AWS Batch

User Guide

AWS Batch: User Guide

Table of Contents

What Is AWS Batch?

AWS Batch enables 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 enables 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 an Amazon EC2 instance 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. 13).

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. 30)

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 instance types at several levels

of detail. You can set up compute environments that use a particular type of instance, a particular model such as `c4.2xlarge` or `m4.10xlarge`, 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 a percentage value for bids on the Spot Market and a target set of VPC subnets. AWS Batch will efficiently launch, manage, and terminate EC2 instances 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 Environments (p. 51).

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. 4)
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. 6)
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

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

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

2. Follow the online instructions.

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

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

Create an IAM User

Services in AWS, such as Amazon EC2 and 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 IAM user for yourself and add the user to an Administrators group

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

   Note
   We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
3. For **User name**, type **`Administrator`**.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, for **Group name** type **`Administrators`**.
9. For **Filter policies**, select the check box for **AWS managed - job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Tags** to add metadata to the user by attaching tags as key-value pairs.
13. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

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

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

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

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

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

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

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. 77) and Amazon ECS Instance Role (p. 79) 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 is easy for you to remember, such as your IAM user name, followed by `-key-pair`, plus the region name. For example, *me*-key-pair-*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
```

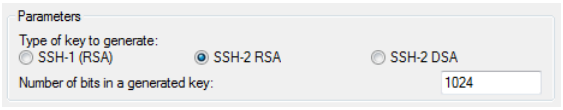
For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*.

To connect to your instance using your key pair

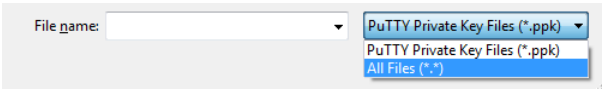
To connect to your Linux instance from a computer running Mac or Linux, specify the `.pem` file to your SSH client with the `-i` option and the path to your private key. To connect to your Linux instance from a computer running Windows, you can use either MindTerm or PuTTY. If you plan to use PuTTY, 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 **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 is easy for you to 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 do not 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

1. Open the AWS Batch console first-run wizard at https://console.aws.amazon.com/batch/home#/wizard.
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. 38).
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. 17).

Parameters

You can optionally specify parameter substitution default values and placeholders in your command. For more information, see Parameters (p. 38).

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 do not 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. 60).

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. 77). 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. 79). 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 below 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. 80).
3. For **Allowed instance types**, choose the Amazon EC2 instance types that may launched. You can specify instance families to launch any instance type within those families (for example, `c4` or `p3`), or you can specify specific sizes within a family (such as `c4.8xlarge`). You can also choose `optimal` to pick instance types (from the latest C, M, and R 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.
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 external network access to communicate with the Amazon ECS service endpoint, so if 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. 94).

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 execute 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. 37).

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

Topics

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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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. 30).
6. For **Job queue**, choose a previously created job queue. For more information, see Creating a Job Queue (p. 47).
7. For **Job type**, choose **Single** for a single job or **Array** to submit an array job. For more information, see Array Jobs (p. 18). This option is not 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. 17).

    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 index job 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. 17).
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. 18).
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. 38).

    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. 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. 38).
16. (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.
17. 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 *`jobDefinitionName`*`/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`.

`PENDING`

A job that resides in the queue and is not 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. 99) 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. 17).

Note
Logs for `RUNNING` jobs are available in CloudWatch Logs; the log group is `/aws/batch/job`, and the log stream name format is *`jobDefinitionName`*`/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*.

`SUCCEEDED`

The job has successfully completed with an exit code of `0`. The job state for `SUCCEEDED` jobs is persisted in AWS Batch for 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 *`jobDefinitionName`*`/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*.

`FAILED`

The job has failed all available attempts. The job state for `FAILED` jobs is persisted in AWS Batch for 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 *`jobDefinitionName`*`/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 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_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. 22).

`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. 17).

`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 is not 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. 44).

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. 15).

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. 30) and Submitting a Job (p. 13).

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. 18).

To submit an AWS Batch job with dependencies, see Submitting a Job (p. 13).

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 should not 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 is not 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.

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. 13).

Array Jobs

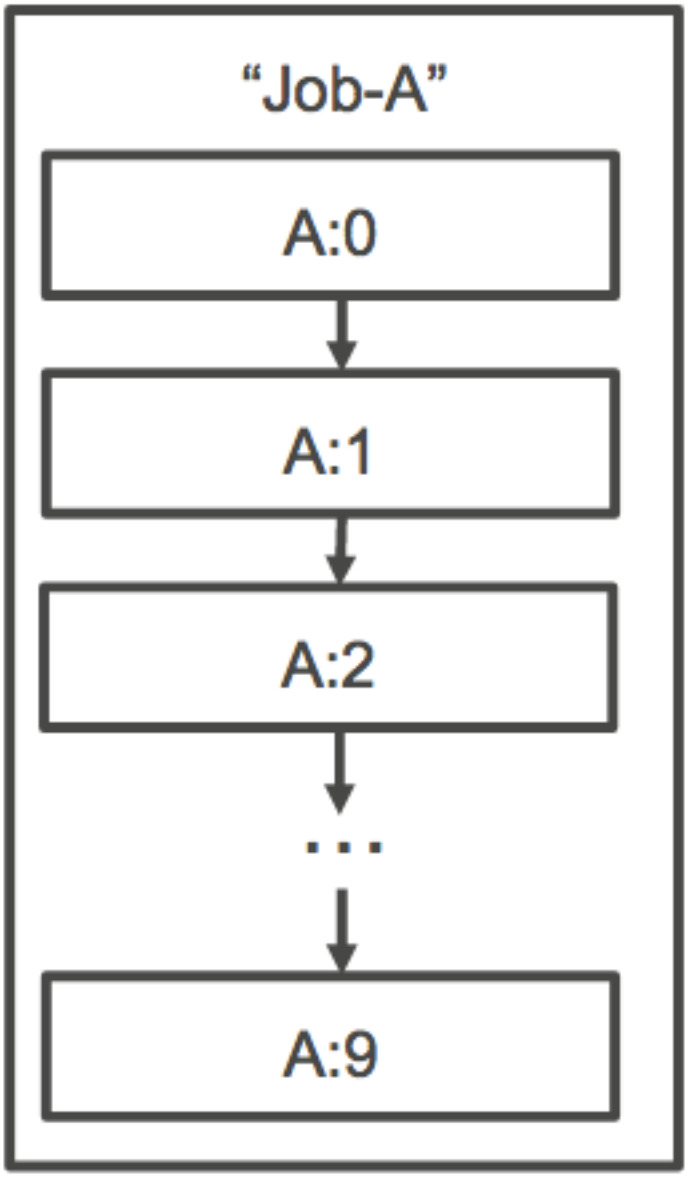
An array job is a job that shares common parameters, such as the job definition, vCPUs, and memory. It runs as a collection of related, yet separate, basic jobs that may be distributed across multiple hosts and may run concurrently. Array jobs are the most efficient way to execute embarrassingly parallel 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.

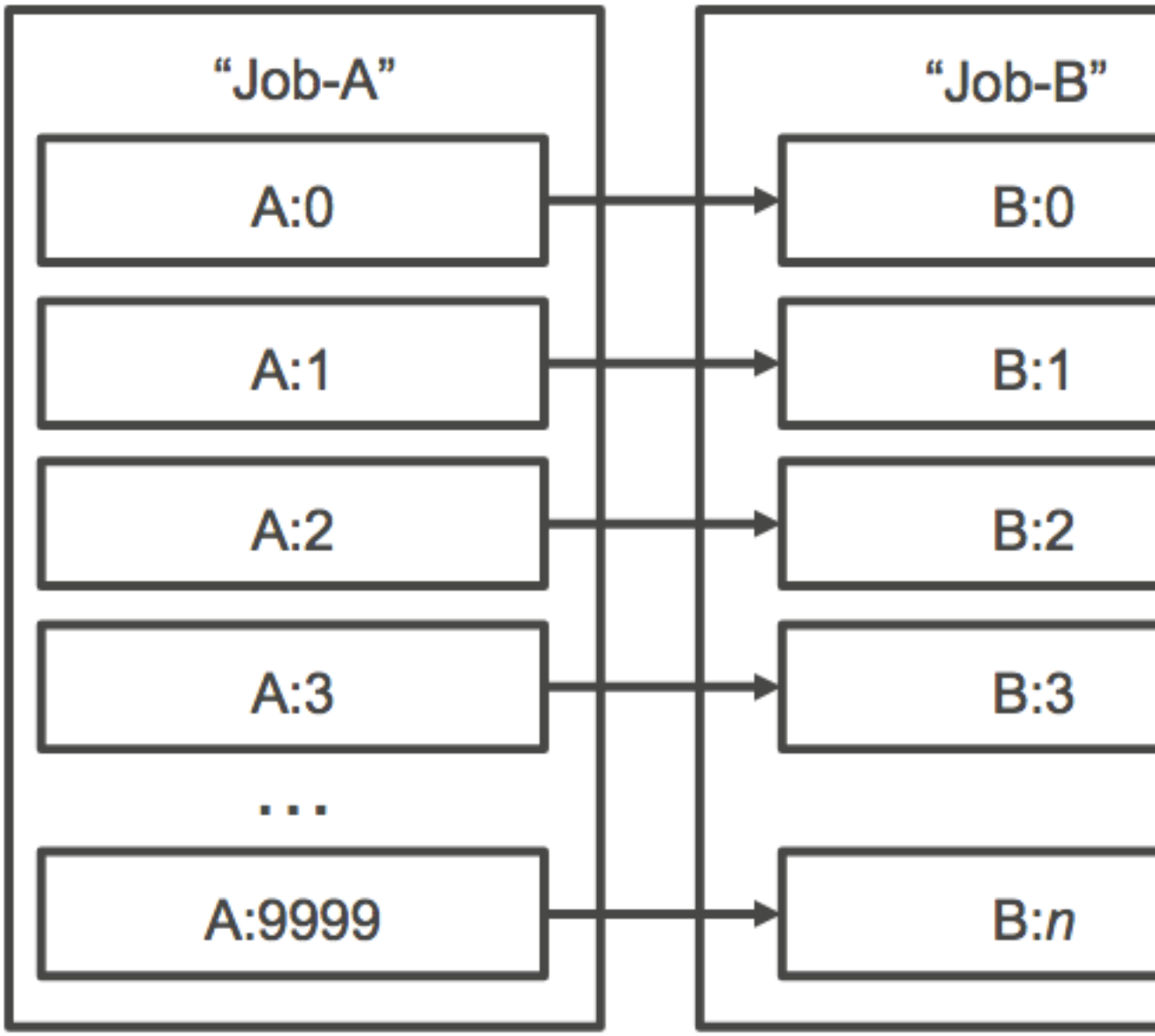
At runtime, the `AWS_BATCH_JOB_ARRAY_INDEX` environment variable is set to the container's corresponding job array index number. The first array job index is numbered `0`, and subsequent attempts are in ascending order (1, 2, 3, and so on). You can use this index value to control how your array job children are differentiated. For more information, see Tutorial: Using the Array Job Index to Control Job Differentiation (p. 22).

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 `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 4 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.

