AWS ParallelCluster

AWS ParallelCluster User Guide

Table of Contents

What Is AWS ParallelCluster

AWS ParallelCluster is an AWS-supported open source cluster management tool that helps you to deploy and manage High Performance Computing (HPC) clusters in the AWS Cloud. Built on the open source CfnCluster project, AWS ParallelCluster enables you to quickly build an HPC compute environment in AWS. It automatically sets up the required compute resources and shared filesystem. You can use AWS ParallelCluster with a variety of batch schedulers, such as AWS Batch, SGE, Torque, and Slurm. AWS ParallelCluster facilitates quick start proof of concept deployments and production deployments. You can also build higher level workflows, such as a genomics portal that automates an entire DNA sequencing workflow, on top of AWS ParallelCluster.

Setting Up AWS ParallelCluster

Topics

Installing AWS ParallelCluster

AWS ParallelCluster is distributed as a Python package and is installed using `pip`, the Python package manager. For more information on installing Python packages, see Installing Packages in the *Python Packaging User Guide*.

Ways to install AWS ParallelCluster:

You can find the version number of the most recent CLI at: https://github.com/aws/aws-parallelcluster/blob/release/CHANGELOG.rst.

In this guide, the command examples assume that you have Python v3 installed. The `pip` command examples use the `pip3` version.

Installing AWS ParallelCluster in a Virtual Environment (Recommended)

We recommend that you install AWS ParallelCluster in a virtual environment. If you encounter issues when you attempt to install AWS ParallelCluster with `pip3`, you can install AWS ParallelCluster in a virtual environment (p. 3) to isolate the tool and its dependencies. Or you can use a different version of Python than you normally do.

Installing AWS ParallelCluster in a non-Virtual Environment Using pip

The primary distribution method for AWS ParallelCluster on Linux, Windows, and macOS is `pip`, which is a package manager for Python. It provides a way to install, upgrade, and remove Python packages and their dependencies.

Current AWS ParallelCluster Version

AWS ParallelCluster is updated regularly. To determine whether you have the latest version, see the releases page on GitHub.

If you already have `pip` and a supported version of Python, you can install AWS ParallelCluster by using the following command. If you have Python version 3+ installed, we recommend that you use the **`pip3`** command.

```
$ pip3 install aws-parallelcluster --upgrade --user
```

Steps to Take after Installation

After you install AWS ParallelCluster, you might need to add the executable file path to your `PATH` variable. For platform-specific instructions, see the following topics:

- **Linux** – Add the AWS ParallelCluster Executable to Your Command Line Path (p. 7)
- **macOS** – Add the AWS ParallelCluster Executable to Your Command Line Path (p. 9)
- **Windows** – Add the AWS ParallelCluster Executable to Your Command Line Path (p. 11)

You can verify that AWS ParallelCluster installed correctly by running `pcluster version`.

```
$ pcluster version
2.4.1
```

AWS ParallelCluster is updated regularly. To update to the latest version of AWS ParallelCluster, run the installation command again. For details about the latest version of AWS ParallelCluster, see the AWS ParallelCluster release notes.

```
$ pip3 install aws-parallelcluster --upgrade --user
```

To uninstall AWS ParallelCluster, use `pip uninstall`.

```
$ pip3 uninstall aws-parallelcluster
```

If you don't have Python and `pip`, use the procedure for your environment.

Detailed Instructions for Each Environment

- Install AWS ParallelCluster in a Virtual Environment (p. 3)
- Install AWS ParallelCluster on Linux (p. 5)
- Install AWS ParallelCluster on macOS (p. 8)
- Install AWS ParallelCluster on Windows (p. 10)

Install AWS ParallelCluster in a Virtual Environment

We recommend that you install AWS ParallelCluster in a virtual environment, to avoid requirement version conflicts with other `pip3` packages.

Prerequisites

- Verify that `pip` and Python are installed. We recommend `pip3`, and Python 3 version 3.6. If you are using Python 2, use `pip` instead of `pip3` and `virtualenv` instead of `venv`.

•

To install AWS ParallelCluster in a virtual environment

1. Install `virtualenv` using `pip3`.

   Linux, macOS, or Unix

   ```
   $ python3 -m pip install --upgrade pip
   $ pip3 install --user --upgrade virtualenv
   ```

   Windows

   ```
   C:\>pip3 install --user --upgrade virtualenv
   ```

2. Create a virtual environment and name it.

   Linux, macOS, or Unix

   ```
   $ virtualenv ~/apc-ve
   ```

   Windows

   ```
   C:\>virtualenv %USERPROFILE%\apc-ve
   ```

   Alternatively, you can use the `-p` option to specify a version of Python other than the default.

   ```
   $ virtualenv -p /usr/bin/python3.6 ~/apc-ve
   ```

3. Activate your new virtual environment.

   Linux, macOS, or Unix

   ```
   $ source ~/apc-ve/bin/activate
   ```

   Windows

   ```
   C:\>%USERPROFILE%\apc-ve\Scripts\activate
   ```

4. Install AWS ParallelCluster into your virtual environment.

   Linux, macOS, or Unix

   ```
   (apc-ve)~$ pip install --upgrade aws-parallelcluster
   ```

   Windows

   ```
   (apc-ve) C:\>pip install --upgrade aws-parallelcluster
   ```

5. Verify that AWS ParallelCluster is installed correctly.

   Linux, macOS, or Unix

   ```
   $ pcluster version
   ```

```
2.4.1
```

Windows

```
(apc-ve) C:\>pcluster version
2.4.1
```

You can use the `deactivate` command to exit the virtual environment. Each time that you start a session, you must reactivate the environment (p. 4).

To upgrade to the latest version of AWS ParallelCluster, run the installation command again.

Linux, macOS, or Unix

```
(apc-ve)~$ pip3 install --upgrade aws-parallelcluster
```

Windows

```
(apc-ve) C:\>pip3 install --upgrade aws-parallelcluster
```

Install AWS ParallelCluster on Linux

You can install AWS ParallelCluster and its dependencies on most Linux distributions by using `pip`, a package manager for Python. First, determine if Python and `pip` are installed:

1. To determine if your version of Linux includes Python and `pip`, run `pip --version`.

   ```
   $ pip --version
   ```

   If you have `pip` installed, go on to the Install AWS ParallelCluster with pip (p. 2) topic. Otherwise, continue with Step 2.
2. To determine if Python is installed, run `python --version`.

