AWS Batch does support resource-based policies.

AWS Batch does support tagging resources and controlling access based on tags.

Authorization based on AWS Batch tags

You can attach tags to AWS Batch resources or pass tags in a request to AWS Batch. To control access based on tags, you provide tag information in the condition element of a policy using the batch:ResourceTag/key-name, aws:RequestTag/key-name, or aws:TagKeys condition keys. For more information about tagging AWS Batch resources, see Tagging your AWS Batch resources (p. 163).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Identity-Based Policy Examples (p. 158).

AWS Batch IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with AWS Batch

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.

AWS Batch 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.

AWS Batch does not support service-linked roles.

**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.

AWS Batch supports service roles.

**AWS Batch execution IAM role**

The execution role grants the Amazon ECS container and AWS Fargate agents permission to make AWS API calls on your behalf. The execution IAM role is required depending on the requirements of your task. You can have multiple execution roles for different purposes and services associated with your account.

*Note*

The 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 execution role for special use cases which are outlined below.

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

An execution role is automatically created for you in the AWS Batch console first-run experience; however, you should manually attach the managed IAM policy for tasks to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. You can use the following procedure to check and see if your account already has the execution role and to attach the managed IAM policy if needed.

**To check for the `ecsTaskExecutionRole` in the IAM console**

2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsTaskExecutionRole`. If the role does not exist, see Creating the execution IAM role (p. 158). If the role does exist, select the role to view the attached policies.
4. On the Permissions tab, verify that the AmazonECSTaskExecutionRolePolicy managed policy is attached to the role. If the policy is attached, your 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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

**Creating the execution IAM role**

If your account does not already have an execution role, use the following steps to create the role.

**To create the ecsTaskExecutionRole IAM role**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create role.
3. In the Select type of trusted entity section, choose Elastic Container Service.
4. For Select your use case, choose Elastic Container Service Task, then choose Next: Permissions.
5. In the Attach permissions policy section, search for AmazonECSTaskExecutionRolePolicy, select the policy, and then choose Next: Review.
6. For Role Name, type ecsTaskExecutionRole and choose Create role.

**AWS Batch identity-based policy examples**

By default, IAM users and roles don't have permission to create or modify AWS Batch 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. 159)
Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete AWS Batch 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 AWS Batch 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.

```json
{
  "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",
      ]
    }
  ]
}
```
Troubleshooting AWS Batch identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS Batch and IAM.

**Topics**
- I am not authorized to perform an action in AWS Batch (p. 160)
- I am not authorized to perform `iam:PassRole` (p. 160)
- I want to view my access keys (p. 161)
- I'm an administrator and want to allow others to access AWS Batch (p. 161)
- I want to allow people outside of my AWS account to access my AWS Batch resources (p. 161)

**I am not authorized to perform an action in AWS Batch**

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 compute environment but does not have `batch:DescribeComputeEnvironments` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: batch:DescribeComputeEnvironments on resource: my-example-compute-environment
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-compute-environment` resource using the `batch:DescribeComputeEnvironments` action.

**I am not authorized to perform `iam:PassRole`**

If you receive an error that you aren't 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 AWS Batch.

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 AWS Batch. However, the action requires the service to have permissions granted by a service role. Mary doesn't have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor isn't 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, wJalrXUtFnFEMI/K7MDENG/bPxRfiCYEXAMPELEKEY). 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**  
Don't 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 AWS Batch

To allow others to access AWS Batch, 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 AWS Batch.

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 AWS Batch 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 AWS Batch supports these features, see How AWS Batch works with IAM (p. 154).
- 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.

Compliance Validation for AWS Batch

Third-party auditors assess the security and compliance of AWS Batch as part of multiple AWS compliance programs. These include SOC, PCI, ISO, FedRAMP, DoD CC SRG, HIPAA BAA, IRAP, MTCS, ENS High, OSPAR, HITRUST CSF, 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 AWS Batch 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 AWS Batch

As a managed service, AWS Batch 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 AWS Batch 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.
Tagging your AWS Batch resources

To help you manage your AWS Batch resources, you can assign your own metadata to each resource in the form of tags. This topic describes tags and shows you how to create them.