```
{
    "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.

```
{
    "jobName": "JobB",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobB-CPU-Intensive-Processing:1",
    "containerOverrides": {
        "vcpus": 32,
        "memory": 4096
   }
    "arrayProperties": {
        "size": 10000
    },
    "dependsOn": [
        {
            "jobId": "JobA_job_ID"
  }
    ]
}
```

- `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.

```
{
    "jobName": "JobC",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobC-Memory-Intensive-Processing:1",
    "containerOverrides": {
        "vcpus": 1,
        "memory": 32768
   }
    "arrayProperties": {
        "size": 10000
    },
    "dependsOn": [
        {
            "jobId": "JobB_job_ID",
            "type": "N_TO_N"
        }
    ]
}
```

- `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.

```
{
    "jobName": "JobD",
    "jobQueue": "ProdQueue",
    "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. 60).
- An AWS Batch job queue and associated compute environment. For more information, see Creating a Job Queue (p. 47).
- 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.

   Note
   The `LINE` variable is set to the `AWS_BATCH_JOB_ARRAY_INDEX` + 1 because the array index starts at 0, but line numbers start at 1. The `COLOR` variable is set to the color in `colors.txt` that is associated with its line number.

```
#!/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.

```
FROM busybox
COPY print-color.sh /tmp/print-color.sh
COPY colors.txt /tmp/colors.txt
ENTRYPOINT /tmp/print-color.sh
```

5. Build your Docker image:

```
docker build -t print-color .
```

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.

```
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*.

```
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.

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

3. Retrieve the login command for your Amazon ECR registry. For more information, see Registry Authentication in the *Amazon Elastic Container Registry User Guide*.

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

4. Run the **docker login ...** command that was returned by the previous step.
5. Push your image to Amazon ECR:

```
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. 30).

To create a job definition

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

```
{
  "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:

```
aws batch register-job-definition --cli-input-json file://print-color.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-def.json` in your workspace directory and paste the contents below into it.

   Note
   This example assumes the default job queue name that is created by the AWS Batch first-run wizard. If your job queue name is different, replace the `first-run-job-queue` name with your job queue name.