   ```
   $ python --version
   ```

   If you have Python 3 version 3.6+ or Python 2 version 2.7 installed, go on to the Install AWS ParallelCluster with pip (p. 2) topic. Otherwise, install Python (p. 7), and then return to this procedure to install `pip`.
3. Install `pip` by using the script that the *Python Packaging Authority* provides.
4. Use the `curl` command to download the installation script.

   ```
   $ curl -O https://bootstrap.pypa.io/get-pip.py
   ```
5. Run the script with Python to download and install the latest version of `pip` and other required support packages.

   ```
   $ python get-pip.py --user
   ```

   or

   ```
   $ python3 get-pip.py --user
   ```

When you include the `--user` switch, the script installs `pip` to the path `~/.local/bin`.

6. To ensure that the folder that contains `pip` is part of your `PATH` variable, do the following:

   a. Find your shell's profile script in your user folder. If you're not sure which shell you have, run `basename $SHELL`.

   ```
   $ ls -a ~
   .  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
   ```

   - **Bash** – `.bash_profile`, `.profile`, or `.bash_login`
   - **Zsh** – `.zshrc`
   - **Tcsh** – `.tcshrc`, `.cshrc` or `.login`

   b. Add an export command at the end of your profile script that's similar to the following example.

   ```
   export PATH=~/.local/bin:$PATH
   ```

   The export command inserts the path, which is `~/.local/bin` in this example, at the front of the existing `PATH` variable.

   c. To put these changes into effect, reload the profile into your current session.

   ```
   $ source ~/.bash_profile
   ```

7. Verify that `pip` is installed correctly.

   ```
   $ pip3 --version
   pip 19.2.3 from ~/.local/lib/python3.6/site-packages (python 3.6)
   ```

Sections

Install AWS ParallelCluster with `pip`

Use `pip` to install AWS ParallelCluster.

```
$ pip3 install aws-parallelcluster --upgrade --user
```

When you use the `--user` switch, `pip` installs AWS ParallelCluster to `~/.local/bin`.

Verify that AWS ParallelCluster installed correctly.

```
$ pcluster version
2.4.1
```

To upgrade to the latest version, run the installation command again.

```
$ pip3 install aws-parallelcluster --upgrade --user
```

Add the AWS ParallelCluster Executable to Your Command Line Path

After installing with `pip`, you might need to add the `pcluster` executable to your operating system's `PATH` environment variable.

To verify the folder in which `pip` installed AWS ParallelCluster, run the following command.

```
$ which pcluster
/home/username/.local/bin/pcluster
```

If you omitted the `--user` switch when you installed AWS ParallelCluster, the executable might be in the `bin` folder of your Python installation. If you don't know where Python is installed, run this command.

```
$ which python
/usr/local/bin/python
```

Note that the output might be the path to a symlink, not to the actual executable. To see where the symlink points, run `ls -al`.

```
$ ls -al $(which python)
/usr/local/bin/python -> ~/.local/Python/3.6/bin/python3.6
```

If this is the same folder that you added to the path in step 3 in Installing AWS ParallelCluster (p. 2), you're done with the installation. Otherwise, you must perform steps 3a – 3c again, adding this additional folder to the path.

Installing Python on Linux

If your distribution didn't come with Python, or came with an earlier version, install Python before installing `pip` and AWS ParallelCluster.

To install Python 3 on Linux

1. Check to see if Python is already installed.

   ```
   $ python3 --version
   ```

   or

   ```
   $ python --version
   ```

   Note
   If your Linux distribution came with Python, you might need to install the Python developer package. The developer package includes the headers and libraries that are required to compile extensions and to install AWS ParallelCluster. Use your package manager to install the developer package. It is typically named `python-dev` or `python-devel`.

2. If Python 2.7 or later is not installed, install Python with your distribution's package manager. The command and package name varies:

   - On Debian derivatives such as Ubuntu, use `apt`.

     ```
     $ sudo apt-get install python3
     ```

- On Red Hat and derivatives, use `yum`.

```
$ sudo yum install python3
```

- On SUSE and derivatives, use `zypper`.

```
$ sudo zypper install python3
```

3. To verify that Python installed correctly, open a command prompt or shell and run the following command.

```
$ python3 --version
Python 3.7.3
```

Install AWS ParallelCluster on macOS

Sections

Prerequisites

- Python 3 version 3.6+ or Python 2 version 2.7

Check your Python installation.

```
$ python --version
```

If your computer doesn't already have Python installed, or if you want to install a different version of Python, follow the procedure in Install AWS ParallelCluster on Linux (p. 5).

Install AWS ParallelCluster on macOS Using pip

You can also use `pip` directly to install AWS ParallelCluster. If you don't have `pip`, follow the instructions in the main installation topic (p. 2). Run `pip3 --version` to see if your version of macOS already includes Python and `pip3`.

```
$ pip3 --version
```

To install AWS ParallelCluster on macOS

1. Download and install the latest version of Python from the downloads page of Python.org.
2. Download and run the `pip3` installation script provided by the Python Packaging Authority.

```
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py --user
```

3. Use your newly installed `pip3` to install AWS ParallelCluster. We recommend that if you use Python version 3+, you use the `pip3` command.

```
$ pip3 install aws-parallelcluster --upgrade --user
```

4. Verify that AWS ParallelCluster is installed correctly.

```
$ pcluster version
2.4.1
```

If the program isn't found, add it to your command line path (p. 9).

To upgrade to the latest version, run the installation command again.

```
$ pip3 install aws-parallelcluster --upgrade --user
```

Add the AWS ParallelCluster Executable to Your Command Line Path

After installing with `pip`, you might need to add the `pcluster` program to your operating system's `PATH` environment variable. The location of the program depends on where Python is installed.

Example AWS ParallelCluster install location - macOS with Python 3.6 and `pip` (user mode)

```
~/Library/Python/3.6/bin
```

Substitute the version of Python that you have for the version in the preceding example.

If you don't know where Python is installed, run `which python`.

```
$ which python3
/usr/local/bin/python3
```

The output might be the path to a symlink, not the path to the actual program. Run `ls -al` to see where it points.

```
$ ls -al /usr/local/bin/python3
lrwxr-xr-x  1 username  admin  36 Mar 12 12:47 /usr/local/bin/python3 -> ../Cellar/
python/3.6.8/bin/python3
```

`pip` installs programs in the same folder that contains the Python application. Add this folder to your `PATH` variable.