Contents
- Tag basics (p. 163)
- Tagging your resources (p. 163)
- Tag restrictions (p. 164)
- Working with tags using the console (p. 164)
- Working with tags using the CLI or API (p. 165)

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 by, for example, purpose, owner, or environment. 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 AWS Batch services to help you track each service's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to AWS Batch and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value.

You can work with tags using the AWS Management Console, the AWS CLI, and the AWS Batch API.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

Tagging your resources

You can tag new or existing AWS Batch compute environments, jobs, job definitions, and job queues.

If you're using the AWS Batch console, you can apply tags to new resources when they are created or to existing resources at any time using the Tags tab on the relevant resource page.

If you're using the AWS Batch 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 to existing resources using the TagResource API action. For more information, see TagResource.

Some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, the resource creation process fails. This ensures that resources you intended to tag on creation are either created with specified tags or not created at all. If you tag resources at the time of creation, you don't need to run custom tagging scripts after resource creation.
The following table describes the AWS Batch resources that can be tagged, and the resources that can be tagged on creation.

Tagging support for AWS Batch resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Supports tags</th>
<th>Supports tag propagation</th>
<th>Supports tagging on creation (AWS Batch API, AWS CLI, AWS SDK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Batch compute environments</td>
<td>Yes</td>
<td>No. Compute environment tags do not propagate to any other resources.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tags for the resources are specified in the tags member of the computeResources object passed in the CreateComputeEnvironment API operation.</td>
<td></td>
</tr>
<tr>
<td>AWS Batch jobs</td>
<td>Yes</td>
<td>No. Tags do not propagate to child jobs for array or multi-node parallel (MNP) jobs.</td>
<td>Yes</td>
</tr>
<tr>
<td>AWS Batch job definitions</td>
<td>Yes</td>
<td>No.</td>
<td>Yes</td>
</tr>
<tr>
<td>AWS Batch job queues</td>
<td>Yes</td>
<td>No.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple AWS services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case sensitive.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values, as it is reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags-per-resource limit.

Working with tags using the console

Using the AWS Batch console, you can manage the tags associated with new or existing compute environments, jobs, job definitions, and job queues.
Adding tags on an individual resource on creation

You can add tags to AWS Batch compute environments, jobs, job definitions, and job queues when you create them.

Adding and deleting tags on an individual resource

AWS Batch allows you to add or delete tags associated with your clusters directly from the resource's page.

To add or delete a tag on an individual resource

2. From the navigation bar, choose the Region to use.
3. In the navigation pane, choose a resource type (for example, Job Queues).
4. Choose a specific resource, then choose Edit tags.
5. Add or delete your tags as necessary.
   - To add a tag — specify the key and value in the empty text boxes at the end of the list.
   - To delete a tag — choose the button next to the tag.
6. Repeat this process for each tag you want to add or delete, and then choose Edit tags to finish.

Working with tags using the CLI or API

Use the following AWS CLI commands or AWS Batch API operations to add, update, list, and delete the tags for your resources.

Tagging support for AWS Batch resources

<table>
<thead>
<tr>
<th>Task</th>
<th>API action</th>
<th>AWS CLI</th>
<th>AWS Tools for Windows PowerShell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add or overwrite one or more tags.</td>
<td>TagResource</td>
<td>tag-resource</td>
<td>Add-BATResourceTag</td>
</tr>
<tr>
<td>Delete one or more tags</td>
<td>UntagResource</td>
<td>untag-resource</td>
<td>Remove-BATResourceTag</td>
</tr>
<tr>
<td>List tags for a resource</td>
<td>ListTagsForResource</td>
<td>list-tags-for-resource</td>
<td>Get-BATResourceTag</td>
</tr>
</tbody>
</table>

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing resource

The following command tags an existing resource.

```
aws batch tag-resource --resource-arn resource_ARN --tags team=devs
```

Example 2: Untag an existing resource

The following command deletes a tag from an existing resource.
aws batch 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 batch list-tags-for-resource --resource-arn resource_ARN