```
{
  "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.

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

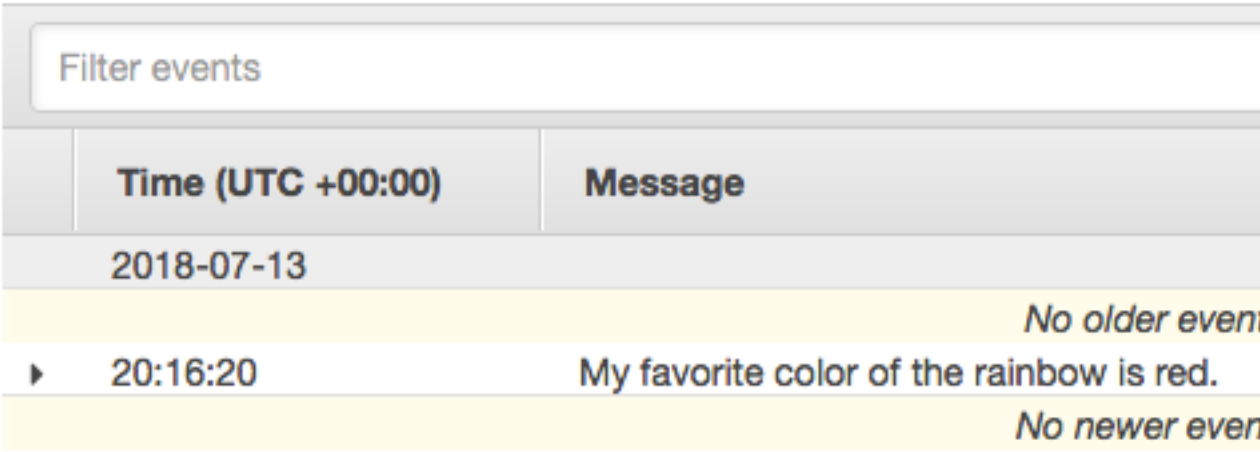
3. Describe the job's status and wait for the job to move to `SUCCEEDED`.

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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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. 27).

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 is not 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.

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

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. 37).

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 is not a multi-node parallel job.

To create a multi-node parallel job definition, see Creating a Multi-node Parallel Job Definition (p. 33). For more information about multi-node parallel jobs, see Multi-node Parallel Jobs (p. 26).

To create a new job definition

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Job definitions**, **Create**.
4. For **Job definition name**, enter a unique name for your job definition. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
5. 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. 17).
6. (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. 18).

7. For **Job requires multiple node configurations**, leave this box unchecked. To create a multi-node parallel job definition instead, see Creating a Multi-node Parallel Job Definition (p. 33).
8. (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. 38).

   a. Choose **Add parameter**.

   b. For **Key**, specify the key for your parameter.

   c. For **Value**, specify the value for your parameter.
9. (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
   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*.
10. 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`).
11. 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 as well as placeholders in your command. For more information, see Parameters (p. 38).
12. 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.
13. 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 are 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. 68).

14. (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**.
15. (Optional) Specify 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**.
16. (Optional) 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 does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.
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. For **Ulimits**, 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.
19. 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. 26).

To create a single-node job definition, see Creating a Job Definition (p. 30).

To create a multi-node parallel job definition

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Job definitions**, **Create**.
4. For **Job definition name**, enter a unique name for your job definition. Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
5. 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. 17).
6. (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. 18).
7. Select **Job requires multiple node configurations** and then complete the following substeps. To create a single node parallel job definition instead, see Creating a Job Definition (p. 30).

   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. (Optional) To restrict your nodes to a particular instance type, choose one from the drop-down menu. If you do not specify an instance type, AWS Batch chooses the smallest instance type that meets the requirements for your largest node (vCPU and memory) from the available instance types in your compute environment.

      Important
      Be sure to choose an instance type that is available for launch in your compute environment. Otherwise, your job gets stuck in the `RUNNABLE` status and block subsequent jobs.
8. (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. 38).

   a. Choose **Add parameter**.
   b. For **Key**, specify the key for your parameter.
   c. For **Value**, specify the value for your parameter.
9. In the **Node properties** section, configure your node groups. By default, a single node group is created for you with the default number of nodes.
10. 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 7.a (p. 33), 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.
11. 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`).

12. 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.
13. 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 are 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. 68).
14. 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 as well as placeholders in your command. For more information, see Parameters (p. 38).
15. (Optional) 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**.
16. (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
    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*.
17. 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**.
18. (Optional) Specify 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**.

19. (Optional) 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 does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.

20. (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.

21. For **Ulimits**, 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.

22. Return to Step 10 (p. 33) and repeat for each node group to configure for your job.
23. Choose **Create job definition**.

Job Definition Template

An empty job definition template is shown below. 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. 37).

```
{
    "jobDefinitionName": "",
    "type": "container",
    "parameters": {
        "KeyName": ""
    },
    "containerProperties": {
        "image": "",
        "vcpus": 0,
        "memory": 0,
        "command": [
```

```
            ""
        ],
        "jobRoleArn": "",
        "volumes": [
            {
                "host": {
                    "sourcePath": ""
                },
                "name": ""
            }
        ],
        "environment": [
            {
                "name": "",
                "value": ""
            }
        ],
        "mountPoints": [
            {
                "containerPath": "",
                "readOnly": true,
                "sourceVolume": ""
            }
        ],
        "readonlyRootFilesystem": true,
        "privileged": true,
        "ulimits": [
            {
                "hardLimit": 0,
                "name": "",
                "softLimit": 0
            }
        ],
        "user": "",
        "instanceType": ""
    },
    "nodeProperties": {
        "numNodes": 0,
        "mainNode": 0,
        "nodeRangeProperties": [
            {
                "targetNodes": "",
                "container": {
                    "image": "",
                    "vcpus": 0,
                    "memory": 0,
                    "command": [
                        ""
                    ],
                    "jobRoleArn": "",
                    "volumes": [
                        {
                            "host": {
                                "sourcePath": ""
                            },
                            "name": ""
                        }
                    ],
                    "environment": [
                        {
                            "name": "",
                            "value": ""
                        }
                    ],
                    "mountPoints": [
                        {
```

```
                            "containerPath": "",
                            "readOnly": true,
                            "sourceVolume": ""
                        }
                    ],
                    "readonlyRootFilesystem": true,
                    "privileged": true,
                    "ulimits": [
                        {
                            "hardLimit": 0,
                            "name": "",
                            "softLimit": 0
                        }
                    ],
                    "user": "",
                    "instanceType": ""
                }
            }
        ]
    },
    "retryStrategy": {
        "attempts": 0
    },
    "timeout": {
        "attemptDurationSeconds": 0
    }
}
```

Note
You can generate the above job definition template with the following AWS CLI command:

```
$ aws batch register-job-definition --generate-cli-skeleton
```

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

`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 is 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. At this time, only container jobs are supported.