To modify your `PATH` variable (Linux, macOS, or Unix)

1. Find your shell's profile script in your user folder. If you're not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

   - **Bash** – `.bash_profile`, `.profile`, or `.bash_login`
   - **Zsh** – `.zshrc`
   - **Tcsh** – `.tcshrc`, `.cshrc`, or `.login`
2. Add an export command to your profile script.

```
export PATH=~/.local/bin:$PATH
```

This command adds a path, `~/.local/bin` in this example, to the current `PATH` variable.

3. Load the profile into your current session.

```
$ source ~/.bash_profile
```

Install AWS ParallelCluster on Windows

You can install AWS ParallelCluster on Windows by using `pip`, which is a package manager for Python. If you already have `pip`, follow the instructions in the main installation topic (p. 2).

Sections

Install AWS ParallelCluster Using Python and `pip` on Windows

The Python Software Foundation provides installers for Windows that include `pip`.

To install Python and `pip` (Windows)

1. Download the Python Windows x86-64 installer from the downloads page of Python.org.
2. Run the installer.
3. Choose **Add Python 3 to PATH**.
4. Choose **Install Now**.

The installer installs Python in your user folder and adds its program folders to your user path.

To install AWS ParallelCluster with `pip3` (Windows)

If you use Python version 3+, we recommend that you use the `pip3` command.

1. Open the **Command Prompt** from the **Start** menu.
2. Use the following commands to verify that Python and `pip` are both installed correctly.

```
C:\>python --version
Python 3.6.8
C:\>pip3 --version
pip 19.0.3 from C:\Users\username\AppData\Roaming\Python\Python36\site-packages\pip
 (python 3.6)
```

3. Install AWS ParallelCluster using `pip`.

```
C:\>pip3 install aws-parallelcluster
```

4. Verify that AWS ParallelCluster is installed correctly.

```
C:\>pcluster version
2.4.1
```

To upgrade to the latest version, run the installation command again.

```
C:\>pip3 install --user --upgrade aws-parallelcluster
```

Add the AWS ParallelCluster Executable to Your Command Line Path

After installing AWS ParallelCluster with `pip`, add the `pcluster` program to your operating system's `PATH` environment variable.

You can find where the aws program is installed by running the following command.

```
C:\>where pcluster
C:\Users\username\AppData\Roaming\Python\Python36\Scripts\pcluster.exe
```

If that command does not return any results, then you must add the path manually. Use the command line or Windows Explorer to discover where it is installed on your computer. Typical paths include:

- **Python 3 and `pip3`** – `C:\Program Files\Python36\Scripts\`
- **Python 3 and `pip3` --user option** – `%APPDATA%\Python\Python36\Scripts`

> **Note**
> Folder names that include version numbers can vary. The preceding examples show Python36. Replace as needed with the version number that you are using.

To modify your PATH variable (Windows)

1. Press the Windows key and enter **`environment variables`**.
2. Choose **Edit environment variables for your account**.
3. Choose **PATH**, and then choose **Edit**.
4. Add the path to the **Variable value** field. For example: ***`C:\new\path`***
5. Choose **OK** twice to apply the new settings.
6. Close any running command prompts and reopen the command prompt window.

Configuring AWS ParallelCluster

After you install AWS ParallelCluster, complete the following configuration steps.

First, set up your AWS credentials. For more information, see Configuring the AWS CLI in the *AWS CLI user guide*.

```
$ aws configure
  AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [us-east-1]: us-east-1
Default output format [None]:
```

The Region where the cluster will be launched must have at least one Amazon EC2 key pair. For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*.

```
$ pcluster configure
```

The configure wizard prompts you for all of the information that's needed to create your cluster. The details of the sequence differ when using AWS Batch as the scheduler compared to using SGE, Slurm, or Torque.

SGE, Slurm, or Torque

From the list of valid AWS Region identifiers, choose the Region in which you want your cluster to run.

```
Allowed values for the AWS Region ID:
1. ap-northeast-1
2. ap-northeast-2
3. ap-south-1
4. ap-southeast-1
5. ap-southeast-2
6. ca-central-1
7. eu-central-1
8. eu-north-1
9. eu-west-1
10. eu-west-2
11. eu-west-3
12. sa-east-1
13. us-east-1
14. us-east-2
15. us-west-1
16. us-west-2
AWS Region ID [ap-northeast-1]:
```

Choose the scheduler to use with your cluster.

```
Allowed values for Scheduler:
1. sge
2. torque
3. slurm
4. awsbatch
Scheduler [sge]:
```

Choose the operating system.

```
Allowed values for Operating System:
1. alinux
2. centos6
3. centos7
4. ubuntu1604
5. ubuntu1804
Operating System [alinux]:
```

The minimum and maximum size of the cluster of compute nodes is entered. This is measured in number of instances.

```
Minimum cluster size (instances) [0]:
Maximum cluster size (instances) [10]:
```

The master and compute nodes instance types are entered. For instance types, your account instance limits are large enough to meet your requirements. For more information, see On-Demand Instance Limits in the *Amazon EC2 User Guide for Linux Instances*.

```
Master instance type [t2.micro]:
Compute instance type [t2.micro]:
```

The key pair is selected from the key pairs registered with Amazon EC2 in the selected Region.

```
Allowed values for EC2 Key Pair Name:
1. prod-uswest1-key
2. test-uswest1-key
EC2 Key Pair Name [prod-uswest1-key]:
```

After the previous steps are completed, decide whether to use an existing VPC or let AWS ParallelCluster create a VPC for you. If you don't have a properly configured VPC, AWS ParallelCluster can create a new one. It either uses both the master and compute nodes in the same public subnet, or only the master node in a public subnet with all nodes in a private subnet. It is possible to reach your limit on number of VPCs in a Region. The default limit is five VPCs per Region. For more information about this limit and how to request an increase, see VPC and Subnets in the *Amazon VPC User Guide*.