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

<table>
<thead>
<tr>
<th>Task</th>
<th>API action</th>
<th>AWS CLI</th>
<th>AWS Tools for Windows PowerShell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a compute environment</td>
<td>CreateComputeEnvironment</td>
<td>create-compute-environment</td>
<td>New-BATComputeEnvironment</td>
</tr>
<tr>
<td>Create a job queue</td>
<td>CreateJobQueue</td>
<td>create-job-queue</td>
<td>New-BATJobQueue</td>
</tr>
<tr>
<td>Register a job definition</td>
<td>RegisterJobDefinition</td>
<td>register-job-definition</td>
<td>Register-BATJobDefinition</td>
</tr>
<tr>
<td>Submit a job</td>
<td>SubmitJob</td>
<td>submit-job</td>
<td>Submit-BATJob</td>
</tr>
</tbody>
</table>
AWS Batch service limits

The following table provides the service limits for AWS Batch that can't be changed. Each limit is Region specific.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of job queues. For more information, see <em>Job queues (p. 76).</em></td>
<td>20</td>
</tr>
<tr>
<td>Maximum number of compute environments. For more information, see <em>Compute environment (p. 81).</em></td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of compute environments for each job queue</td>
<td>3</td>
</tr>
<tr>
<td>Maximum number of job dependencies for a job</td>
<td>20</td>
</tr>
<tr>
<td>Maximum job definition size (for <em>RegisterJobDefinition</em> API operations)</td>
<td>24 KiB</td>
</tr>
<tr>
<td>Maximum job payload size (for <em>SubmitJob</em> API operations)</td>
<td>30 KiB</td>
</tr>
<tr>
<td>Maximum array size for array jobs</td>
<td>10000</td>
</tr>
<tr>
<td>Maximum number of jobs in SUBMITTED state</td>
<td>1000000</td>
</tr>
</tbody>
</table>

Depending on how you use AWS Batch, additional quotas might apply. To learn about Amazon EC2 quotas, see *Amazon EC2 Service Quotas* in the *AWS General Reference*. For more information about Amazon ECS quotas, see *Amazon ECS Service Quotas* in the *AWS General Reference*. 
Troubleshooting AWS Batch

You might find the need to troubleshoot issues with your compute environments, job queues, job definitions, or jobs. This chapter helps you troubleshoot and repair issues with your AWS Batch environment.

INVALID Compute Environment

It is possible to incorrectly configure a managed compute environment so that it enters an INVALID state and cannot accept jobs for placement. These sections describe the possible causes and how to fix them.

Incorrect Role Name or ARN

The most common cause for invalid compute environments is an incorrect name or ARN for the AWS Batch service role or the Amazon EC2 Spot Fleet role. This is more of an issue for compute environments that are created with the AWS CLI or the AWS SDKs; when you create a compute environment in the AWS Management Console, AWS Batch can help you choose the correct service or Spot Fleet roles and you cannot misspell the name or deform the ARN.

However, if you manually type the name or ARN for an IAM in an AWS CLI command or your SDK code, AWS Batch is unable to validate the string and it accepts the bad value and attempts to create the environment. After failing to create the environment, the environment moves to an INVALID state, and you see the following errors.

For an invalid service role:

```
CLIENT_ERROR - Not authorized to perform sts:AssumeRole (Service: AWSSecurityTokenService; Status Code: 403; Error Code: AccessDenied; Request ID: dc0e2d28-2e99-11e7-b372-7fcc6fb65fe7)
```

For an invalid Spot Fleet role:

```
CLIENT_ERROR - Parameter: SpotFleetRequestConfig.IamFleetRole is invalid. (Service: AmazonEC2; Status Code: 400; Error Code: InvalidSpotFleetRequestConfig; Request ID: 331205f0-5ae3-4cea-bac4-897769639f8d) Parameter: SpotFleetRequestConfig.IamFleetRole is invalid
```

One common cause for this issue is if you only specify the name of an IAM role when using the AWS CLI or the AWS SDKs, instead of the full ARN. This is because depending on how you created the role, the ARN might contain a service-role path prefix. For example, if you manually create the AWS Batch service role using the procedures in AWS Batch Service IAM Role (p. 126), your service role ARN would look like this:

```
arn:aws:iam::123456789012:role/AWSBatchServiceRole
```

However, if you created the service role as part of the console first run wizard today, your service role ARN would look like this:

```
arn:aws:iam::123456789012:role/service-role/AWSBatchServiceRole
```
When you only specify the name of an IAM role when using the AWS CLI or the AWS SDKs, AWS Batch assumes that your ARN does not use the service-role path prefix. Because of this, we recommend that you specify the full ARN for your IAM roles when you create compute environments.