Type: String

Valid values: `container`

Required: Yes

Parameters

`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

Required: No

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`.

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. 43) level, for each node group.

`command`

The command that is passed to the container. This parameter maps to `Cmd` in the Create a container section of the Docker Remote API and the `COMMAND` parameter to **docker run**. For more information about the Docker `CMD` parameter, see 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 do not recommend using plaintext environment variables for sensitive information, such as credential data.

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" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`image`

The image used to start a container. This string is passed directly to the Docker daemon. Images in the Docker Hub registry are available by default. You can also specify other repositories with *`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

`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

`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.

Note
If you are 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. 68).

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. The default value is `false`.

Type: Boolean

Required: No

`privileged`

When this parameter is true, the container is given elevated privileges on the host container instance (similar to the `root` user). This parameter maps to `Privileged` in the Create a container section of the Docker Remote API and the `--privileged` option to **docker run**.

```
"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

`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.

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: Yes

`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 is not guaranteed to persist after the container associated with it stops running.

Type: Object

Required: No

`sourcePath`

The path on the host container instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you.

If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host container instance until you delete it manually. If the `sourcePath` value does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

Type: String

Required: No

Node Properties

`nodeProperties`

When you register a multi-node parallel job definition, you must specify a list of node properties that define the number of nodes to use in your job, the main node index, and the different node ranges to use. The following node properties are allowed in a job definition. For more information, see Multi-node Parallel Jobs (p. 26).

Type: NodeProperties object

Required: No

`mainNode`

Specifies the node index for the main node of a multi-node parallel job.

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 may 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. 38).

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. By default, each job is attempted one time. If you specify more than one attempt, the job is retried if it fails (for example, if it returns a non-zero exit code or the container instance is terminated). For more information, see Automated Job Retries (p. 17).

Type: RetryStrategy object

Required: No

`attempts`

The number of times to move a job to the `RUNNABLE` status. You may 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

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. 18). If a job is terminated due to a timeout, it is not retried. Any timeout configuration that is specified during a SubmitJob operation overrides the timeout configuration defined here. For more information, see Job Timeouts (p. 18).

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

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 is described in the blog post uses these environment variables to download the `myjob.sh` script from S3 and declare its file type.

Although the command and environment variables are hard-coded into the job definition in this example, you can submit a job with this definition and specify command and environment variable overrides to make the job definition more versatile.

```
{
    "jobDefinitionName": "fetch_and_run",
    "type": "container",
    "containerProperties": {
        "image": "012345678910.dkr.ecr.us-east-1.amazonaws.com/fetch_and_run",
        "vcpus": 2,
        "memory": 2000,
        "command": [
            "myjob.sh",
            "60"
        ],
        "jobRoleArn": "arn:aws:iam::012345678910: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 below sets a default for `codec`, but you can override that parameter as needed.

For more information, see Parameters (p. 38).

```
{
    "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::012345678910:role/ECSTask-S3FullAccess",
        "user": "nobody"
    }
}
```

Test GPU Functionality

The following example job definition tests if the GPU workload AMI described in Creating a GPU Workload AMI (p. 55) 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 JSON text above 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
```

Job Queues

Jobs are submitted to a job queue, where they reside until they are able to be scheduled to run in a compute environment. An AWS account can have multiple job queues. For example, you might 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 is 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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, choose **Job queues**, **Create queue**.
4. For **Queue name**, enter a unique name for your job queue.
5. Ensure that **Enable job queue** is selected so that your job queue can accept job submissions.
6. 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.
7. In the **Connected compute environments for this queue** 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 execute 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 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.
8. 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*.

```
{
    "jobQueueName": "",
    "state": "",
    "priority": 0,
    "computeEnvironmentOrder": [{
        "order": 0,
        "computeEnvironment": ""
    }]
}
```

Note
You can generate the above 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 compute environment. 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 is able to accept jobs.

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.

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 execute 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 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.

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.

Compute Environments

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 given 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 is used by the scheduler to determine where to place jobs that are ready to be executed. If the first compute environment has free resources, the job is scheduled to a container instance within that compute environment. If the compute environment is unable to provide a suitable compute resource, the scheduler attempts to run the job on the next compute environment.

Topics

Managed Compute Environments

Managed compute environments enable you to describe your business requirements. In a managed compute environment, AWS Batch manages the capacity and instance types of the compute resources within the environment, based on the compute resource specification that you define when you create the compute environment. You can choose to use Amazon EC2 On-Demand Instances or Spot Instances in your managed compute environment. You can optionally set a maximum price so that Spot Instances only launch when the Spot Instance price is below 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. If your container instances do not have public IP addresses (because the subnets you've chosen do not provide them by default), 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 help creating a VPC, see Tutorial: Creating a VPC with Public and Private Subnets for Your Compute Environments (p. 94).

By default, AWS Batch managed compute environments use a recent, approved version of the Amazon ECS-optimized AMI for compute resources. However, you may want to create your own AMI to use for your managed compute environments for various reasons. For more information, see Compute Resource AMIs (p. 52).

Note
AWS Batch does not upgrade the AMIs in a compute environment after it is created (for example, when a newer version of the Amazon ECS-optimized AMI is available). You are responsible for the management of the guest operating system (including updates and security patches) and 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 old compute environment from your job queue.

4. Delete the old compute environment.

Unmanaged Compute Environments

In an unmanaged compute environment, you manage your own compute resources. You must ensure 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. 52) and Creating a Compute Resource AMI (p. 53).

After you have created your unmanaged compute environment, use the DescribeComputeEnvironments API operation to view the compute environment details. Find the Amazon ECS cluster that is 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:

```
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.

```
#!/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 may want to create your own AMI to use for your managed and unmanaged compute environments for the following reasons:

- 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

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

Required

- A modern Linux distribution 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 (by not specifying a custom AMI). For more information, see Multi-node Parallel Jobs (p. 26).
- The Amazon ECS container agent (preferably 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_DRIVERS` 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 running at least version 1.9, and any Docker runtime dependencies. For more information, see Check runtime dependencies in the Docker documentation.

  Note
  For the best experience, we recommend the Docker version that ships with and is tested with the corresponding Amazon ECS agent version that you are using. For more information, see Amazon ECS Container Agent Versions 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 may use `systemd`. 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*. 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 is not running on your compute resource, then it cannot 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. Choose another AMI if your application requires a specific operating system or a Docker version that is 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. 52). After you have created your custom AMI, you can create a compute environment that uses that AMI, 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 cannot 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 is 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:

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

For example, if you want to run GPU workloads on your AWS Batch compute resources, you could start with the Amazon Linux Deep Learning AMI and configure it to be able to run AWS Batch jobs. For more information, see Creating a GPU Workload AMI (p. 55).

Important
If you choose a base AMI that does not 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, such as:

   - Install 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 are mounted at boot.
   - Configuring Docker options (enable debugging, adjust base image size, and so on).
   - 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 the persistent data checkpoint file before creating your AMI; otherwise, the agent will not start on instances that are launched from your AMI.

   a. Stop the Amazon ECS container agent.

   ```
   sudo stop ecs
   ```

   b. Remove the persistent data checkpoint file. By default, this file is located at `/var/lib/ecs/data/ecs_agent_data.json`. Use the following command to remove the file.

   ```
   sudo rm -rf /var/lib/ecs/data/ecs_agent_data.json
   ```
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. When the AMI creation process is complete, create a compute environment with your new AMI (be sure to select **Enable user-specified AMI ID** and specify your custom AMI ID in Step 5.i (p. 61)). For more information, see Creating a Compute Environment (p. 60).

   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. 47).

   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. 45), Creating a Job Definition (p. 30), and Submitting a Job (p. 13).

Creating a GPU Workload AMI

To run GPU workloads on your AWS Batch compute resources, you can start with the Deep Learning AMI (Amazon Linux) as a base AMI and configure it to be able to run AWS Batch jobs.

This deep learning AMI is based on Amazon Linux, so you can install the `ecs-init` package and make it compatible as a compute resource AMI. The `nvidia-docker2` RPM installs the required components for Docker containers in AWS Batch jobs to be able to access the GPUs on supported instance types.

To configure the Deep Learning AMI for AWS Batch

1. Launch a GPU instance type (for example, P3) with the Deep Learning AMI (Amazon Linux) in a region that AWS Batch supports.
2. Connect to your instance with SSH. For more information, see Connecting to Your Linux Instance Using SSH in the *Amazon EC2 User Guide for Linux Instances*.
3. With your favorite text editor, create a file called `configure-gpu.sh` with the following contents:

```
#!/bin/bash
# Install ecs-init, start docker, and install nvidia-docker 2
sudo yum install -y ecs-init
sudo service docker start
DOCKER_VERSION=$(docker -v | awk '{ print $3 }' | cut -f1 -d"-")
DISTRIBUTION=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$DISTRIBUTION/nvidia-docker.repo | \
  sudo tee /etc/yum.repos.d/nvidia-docker.repo
PACKAGES=$(sudo yum search -y --showduplicates nvidia-docker2 nvidia-container-runtime
 | grep $DOCKER_VERSION | awk '{ print $1 }')
sudo yum install -y $PACKAGES
sudo pkill -SIGHUP dockerd

# Get CUDA version
CUDA_VERSION=$(cat /usr/local/cuda/version.txt | grep "^CUDA Version" | awk '{ print
 $3 }' | cut -f1-2 -d".")

# Run test container to verify installation
```

```
sudo docker run --privileged --runtime=nvidia --rm nvidia/cuda:$CUDA_VERSION-base
 nvidia-smi

# Update Docker daemon.json to user nvidia-container-runtime by default
sudo tee /etc/docker/daemon.json <<EOF
{
    "runtimes": {
        "nvidia": {
            "path": "/usr/bin/nvidia-container-runtime",
            "runtimeArgs": []
        }
    },
    "default-runtime": "nvidia"
}
EOF

sudo service docker restart
```

4. Run the script.

```
bash ./configure-gpu.sh
```

5. Get the CUDA version that is installed on your instance.

```
CUDA_VERSION=$(cat /usr/local/cuda/version.txt | grep "^CUDA Version" | awk '{ print
 $3 }' | cut -f1-2 -d".")
```

6. Validate that you can run a Docker container and access the installed drivers with the following command.

```
sudo docker run --privileged --runtime=nvidia --rm nvidia/cuda:$CUDA_VERSION-base
 nvidia-smi
```

You should see something similar to the following output.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   38C    P0    25W / 300W |      0MiB / 16130MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

7. Remove any Docker containers and images on the instance to reduce the size of your AMI.

   a. Remove containers.

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

   b. Remove images.

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

8. If you started the Amazon ECS container agent on your instance, you must stop it and remove the persistent data checkpoint file before creating your AMI; otherwise, the agent will not start on instances that are launched from your AMI.

   a. Stop the Amazon ECS container agent.

   ```
   sudo stop ecs
   ```

   b. Remove the persistent data checkpoint file. By default, this file is located at `/var/lib/ecs/data/ecs_agent_data.json`. Use the following command to remove the file.

   ```
   sudo rm -rf /var/lib/ecs/data/ecs_agent_data.json
   ```

9. 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. When the AMI creation process is complete, create a compute environment with your new AMI (be sure to select **Enable user-specified AMI ID** and specify your custom AMI ID in Step 5.i (p. 61)). For more information, see Creating a Compute Environment (p. 60).
2. Create a job queue and associate your new compute environment. For more information, see Creating a Job Queue (p. 47).
3. (Optional) Submit a sample job to your new job queue to test the GPU functionality. For more information, see Test GPU Functionality (p. 46), Creating a Job Definition (p. 30), and Submitting a Job (p. 13).

Launch Template Support

AWS Batch supports using Amazon EC2 launch templates with your compute environments. Launch template support allows you to modify the default configuration of your AWS Batch compute resources without requiring you to create customized AMIs.

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 JSON file below 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.

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

You can create the above 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 does not 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 is executed 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, because your user data is merged with other AWS Batch user data that is 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.

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

- The content type and part boundary declaration: `Content-Type: multipart/mixed; boundary="==BOUNDARY=="`
- The MIME version declaration: `MIME-Version: 1.0`
- One or more user data blocks, which contain the following components:
  - The opening boundary, which signals the beginning of a user data block: `--==BOUNDARY==`
  - The content type declaration for the block: `Content-Type: text/cloud-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, which signals the end of the MIME multi-part file: `--==BOUNDARY==--`

Below are some 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 plain text, 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 the JSON below.