If you let AWS ParallelCluster create a VPC, you must decide whether all nodes should be in a public subnet.

```
Automate VPC creation? (y/n) [n]: y
Allowed values for Network Configuration:
1. Master in a public subnet and compute fleet in a private subnet
2. Master and compute fleet in the same public subnet
Network Configuration [Master in a public subnet and compute fleet in a private
 subnet]: 1
Beginning VPC creation. Please do not leave the terminal until the creation is
 finalized
```

If you do not create a new VPC. you must select an existing VPC.

```

```

```
Automate VPC creation? (y/n) [n]: n
Allowed values for VPC ID:
1. subnet-0b4ad9c4678d3c7ad
2. vpc-0e87c753286f37eef | ParallelClusterVPC-20191118233938 | 5 subnets inside
VPC ID [vpc-0b4ad9c4678d3c7ad]: 1
```

After the VPC has been selected, decide whether to use existing subnets or create new ones.

```
Automate Subnet creation? (y/n) [y]: y
```

```
Creating CloudFormation stack...
Do not leave the terminal until the process has finished
```

AWS Batch

From the list of valid AWS Region identifiers, choose the Region in which you want your cluster to run.

```
Allowed values for AWS Region ID:
1. ap-northeast-1
2. ap-northeast-2
3. ap-south-1
4. ap-southeast-1
5. ap-southeast-2
6. ca-central-1
7. eu-central-1
8. eu-north-1
```

```
9. eu-west-1
10. eu-west-2
11. eu-west-3
12. sa-east-1
13. us-east-1
14. us-east-2
15. us-west-1
16. us-west-2
AWS Region ID [ap-northeast-1]:
```

Choose the scheduler to use with your cluster.

```
Allowed values for Scheduler:
1. sge
2. torque
3. slurm
4. awsbatch
Scheduler [sge]:
```

When `awsbatch` is selected as the scheduler, `alinux` is used as the operating system.

The minimum and maximum size of the cluster of compute nodes is entered. This is measured in vCPUs.

```
Minimum cluster size (vcpus) [0]:
Maximum cluster size (vcpus) [10]:
```

The master node instance type is entered. When using the `awsbatch` scheduler, the compute nodes use an instance type of `optimal`.

```
Master instance type [t2.micro]:
```

The Amazon EC2 key pair is selected from the key pairs registered with Amazon EC2 in the selected Region.

```
Allowed values for EC2 Key Pair Name:
1. prod-uswest1-key
2. test-uswest1-key
EC2 Key Pair Name [prod-uswest1-key]:
```

Decide whether to use existing VPCs or let AWS ParallelCluster create VPCs for you. If you don't have a properly configured VPC, AWS ParallelCluster can create a new one. It either uses both the master and compute nodes in the same public subnet, or only the master node in a public subnet with all nodes in a private subnet. It is possible to reach your limit on number of VPCs in a Region. The default number of VPCs is five. For more information about this limit and how to request an increase, see VPC and Subnets in the *Amazon VPC User Guide*.

If you let AWS ParallelCluster create a VPC, decide whether all nodes should be in a public subnet.

```
Automate VPC creation? (y/n) [n]: y
Allowed values for Network Configuration:
1. Master in a public subnet and compute fleet in a private subnet
2. Master and compute fleet in the same public subnet
Network Configuration [Master in a public subnet and compute fleet in a private
 subnet]: 1
Beginning VPC creation. Please do not leave the terminal until the creation is
 finalized
```

If you do not create a new VPC. you must select an existing VPC

```
Automate VPC creation? (y/n) [n]: n
Allowed values for VPC ID:
1. subnet-0b4ad9c4678d3c7ad
2. vpc-0e87c753286f37eef | ParallelClusterVPC-20191118233938 | 5 subnets inside
VPC ID [vpc-0b4ad9c4678d3c7ad]: 1
```

After the VPC has been selected, decide whether to use existing subnets or create new ones.

```
Automate Subnet creation? (y/n) [y]: y
```

```
Creating CloudFormation stack...
Do not leave the terminal until the process has finished
```

When you have completed the preceding steps, a simple cluster launches into a VPC, using an existing subnet that supports public IP's (the route table for the subnet is `0.0.0.0/0 => igw-xxxxxx`). Note the following:

- The VPC must have `DNS Resolution = yes` and `DNS Hostnames = yes`.
- The VPC should also have DHCP options with the correct `domain-name` for the Region. The default DHCP Option Set already specifies the required AmazonProvidedDNS. If specifying more than one domain name server, see DHCP Options Sets in the *Amazon VPC User Guide*.

When all settings contain valid values, you can launch the cluster by running the create command.

```
$ pcluster create mycluster
```

After the cluster reaches the "CREATE_COMPLETE" status, you can connect to it by using your normal SSH client settings. For more details on connecting to Amazon EC2 instances, see the EC2 User Guide in the *Amazon EC2 User Guide for Linux Instances*.

Moving from CfnCluster to AWS ParallelCluster

AWS ParallelCluster is an enhanced version of CfnCluster.

If you currently use CfnCluster, we encourage you to start using and creating new clusters with AWS ParallelCluster instead. Although you can continue to use CfnCluster, it is no longer being developed, and no new features or functionality will be added.

The main differences between CfnCluster and AWS ParallelCluster are described in the following sections.

AWS ParallelCluster CLI manages a different set of clusters

Clusters created with the `cfncluster` CLI cannot be managed with the `pcluster` CLI. The following commands do not work on clusters created by CfnCluster:

```
pcluster list
pcluster update cluster_name
pcluster start cluster_name
```

```
pcluster status cluster_name
```

To manage clusters that you created with CfnCluster, you must use the `cfncluster` CLI.

If you need a CfnCluster package to manage your old clusters, we recommend that you install and use it from a Python Virtual Environment.

AWS ParallelCluster and CfnCluster use different IAM custom policies

Custom IAM policies that were previously used for CfnCluster cluster creation cannot be used with AWS ParallelCluster. If you require custom policies for AWS ParallelCluster, you must create new ones. See the AWS ParallelCluster guide.

AWS ParallelCluster and CfnCluster use different configuration files

The AWS ParallelCluster configuration file resides in the `~/.parallelcluster` folder. The CfnCluster configuration file resides in the `~/.cfncluster` folder.

If you want to use an existing CfnCluster configuration file with AWS ParallelCluster, you must:

1. Move the configuration file from `~/.cfncluster/config` to `~/.parallelcluster/config`.
2. If you use the `extra_json` configuration parameter, change it as shown.

CfnCluster setting:

```
extra_json = { "cfncluster" : { } }
```

AWS ParallelCluster setting:

```
extra_json = { "cluster" : { } }
```

In AWS ParallelCluster, ganglia is disabled by default

In AWS ParallelCluster, ganglia is disabled by default. To enable ganglia:

1. Set the `extra_json` parameter as shown:

   ```
   extra_json = { "cluster" : { "ganglia_enabled" : "yes" } }
   ```

2. Change the master security group to allow connections to port 80.

   The `parallelcluster-<CLUSTER_NAME>-MasterSecurityGroup-<xxx>` security group must be modified by adding a new security group rule to allow Inbound connection to port 80 from your Public IP. For more information, see Adding Rules to a Security Group in the *Amazon EC2 User Guide for Linux Instances*.