To repair a compute environment that's misconfigured this way, see Repairing an INVALID Compute Environment (p. 169).

Repairing an INVALID Compute Environment

When you have a compute environment in an INVALID state, you should update it to repair the invalid parameter. For the case of an Incorrect Role Name or ARN (p. 168), you can update the compute environment with the correct service role.

To repair a misconfigured compute environment

2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose Compute environments.
4. On the Compute environments page, select the radio button next to the compute environment to edit, and then choose Edit.
5. On the Update compute environment page, for Service role, choose the IAM role to use with your compute environment. The AWS Batch console only displays roles that have the correct trust relationship for compute environments.
6. Choose Save to update your compute environment.

Jobs Stuck in RUNNABLE Status

If your compute environment contains compute resources, but your jobs don't progress beyond the RUNNABLE status, then there is something preventing the jobs from actually being placed on a compute resource. Here are some common causes for this issue:

The awslogs log driver isn't configured on your compute resources

AWS Batch jobs send their log information to CloudWatch Logs. To enable this, you must configure your compute resources to use the awslogs log driver. If you base your compute resource AMI off of the Amazon ECS-optimized AMI (or Amazon Linux), then this driver is registered by default with the ecs-init package. If you use a different base AMI, then you must ensure that the awslogs log driver is specified as an available log driver with the ECS_AVAILABLE_LOGGING_DRIVERS environment variable when the Amazon ECS container agent is started. For more information, see Compute resource AMI specification (p. 83) and Creating a compute resource AMI (p. 83).

Insufficient resources

If your job definitions specify more CPU or memory resources than your compute resources can allocate, then your jobs will never be placed. For example, if your job specifies 4 GiB of memory, and your compute resources have less than that available, then the job cannot be placed on those compute resources. In this case, you must reduce the specified memory in your job definition or add larger compute resources to your environment. Some memory is reserved for the Amazon ECS container agent and other critical system processes. For more information, see Compute Resource Memory Management (p. 107).

No internet access for compute resources

Compute resources need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your compute resources having public IP addresses.
For more information about interface VPC endpoints, see Amazon ECS Interface VPC Endpoints (AWS PrivateLink) in the Amazon Elastic Container Service Developer Guide.

If you do not have an interface VPC endpoint configured and your compute resources 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. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Compute Environments (p. 147).

Amazon EC2 instance limit reached

The number of Amazon EC2 instances that your account can launch in an AWS Region is determined by your EC2 instance limit. Certain instance types have a per-instance-type limit as well. For more information on your account’s Amazon EC2 instance limits (including how to request a limit increase), see Amazon EC2 Service Limits in the Amazon EC2 User Guide for Linux Instances.

For more information on diagnosing jobs stuck in RUNNABLE status, see Why is my AWS Batch job stuck in RUNNABLE status? in the AWS Knowledge Center.

Spot Instances Not Tagged on Creation

Spot Instance tagging for AWS Batch compute resources is supported as of October 25, 2017. Prior to that support, the recommended IAM managed policy (AmazonEC2SpotFleetRole) for the Amazon EC2 Spot Fleet role did not contain permissions to tag Spot Instances at launch. The new recommended IAM managed policy is called AmazonEC2SpotFleetTaggingRole.

To fix Spot Instance tagging on creation, follow the following procedure to apply the current recommended IAM managed policy to your Amazon EC2 Spot Fleet role, and then any future Spot Instances that are created with that role have permissions to apply instance tags on creation.