```
{
    "LaunchTemplateName": "base64-user-data",
    "LaunchTemplateData": {
        "UserData":
 "ewogICAgIkxhdW5jaFRlbXBsYXRlTmFtZSI6ICJpbmNyZWFzZS1jb250YWluZXItdm9sdW..."
    }
}
```

Example Mount an existing Amazon EFS file system

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==--
```

Example Override default Amazon ECS container agent configuration

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==--
```

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 instances within the environment based on your specifications, or you can create an unmanaged compute environment where you handle the instance configuration within the environment.

To create a managed compute environment

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, choose **Compute environments**, **Create environment**.
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 letters (uppercase and lowercase), numbers, hyphens, and underscores.
    c. 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 APIs on your behalf. For more information, see AWS Batch Service IAM Role (p. 77). If you choose to create a new role, the required role (`AWSBatchServiceRole`) is created for you.
    d. 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 APIs on your behalf. For more information, see Amazon ECS Instance Role (p. 79). If you choose to create a new instance profile, the required role (`ecsInstanceRole`) is created for you.
    e. 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 ingress on port 22).
    f. Ensure that **Enable compute environment** is selected so that your compute environment can accept jobs from the AWS Batch job scheduler.
5. Configure your compute resources.
    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:
        i. (Optional) For **Maximum 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 below 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.

ii. For **Spot fleet role**, choose an existing Amazon EC2 Spot Fleet IAM role to apply to your Spot compute environment. If you do not already have an existing Amazon EC2 Spot Fleet IAM role, you must create one first. For more information, see Amazon EC2 Spot Fleet Role (p. 80).

Important
To tag your Spot Instances on creation (see Step 7 (p. 62)), your Amazon EC2 Spot Fleet IAM role must use the newer **AmazonEC2SpotFleetTaggingRole** managed policy. The **AmazonEC2SpotFleetRole** managed policy does not have the required permissions to tag Spot Instances. For more information, see Spot Instances Not Tagged on Creation (p. 100).

c. For **Allowed instance types**, choose the Amazon EC2 instance types that may be launched. You can specify instance families to launch any instance type within those families (for example, `c4` or `p3`), or you can specify specific sizes within a family (such as `c4.8xlarge`). You can also choose `optimal` to pick instance types (from the latest C, M, and R 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.

d. (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. 57).

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

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

g. 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.

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

i. (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. 52).

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**.

6. Configure networking.

Important
Compute resources need external network access to communicate with the Amazon ECS service endpoint, so if 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. 94).

   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. For **Security groups**, choose a security group to attach to your instances. By default, the default security group for your VPC is chosen.
7. (Optional) Tag your 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.
8. Choose **Create** to finish.

To create an unmanaged compute environment

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 APIs on your behalf. For more information, see AWS Batch Service IAM Role (p. 77). 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:

```
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.

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

Note
Your unmanaged compute environment does not have any compute resources until you launch them manually.

Compute Environment Template

An empty compute environment template is shown below. 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*.

```
{
    "computeEnvironmentName": "",
    "type": "UNMANAGED",
    "state": "ENABLED",
    "computeResources": {
        "type": "EC2",
        "minvCpus": 0,
        "maxvCpus": 0,
        "desiredvCpus": 0,
        "instanceTypes": [
            ""
        ],
        "imageId": "",
        "subnets": [
            ""
        ],
        "securityGroupIds": [
            ""
        ],
        "ec2KeyPair": "",
        "instanceRole": "",
        "tags": {
            "KeyName": ""
        },
        "bidPercentage": 0,
        "spotIamFleetRole": "",
        "launchTemplate": {
            "launchTemplateId": "",
            "launchTemplateName": "",
            "version": ""
        }
    },
    "serviceRole": ""
}
```

Note
You can generate the above 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 is a managed compute environment), and the service role to use to provide IAM permissions to AWS Batch.

Topics

Compute Environment Name

`computeEnvironmentName`

The name for your compute environment. You can use up to 128 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 compute resources that you define). For more information, see Compute Resources (p. 64). Choose `UNMANAGED` to manage your own compute resources.

Type: String

Valid values: `MANAGED` | `UNMANAGED`

Required: Yes

State

`state`

The state of the compute environment.

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

If the state is `DISABLED`, then the AWS Batch scheduler does not 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 do not scale out; however, they scale in to `minvCpus` value once instances become idle.

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Compute Resources

`computeResources`

Details of the compute resources managed by the compute environment.

Type: ComputeResource object

Required: this parameter is required for managed compute environments

`type`

The type of compute environment. Use this parameter to specify whether to use Amazon EC2 On-Demand Instances or Amazon EC2 Spot Instances in your 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. 80).

Valid values: `EC2` | `SPOT`

Required: Yes

`minvCpus`

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

Type: Integer

Required: Yes

`maxvCpus`

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

Type: Integer

Required: Yes

`desiredvCpus`

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

Type: Integer

Required: No

`instanceTypes`

The instance types that may be launched. You can specify instance families to launch any instance type within those families (for example, `c4` or `p3`), or you can specify specific sizes within a family (such as `c4.8xlarge`). You can also choose `optimal` to pick instance types (from the latest C, M, and R 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.

Type: Array of strings

Required: yes

`imageId`

The Amazon Machine Image (AMI) ID used for instances launched in the compute environment.

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.

Type: Array of strings

Required: Yes

`securityGroupIds`

The EC2 security groups to associate with the instances launched in the compute environment.

Type: Array of strings

Required: Yes

`ec2KeyPair`

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

Type: String

Required: No

`instanceRole`

The Amazon ECS instance profile to attach to Amazon EC2 instances in a compute environment. 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. 79).

Type: String

Required: Yes

`tags`

Key-value pair tags to be applied to 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.

Type: String to string map

Required: No

`bidPercentage`

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 percentage is 20%, then the Spot price must be below 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.

Required: No

`spotIamFleetRole`

The Amazon Resource Name (ARN) of the Amazon EC2 Spot Fleet IAM role applied to a `SPOT` compute environment. For more information, see Amazon EC2 Spot Fleet Role (p. 80).

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 does not have the required permissions to tag Spot Instances. For more information, see Spot Instances Not Tagged on Creation (p. 100).

Type: String

Required: This parameter is required for `SPOT` compute environments.

`launchTemplate`

An optional launch template to associate with your compute resources. 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. 57).

Type: LaunchTemplateSpecification

object

Required: No

`launchTemplateId`

The ID of the launch template.

Type: String

Required: No

`launchTemplateName`

The name of the launch template.

Type: String

Required: No

`version`

The version number of the launch template.

Type: String

Required: No

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. 77).

Type: String

Required: Yes

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. 68).

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.

```
             total       used       free     shared    buffers     cached
Mem:    8373026816  348180480 8024846336      90112   25534464  205418496
-/+ buffers/cache:  117227520 8255799296
```

This instance has 8373026816 bytes of total memory, which translates to 7985 MiB available for tasks.