Supported Regions

AWS ParallelCluster is available in the following AWS Regions:

Region Name	Region
US East (Ohio)	us-east-2

Region Name	Region
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
EU (Frankfurt)	eu-central-1
EU (Ireland)	eu-west-1
EU (London)	eu-west-2
EU (Paris)	eu-west-3
EU (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Using AWS ParallelCluster

Topics

AWS ParallelCluster CLI Commands

`pcluster` is the AWS ParallelCluster CLI. You use the AWS ParallelCluster CLI to launch and manage HPC clusters in the AWS Cloud.

```
pcluster [ -h ] ( create | update | delete | start | stop | status | list |
                  instances | ssh | dcv | createami | configure | version ) ...
```

Arguments

`pcluster` ***`command`***

Possible choices: `configure (p. 19)`, `create (p. 19)`, `createami (p. 20)`, `dcv (p. 21)`, `delete (p. 22)`, `instances (p. 22)`, `list (p. 23)`, `ssh (p. 23)`, `start (p. 24)`, `status (p. 25)`, `stop (p. 25)`, `update (p. 26)`, `version (p. 27)`

Sub-commands:

Topics

pcluster configure

Begins an AWS ParallelCluster configuration.

```
pcluster configure [ -h ] [ -c CONFIG_FILE ]
```

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the full path of the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

For more information, see the section called “Configuring AWS ParallelCluster” (p. 11).

pcluster create

Creates a new cluster.

```
pcluster create [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] [ -nw ] [ -nr ]
                [ -u TEMPLATE_URL ] [ -t CLUSTER_TEMPLATE ]
                [ -p EXTRA_PARAMETERS ] [ -g TAGS ]
                cluster_name
```

Positional Arguments

cluster_name

Defines the name of the cluster. The AWS CloudFormation stack name is `parallelcluster-`*`cluster_name`*.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the AWS Region to use. Defaults to the Region specified by using the the section called “pcluster configure” (p. 19) command.

-nw, --nowait

Indicates not to wait for stack events after executing a stack command.

Defaults to `False`.

-nr, --norollback

Disables stack rollback on error.

Defaults to `False`.

-u *TEMPLATE_URL*, --template-url *TEMPLATE_URL*

Specifies a URL for the custom AWS CloudFormation template, if it was used at creation time.

-t *CLUSTER_TEMPLATE*, --cluster-template *CLUSTER_TEMPLATE*

Indicates the cluster template to use.

-p *EXTRA_PARAMETERS*, --extra-parameters *EXTRA_PARAMETERS*

Adds extra parameters to stack create.

-g *TAGS*, --tags *TAGS*

Specifies additional tags to add to the stack.

When the command is called and begins polling for the status of that call, it is safe to use "Ctrl-C" to exit. You can return to viewing the current status by calling `pcluster status mycluster`.

Examples:

```
$ pcluster create mycluster
$ pcluster create mycluster --tags '{ "Key1" : "Value1" , "Key2" : "Value2" }'
```

pcluster createami

(Linux/macOS) Creates a custom AMI to use with AWS ParallelCluster.

```
pcluster createami [ -h ] - ai BASE_AMI_ID - os BASE_AMI_OS [ -ap CUSTOM_AMI_NAME_PREFIX ]
                   [ -cc CUSTOM_AMI_COOKBOOK ] [ -c CONFIG_FILE ] [ -r REGION ]
```

Required Dependencies

In addition to the AWS ParallelCluster CLI, the following dependencies are required to run `pcluster createami`:

- **Packer**: Download the latest version from https://www.packer.io/downloads.html.
- **ChefDK**: Download the latest version from https://downloads.chef.io/chefdk/.

Named Arguments

-h, --help

Shows the help text for the specified command.

-ai *BASE_AMI_ID*, --ami-id *BASE_AMI_ID*

Specifies the base AMI to use for building the AWS ParallelCluster AMI.

-os *BASE_AMI_OS*, --os *BASE_AMI_OS*

Specifies the OS of the base AMI. Valid options are: `alinux`, `ubuntu1604`, `centos6`, and `centos7`.

-ap *CUSTOM_AMI_NAME_PREFIX*, --ami-name-prefix *CUSTOM_AMI_NAME_PREFIX*

Specifies the prefix name of the resulting AWS ParallelCluster AMI.

Defaults to `custom-ami-`.

-cc *CUSTOM_AMI_COOKBOOK*, --custom-cookbook *CUSTOM_AMI_COOKBOOK*

Specifies the cookbook to use to build the AWS ParallelCluster AMI.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

pcluster dcv

Interacts with the NICE DCV server running on the master instance.

```
pcluster dcv [ -h ] ( connect )
```

pcluster dcv *command*

Possible choices: connect (p. 21)

Note
Support for the `pcluster dcv` command was added in AWS ParallelCluster 2.5.0.

Named Arguments

-h, --help

Shows the help text for the specified command.

Sub-commands

pcluster dcv connect

```
pcluster dcv connect [ -h ] [ -k SSH_KEY_PATH ] cluster_name
```

Positional Arguments

cluster_name

Specifies the name of the cluster to connect to.

Named Arguments

-h, --help

Shows the help text for the specified command.

-k *SSH_KEY_PATH*, --key-path *SSH_KEY_PATH*

Key path of the SSH key to use for the connection.

It must be the one specified at cluster creation time in the `key_name` configuration parameter.

Example:

```
$ pcluster dcv connect -k ~/.ssh/id_rsa
```

Opens the default browser to connect to the NICE DCV session running on the master instance.

A new NICE DCV session is created if one is not already started.

pcluster delete

Deletes a cluster.

```
pcluster delete [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] [ -nw ] cluster_name
```

Positional Arguments

cluster_name

Specifies the name of the cluster to delete.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

When the command is called and begins polling for the status of that call, it is safe to use "Ctrl-C" to exit. You can return to viewing the current status by calling `pcluster status mycluster`.

pcluster instances

Displays a list of all instances in a cluster.

```
pcluster instances [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] cluster_name
```

Positional Arguments

cluster_name

Displays the instances for the cluster with the provided name.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

pcluster list

Displays a list of stacks that are associated with AWS ParallelCluster.

```
pcluster list [ -h ] [ -c CONFIG_FILE ] [ -r REGION ]
```

Named Arguments

-h, --help

Shows the help text for the specified command.

--color

Displays the cluster status in color.