To apply the current IAM managed policy to your Amazon EC2 Spot Fleet role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Roles, and choose your Amazon EC2 Spot Fleet role.
3. Choose Attach policy.
4. Select the AmazonEC2SpotFleetTaggingRole and choose Attach policy.
5. Choose your Amazon EC2 Spot Fleet role again to remove the previous policy.
6. Select the x to the right of the AmazonEC2SpotFleetRole policy, and choose Detach.
## Document history

The following table describes the important changes to the documentation since the initial release of AWS Batch. We also update the documentation frequently to address the feedback that you send us.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Fargate support</td>
<td>AWS Batch adds support running jobs on Fargate resources.</td>
<td>December 3, 2020</td>
</tr>
<tr>
<td>Amazon Linux 2 support</td>
<td>AWS Batch adds support for automatic selection of Amazon Linux 2 AMIs in the compute environment using the EC2 Configuration parameters.</td>
<td>November 24, 2020</td>
</tr>
<tr>
<td>Enhanced retry strategy</td>
<td>AWS Batch enhances the retry strategy for jobs. Now jobs can be retried or stop further retries by matching the ExitCode, Reason, or StatusReason of a job with patterns.</td>
<td>October 20, 2020</td>
</tr>
<tr>
<td>Resource tagging</td>
<td>AWS Batch adds support for adding metadata tags to your compute environments, job definitions, job queues, and jobs.</td>
<td>October 7, 2020</td>
</tr>
<tr>
<td>Secrets</td>
<td>AWS Batch adds support for passing secrets to jobs.</td>
<td>October 1, 2020</td>
</tr>
<tr>
<td>Logging</td>
<td>AWS Batch adds support for specifying additional log drivers for jobs.</td>
<td>October 1, 2020</td>
</tr>
<tr>
<td>Allocation strategies</td>
<td>AWS Batch adds support for multiple strategies to choose instance types.</td>
<td>October 16, 2019</td>
</tr>
<tr>
<td>EFA support</td>
<td>AWS Batch adds support for Elastic Fabric Adapter (EFA) devices.</td>
<td>August 2, 2019</td>
</tr>
<tr>
<td>GPU scheduling</td>
<td>AWS Batch adds GPU scheduling. With this feature, you can specify the number of GPUs each job requires, and AWS Batch scales up instances accordingly.</td>
<td>April 4, 2019</td>
</tr>
<tr>
<td>Multi-node parallel jobs</td>
<td>AWS Batch adds support for multi-node parallel jobs. You can use this feature run single jobs that span over multiple Amazon EC2 instances.</td>
<td>November 19, 2018</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Resource-level permissions</td>
<td>AWS Batch supports resource-level permissions on several API operations.</td>
<td>November 12, 2018</td>
</tr>
<tr>
<td>Amazon EC2 Launch template support</td>
<td>AWS Batch adds support for using launch templates with compute environments.</td>
<td>November 12, 2018</td>
</tr>
<tr>
<td>AWS Batch job timeouts</td>
<td>AWS Batch adds support for job timeout. With this support, you can configure a specific timeout duration for your jobs so that if a job runs longer than they should, AWS Batch terminates the job.</td>
<td>April 5, 2018</td>
</tr>
<tr>
<td>AWS Batch jobs as CloudWatch Events targets</td>
<td>AWS Batch jobs are made available as CloudWatch Events targets. By creating simple rules, you can match events and submit AWS Batch jobs in response to them.</td>
<td>March 1, 2018</td>
</tr>
<tr>
<td>CloudTrail auditing for AWS Batch Array jobs</td>
<td>AWS Batch adds support for array jobs. You can use array jobs for parameter sweep and Monte Carlo workloads.</td>
<td>November 28, 2017</td>
</tr>
<tr>
<td>Expanded AWS Batch tagging</td>
<td>AWS Batch expands support for the tagging function. You can use this function to specify tags for Amazon EC2 Spot Instances launched within managed compute environments.</td>
<td>October 26, 2017</td>
</tr>
<tr>
<td>AWS Batch event stream for CloudWatch Events</td>
<td>AWS Batch adds the event stream for CloudWatch Events. You can use AWS Batch event stream to receive near real-time notifications regarding the state of jobs that are submitted to your job queues.</td>
<td>October 24, 2017</td>
</tr>
<tr>
<td>Automated job retries</td>
<td>AWS Batch adds support for job retries. With this update, you can apply a retry strategy to your jobs and job definitions that allows your jobs to be automatically retried if they fail.</td>
<td>March 28, 2017</td>
</tr>
<tr>
<td>AWS Batch general availability (p. 171)</td>
<td>AWS Batch is introduced, designed as a means for you to run batch computing workloads on the AWS Cloud.</td>
<td>January 5, 2017</td>
</tr>
</tbody>
</table>