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.

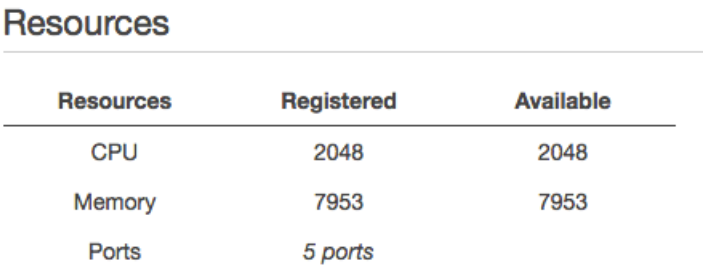
The default AWS Batch compute resource AMI reserves 32 MiB of memory for the Amazon ECS container agent and other critical system processes.

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.



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 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 actions 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 the *IAM 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. 79) and also IAM Roles 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

The following topics explain the structure of an IAM policy.

Topics

Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```
{
  "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. 71).
- **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. 73) and Amazon Resource Names for AWS Batch (p. 72). 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. 77).

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-1`).

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. 73).
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 table 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 is not listed in this table, 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.

API action	Resource	Condition keys
RegisterJobDefinition	Job Definition arn:aws:batch:*region*:*account*:job-definition/* arn:aws:batch:*region*:*account*:job-definition/definition-name:revision	batch:User batch:Privileged batch:Image
DeregisterJobDefinition	Job Definition arn:aws:batch:*region*:*account*:job-definition/* arn:aws:batch:*region*:*account*:job-definition/definition-name:revision	N/A
SubmitJob	Job Definition arn:aws:batch:*region*:*account*:job-definition/* arn:aws:batch:*region*:*account*:job-definition/definition-name:revision Job Queue arn:aws:batch:*region*:*account*:job-queue/* arn:aws:batch:*region*:*account*:job-queue/queue-name	N/A

Example Policies

The following examples show policy statements that you could use to control the permissions that IAM users have to AWS Batch.

Examples

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.

```
{
    "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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "batch:RegisterJobDefinition"
            ],
            "Resource": [
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/JobDefA_*"
            ],
            "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": [
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/JobDefA_*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::<aws_account_id>:role/MyBatchJobRole"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "batch.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

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.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob"
            ],
            "Resource": [
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/JobDefA_*",
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/*"
            ]
        }
    ]
}
```

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.

```
{
    "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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:*",
                "cloudwatch:GetMetricStatistics",
                "ec2:DescribeSubnets",
```

```
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeKeyPairs",
                "ecs:DescribeClusters",
                "ecs:Describe*",
                "ecs:List*",
                "logs:Describe*",
                "logs:Get*",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents",
                "iam:ListInstanceProfiles",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["iam:PassRole"],
            "Resource": [
                "arn:aws:iam::*:role/AWSBatchServiceRole",
                "arn:aws:iam::*:role/ecsInstanceRole",
                "arn:aws:iam::*:role/iaws-ec2-spot-fleet-role",
                "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
                "arn:aws:iam::*:role/AWSBatchJobRole*"
            ]
        }
    ]
}
```

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.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeAccountAttributes",
            "ec2:DescribeInstances",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
```

```
            "ec2:DescribeKeyPairs",
            "ec2:DescribeImages",
            "ec2:DescribeImageAttribute",
            "ec2:DescribeSpotFleetInstances",
            "ec2:DescribeSpotFleetRequests",
            "ec2:DescribeSpotPriceHistory",
            "ec2:RequestSpotFleet",
            "ec2:CancelSpotFleetRequests",
            "ec2:ModifySpotFleetRequest",
            "ec2:TerminateInstances",
            "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:DescribeTaskDefinitions",
            "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:PassRole"
        ],
        "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**.

   b. To narrow the list of available policies to attach, for **Filter**, type **AWSBatchServiceRole**.

   c. Select the **AWSBatchServiceRole** policy and choose **Attach Policy**.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {"Service": "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**.
4. Choose **Next: Permissions**, **Next: Review**.
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 New Role**.

b. For **Select type of trusted entity**, choose **AWS service**. For **Choose the service that will use this role**, choose **Elastic Container Service**.
c. Choose **Next: Review**, **Next: Permissions**.
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 bid on, launch, tag, and terminate instances on your behalf. Specify the role in your Spot Fleet request. You must also have the **AWSServiceRoleForEC2Spot** and **AWSServiceRoleForEC2SpotFleet** service-linked roles for Amazon EC2 Spot and Spot Fleet. Use the procedures below to create all of these roles.

Topics

Create Amazon EC2 Spot Fleet Roles in the AWS Management Console

To create the `AmazonEC2SpotFleetRole` IAM role for your Spot Fleet compute environments

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**. For **Choose the service that will use this role**, choose **EC2**.
4. In the **Select your use case** section, choose **EC2 Spot Fleet Role** and choose **Next: Permissions**.
5. Choose **Next: Review**.

   Note
   Historically, there have been two managed policies for the Amazon EC2 Spot Fleet role.

   - **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. 100) to apply the new recommended policy to that role.
   - **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.
6. For **Role Name**, type `AmazonEC2SpotFleetRole`. Choose **Create Role**.

To create the `AWSServiceRoleForEC2Spot` IAM service-linked role for Amazon EC2 Spot

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**. For **Choose the service that will use this role**, choose **EC2**.

4. In the **Select your use case** section, choose **EC2 - Spot Instances** and then choose **Next: Permissions**.
5. Choose **Next: Review**, **Create role**.

To create the `AWSServiceRoleForEC2SpotFleet` IAM service-linked role for Amazon EC2 Spot Fleet

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**. For **Choose the service that will use this role**, choose **EC2**.
4. In the **Select your use case** section, choose **EC2 - Spot Fleet** and then choose **Next: Permissions**.
5. Choose **Next: Review**, **Create role**.

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:

```
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:

```
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:

```
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:

```
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. 84).

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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

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.

```
{
       "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. 84).

Topics

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 programmatically deserializing event JSON data, make sure that your application is prepared to handle unknown properties to avoid issues if and when these additional properties are added.

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. 15).

> **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 Job 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*.

```
{
  "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",
    "jobQueue": "arn:aws:batch:us-east-1:aws_account_id:job-queue/HighPriority",
    "status": "RUNNABLE",
    "attempts": [],
    "createdAt": 1508781340401,
    "retryStrategy": {
      "attempts": 1
    },
    "dependsOn": [],
    "jobDefinition": "arn:aws:batch:us-east-1:aws_account_id:job-definition/first-run-job-
definition:1",
    "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. 82).