Defaults to `False`.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `c`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

Lists the name of any AWS CloudFormation stacks named `parallelcluster-*`.

pcluster ssh

Runs an `ssh` command with the user name and IP address of the cluster pre-populated. Arbitrary arguments are appended to the end of the `ssh` command. This command can be customized in the aliases section of the configuration file.

```
pcluster ssh [ -h ] [ -d ] cluster_name
```

Positional Arguments

cluster_name

Specifies the name of the cluster to connect to.

Named Arguments

-h, --help

Shows the help text for the specified command.

-d, --dryrun

Prints the command that would be run and exits.

Defaults to `False`.

Example:

```
$ pcluster ssh -d mycluster -i ~/.ssh/id_rsa
```

Returns an `ssh` command with the user name and IP address of the cluster pre-populated:

```
$ ssh ec2-user@1.1.1.1 -i ~/.ssh/id_rsa
```

The `ssh` command is defined in the global configuration file under the [aliases] section (p. 56). It can be customized as follows.

```
[ aliases ]
ssh = ssh {CFN_USER}@{MASTER_IP} {ARGS}
```

Variables substituted:

CFN_USER

The user name for the the section called "`base_os`" (p. 59) that is selected.

MASTER_IP

The IP address of the master node.

ARGS

Optional arguments to pass to the `ssh` command.

pcluster start

Starts the compute fleet for a cluster that has been stopped.

```
pcluster start [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] cluster_name
```

Positional Arguments

cluster_name

Starts the compute fleet of the provided cluster name.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

This command sets the Auto Scaling Group parameters to one of the following:

- The initial configuration values (`max_queue_size` and `initial_queue_size`) from the template that was used to create the cluster.
- The configuration values that were used to update the cluster since it was first created.

pcluster status

Pulls the current status of the cluster.

```
pcluster status [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] [ -nw ] cluster_name
```

Positional Arguments

cluster_name

Shows the status of the cluster with the provided name.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

-nw, --nowait

Indicates not to wait for stack events after executing a stack command.

Defaults to `False`.

pcluster stop

Stops the compute fleet, leaving the master node running.

```
pcluster stop [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] cluster_name
```

Positional Arguments

cluster_name

Stops the compute fleet of the provided cluster name.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

Sets the Auto Scaling Group parameters to min/max/desired = 0/0/0, and terminates the compute fleet. The master remains running. To terminate all EC2 resources and avoid EC2 charges, consider deleting the cluster.

pcluster update

Updates a running cluster by using the values in the configuration file.

```
pcluster update [ -h ] [ -c CONFIG_FILE ] [ -r REGION ] [ -nw ] [ -nr ]
                [ -t CLUSTER_TEMPLATE ] [ -p EXTRA_PARAMETERS ] [ -rd ]
                cluster_name
```

Positional Arguments

cluster_name

Specifies the name of the cluster to update.

Named Arguments

-h, --help

Shows the help text for the specified command.

-c *CONFIG_FILE*, --config *CONFIG_FILE*

Specifies the alternative configuration file to use.

Defaults to `~/.parallelcluster/config`.

-r *REGION*, --region *REGION*

Specifies the Region to connect to.

-nw, --nowait

Indicates not to wait for stack events after executing a stack command.

Defaults to `False`.

-nr, --norollback

Disables AWS CloudFormation stack rollback on error.

Defaults to `False`.

-t *CLUSTER_TEMPLATE*, --cluster-template *CLUSTER_TEMPLATE*

Specifies the section of the cluster template to use.

-p *EXTRA_PARAMETERS*, --extra-parameters *EXTRA_PARAMETERS*

Adds extra parameters to a stack update.

-rd, --reset-desired

Resets the current capacity of an Auto Scaling Group to the initial configuration values.

Defaults to `False`.

When the command is called and begins polling for the status of that call, it is safe to use "Ctrl-C" to exit. You can return to viewing the current status by calling `pcluster status mycluster`.

pcluster version

Displays the AWS ParallelCluster version.

```
pcluster version [ -h ]
```

For command-specific flags, run: `pcluster [command] -help`.

Named Arguments

-h, --help

Shows the help text for the specified command.

When the command is called and begins polling for the status of that call, it is safe to use "Ctrl-C" to exit. You can return to viewing the current status by calling `pcluster status mycluster`.

Network Configurations

AWS ParallelCluster uses Amazon Virtual Private Cloud (VPC) for networking. VPC provides a flexible and configurable networking platform in which to deploy clusters.

The VPC must have `DNS Resolution = yes`, `DNS Hostnames = yes` and DHCP options with the correct domain-name for the Region. The default DHCP Option Set already specifies the required *AmazonProvidedDNS*. If specifying more than one domain name server, see DHCP Options Sets in the *Amazon VPC User Guide*.

AWS ParallelCluster supports the following high-level configurations:

- One subnet for both master and compute instances.
- Two subnets, with the master in one public subnet, and compute instances in a private subnet. The subnets can be new or existing.

All of these configurations can operate with or without public IP addressing. AWS ParallelCluster can also be deployed to use an HTTP proxy for all AWS requests. The combinations of these configurations result in many deployment scenarios. For example, you can configure a single public subnet with all access over the internet., Or you can configure a fully private network using AWS Direct Connect and HTTP proxy for all traffic.

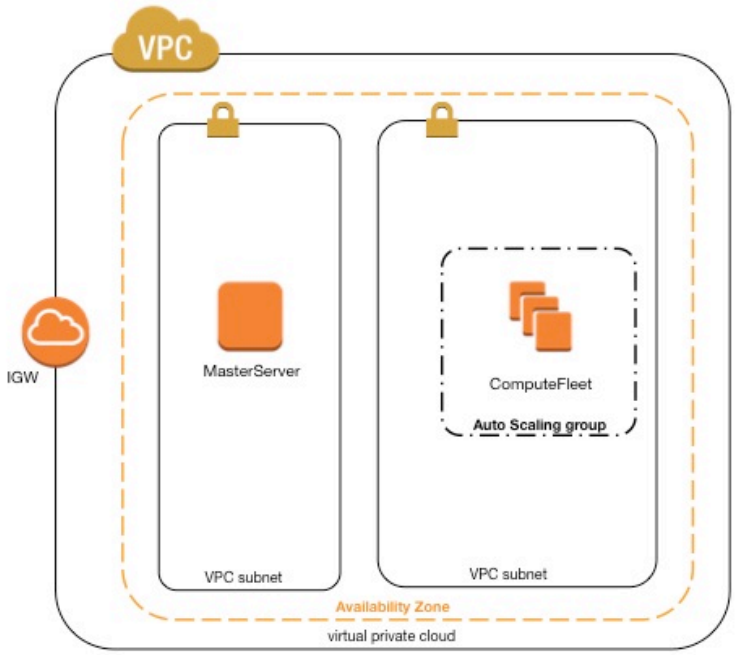
See the following architecture diagrams for illustrations of some of these scenarios:

AWS ParallelCluster in a Single Public Subnet

The configuration for this architecture requires the following settings:

```
[vpc public]
vpc_id = vpc-xxxxxx
master_subnet_id = subnet-<public>
```

AWS ParallelCluster Using Two Subnets



The configuration to create a new private subnet for compute instances requires the following settings:

Note that all values are examples only

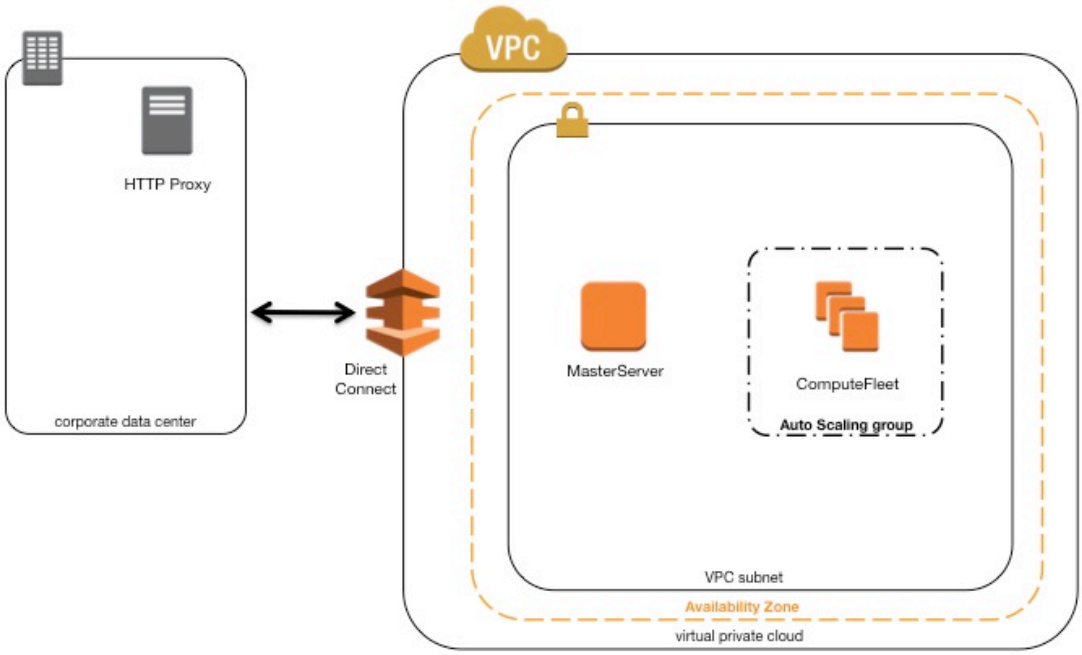
```
[vpc public-private-new]
vpc_id = vpc-xxxxxx
master_subnet_id = subnet-<public>
compute_subnet_cidr = 10.0.1.0/24
```

The configuration to use an existing private network requires the following settings:

```
[vpc public-private-existing]
vpc_id = vpc-xxxxxx
master_subnet_id = subnet-<public>
compute_subnet_id = subnet-<private>
```

Both of these configurations require a NAT Gateway or an internal PROXY to enable web access for compute instances.

AWS ParallelCluster in a Single Private Subnet Connected Using AWS Direct Connect



The configuration for this architecture requires the following settings:

```
[cluster private-proxy]
proxy_server = http://proxy.corp.net:8080
```

```
[vpc private-proxy]
vpc_id = vpc-xxxxxx
master_subnet_id = subnet-<private>
use_public_ips = false
```

When use_public_ips is set to false, the VPC must be correctly set up to use the Proxy for all traffic. Web access is required for both master and compute instances.

AWS ParallelCluster with awsbatch Scheduler

When you use awsbatch as the scheduler type, AWS ParallelCluster creates an AWS Batch managed compute environment. The AWS Batch environment takes care of managing Amazon Elastic Container Service (Amazon ECS) container instances, which are launched in the compute_subnet. In order for AWS Batch to function correctly, Amazon ECS container instances need external network access to communicate with the Amazon ECS service endpoint. This translates into the following scenarios:

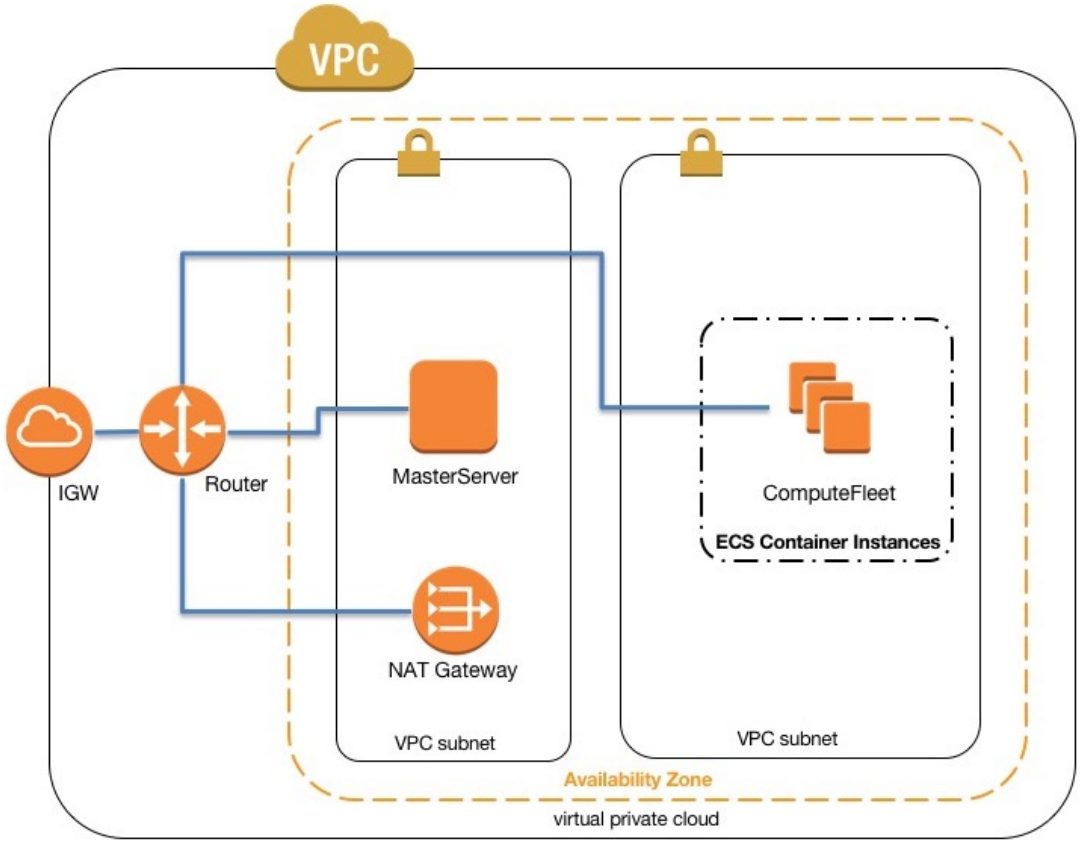
- The compute_subnet uses a NAT Gateway to access the internet. (We recommended this approach.)
- Instances launched in the compute_subnet have public IP addresses and can reach the internet through an Internet Gateway.