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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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. 18).
   - **Job attempts:** (Optional) Enter the number of times to retry your job if it fails. For more information, see Automated Job Retries (p. 17).
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. 82).
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*.

```
{
  "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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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. For **Configure input**, 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.

   ```
   {"S3BucketValue":"$.detail.requestParameters.bucketName","S3KeyValue":"$.detail.requestParameters.k
   ```

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.

   ```
   {"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. 82).
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 **Name**, enter **batch-event-stream-handler**.
4. For **Role**, choose **Create a custom role**, **Allow**.
5. Choose **Create function**.
6. In the **Function code** section, choose **Python 2.7** for the runtime and edit the sample code to match the following example:

```
import json

def lambda_handler(event, 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 2.7 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 **Save**.

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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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.

```
{
  "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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. Submit a new AWS Batch job. For more information, see Submitting a Job (p. 13).
3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v2/home.
2. Choose **Topics**, **Create new 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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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:

```
{
  "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

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
2. Submit a new AWS Batch job. For more information, see Submitting a Job (p. 13). 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.

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.

Topics

Step 1: Create an Elastic IP Address for Your NAT Gateway

A NAT gateway requires an Elastic IP address in your public subnet, but the VPC wizard does not create one for you. Create the Elastic IP address before running the VPC wizard.

To create an Elastic IP address

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose **Elastic IPs**.
3. Choose **Allocate new address**, **Allocate**, **Close**.
4. Note the **Allocation ID** for your newly created Elastic IP address; you enter this later in the VPC wizard.

Step 2: Run the VPC Wizard

The VPC wizard automatically creates and configures most of your VPC resources for you.

To run the VPC wizard

1. In the left navigation pane, choose **VPC Dashboard**.
2. Choose **Start VPC Wizard**, **VPC with Public and Private Subnets**, **Select**.
3. For **VPC name**, give your VPC a unique name.
4. For **Elastic IP Allocation ID**, choose the ID of the Elastic IP address that you created earlier.

5. Choose **Create VPC**.
6. When the wizard is finished, choose **OK**. Note the Availability Zone in which your VPC subnets were created. Your additional subnets should be created in a different Availability Zone.

Step 3: Create Additional Subnets

The wizard creates a VPC with a single public and a single private subnet in a single Availability Zone. For greater availability, you should create at least one more of each subnet type in a different Availability Zone so that your VPC has both public and private subnets across two Availability Zones.

To create an additional private subnet

1. In the left navigation pane, choose **Subnets**.
2. Choose **Create Subnet**.
3. For **Name tag**, enter a name for your subnet, such as **Private subnet**.
4. For **VPC**, choose the VPC that you created earlier.
5. For **Availability Zone**, choose a different Availability Zone than your original subnets in the VPC.
6. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.3.0/24** for your second private subnet.
7. Choose **Yes, Create**.

To create an additional public subnet

1. In the left navigation pane, choose **Subnets** and then **Create Subnet**.
2. For **Name tag**, enter a name for your subnet, such as **Public subnet**.
3. For **VPC**, choose the VPC that you created earlier.
4. For **Availability Zone**, choose the same Availability Zone as the additional private subnet that you created in the previous procedure.
5. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.2.0/24** for your second public subnet.
6. Choose **Yes, Create**.
7. Select the public subnet that you just created and choose **Route Table**, **Edit**.
8. By default, the private route table is selected. Choose the other available route table so that the **0.0.0.0/0** destination is routed to the internet gateway (**igw-*xxxxxxxx***) and choose **Save**.
9. With your second public subnet still selected, choose **Subnet Actions**, **Modify auto-assign IP settings**.
10. Select **Enable auto-assign public IPv4 address** and choose **Save**, **Close**.

Next Steps

After you have created your VPC, you should consider the following next steps:

- Create security groups for your public and private resources if they require inbound network access. For more information, see Working with Security Groups in the *Amazon VPC User Guide*.
- Create an AWS Batch managed compute environment that launches compute resources into your new VPC. For more information, see Creating a Compute Environment (p. 60). 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. 47).
- Create a job definition to run your jobs with. For more information, see Creating a Job Definition (p. 30).
- 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. 13).

AWS Batch Service Limits

The following table provides the service limits for AWS Batch, which cannot be changed.

Resource	Limit
Maximum number of job queues	20
Maximum number of compute environments	18
Maximum number of compute environments per job queue	3
Maximum number of job dependencies	20
Maximum job definition size (for `RegisterJobDefinition` API operations)	24 KiB
Maximum job payload size (for `SubmitJob` API operations)	30 KiB
Maximum array size for array jobs	10,000
Maximum number of jobs in `SUBMITTED` state	1,000,000

Troubleshooting AWS Batch

You may 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 may 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. 77), 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 is misconfigured this way, see Repairing an INVALID Compute Environment (p. 99).

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. 98), you can update the compute environment with the correct service role.

To repair a misconfigured compute environment

1. Open the AWS Batch console at https://console.aws.amazon.com/batch/.
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 do not 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 is not 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. 52) and Creating a Compute Resource AMI (p. 53).

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, 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.

No internet access for compute resources

Compute resources need external network access to communicate with the Amazon ECS service endpoint, so if 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. 94).

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*

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 procedure below 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.

update-history-change	update-history-description	update-history-date
Multi-node Parallel Jobs	Multi-node parallel jobs enable you to run single jobs that span multiple Amazon EC2 instances.	November 19, 2018
Resource-level Permissions	AWS Batch now supports resource-level permissions on several API operations.	November 12, 2018
Amazon EC2 Launch Template Support	AWS Batch adds support for using launch templates with compute environments.	November 12, 2018
AWS Batch 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.	April 5, 2018
AWS Batch Jobs as CloudWatch Events Targets	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.	March 1, 2018
CloudTrail Auditing for AWS Batch	CloudTrail can audit calls made to AWS Batch API actions.	January 10, 2018
Array Jobs	AWS Batch supports array jobs, which are useful for parameter sweep and Monte Carlo workloads.	November 28, 2017
Expanded AWS Batch Tagging	AWS Batch enables you to specify tags for the Amazon EC2 Spot Instances launched within managed compute environments.	October 26, 2017
AWS Batch Event Stream for CloudWatch Events	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.	October 24, 2017
Automated Job Retries	You can apply a retry strategy to your jobs and job definitions that allows your jobs to be automatically retried if they fail.	March 28, 2017

AWS Batch General Availability (p. 101)	AWS Batch enables you to run batch computing workloads on the AWS Cloud.	January 5, 2017

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.