Additionally, if you are interested in multi-node parallel jobs (from the AWS Batch docs):

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 Amazon VPC subnet as its host compute resource. Any security groups that are applied to your compute resources are also applied to it.

When using Amazon ECS Task Networking, the awsvpc network mode does not provide elastic network interfaces with public IP addresses for tasks that use the Amazon EC2 launch type. To access the internet, tasks that use the Amazon EC2 launch type must be launched in a private subnet that is configured to use a NAT Gateway.

This leaves us with the only option, to configure a NAT Gateway in order to enable the cluster to execute multi-node parallel jobs.



Additional details can be found in the following AWS documents:

- AWS Batch Managed Compute Environments
- AWS Batch Multi-node Parallel Jobs
- Amazon ECS Task Networking with the `awsvpc` Network Mode

Custom Bootstrap Actions

AWS ParallelCluster can execute arbitrary code either before (pre-install) or after (post-install) the main bootstrap action during cluster creation. This code is typically stored in Amazon Simple Storage Service (Amazon S3) and accessed via HTTPS during cluster creation. The code is executed as root and can be in any script language that is supported by the cluster OS, typically *bash* or *python*.

Pre-install actions are called before any cluster deployment bootstrap, such as configuring NAT, Amazon Elastic Block Store (Amazon EBS) or the scheduler. Typical pre-install actions can include modifying storage, adding extra users, or adding packages.

Post-install actions are called after cluster bootstrap is complete, as the last action before an instance is considered complete. Typical post-install actions can include changing scheduler settings, modifying storage, or modifying packages.

Arguments can be passed to scripts by specifying them in the configuration. These are passed double-quoted to the pre-install or post-install actions.

If a pre-install or post-install action fails, the instance bootstrap fails. Success is signaled with an exit code of 0. Any other exit code is considered a failure.

You can differentiate between master and compute nodes execution. Source the `/etc/parallelcluster/cfnconfig` file and evaluate the `cfn_node_type` environment variable, whose possible values are "MasterServer" and "ComputeFleet" for the master and compute nodes respectively.

```
#!/bin/bash

. "/etc/parallelcluster/cfnconfig"

case "${cfn_node_type}" in
    MasterServer)
        echo "I am the master node" >> /tmp/master.txt
    ;;
    ComputeFleet)
        echo "I am a compute node" >> /tmp/compute.txt
    ;;
    *)
    ;;
esac
```

Configuration

The following configuration settings are used to define pre-install and post-install actions and arguments.

```
# URL to a preinstall script. This is executed before any of the boot_as_* scripts are run
# (defaults to NONE)
pre_install = NONE
# Arguments to be passed to preinstall script
# (defaults to NONE)
pre_install_args = NONE
# URL to a postinstall script. This is executed after any of the boot_as_* scripts are run
# (defaults to NONE)
post_install = NONE
# Arguments to be passed to postinstall script
# (defaults to NONE)
post_install_args = NONE
```

Arguments

The first two arguments — `$0` and `$1` — are reserved for the script name and url.

```
$0 => the script name
$1 => s3 url
$n => args set by pre/post_install_args
```

Example

The following steps create a simple post-install script that installs the R packages in a cluster.

1. Create a script.

```
#!/bin/bash

echo "post-install script has $# arguments"
```

```
for arg in "$@"
do
    echo "arg: ${arg}"
done

yum -y install "${@:2}"
```

2. Upload the script with the correct permissions to Amazon S3.

```
$ aws s3 cp --acl public-read /path/to/myscript.sh s3://<bucket-name>/myscript.sh
```

3. Update the AWS ParallelCluster configuration to include the new post-install action.

```
[cluster default]
...
post_install = https://<bucket-name>.s3.amazonaws.com/myscript.sh
post_install_args = "R curl wget"
```

   If the bucket does not have public-read permission, use `s3` as the URL protocol.

```
[cluster default]
...
post_install = s3://<bucket-name>/myscript.sh
post_install_args = "R curl wget"
```

4. Launch the cluster.

```
$ pcluster create mycluster
```

5. Verify the output.

```
$ less /var/log/cfn-init.log
2019-04-11 10:43:54,588 [DEBUG] Command runpostinstall output: post-install script has 4
 arguments
arg: s3://eu-eu-west-1/test.sh
arg: R
arg: curl
arg: wget
Loaded plugins: dkms-build-requires, priorities, update-motd, upgrade-helper
Package R-3.4.1-1.52.amzn1.x86_64 already installed and latest version
Package curl-7.61.1-7.91.amzn1.x86_64 already installed and latest version
Package wget-1.18-4.29.amzn1.x86_64 already installed and latest version
Nothing to do
```

Working with Amazon S3

You can access Amazon S3 from within AWS ParallelCluster. You control the access to Amazon S3 through two parameters in the AWS ParallelCluster configuration.

```
# Specify Amazon S3 resource which AWS ParallelCluster nodes will be granted read-only
 access
# (defaults to NONE)
s3_read_resource = NONE
# Specify Amazon S3 resource which AWS ParallelCluster nodes will be granted read-write
 access
# (defaults to NONE)
s3_read_write_resource = NONE
```

Both parameters accept either * or a valid Amazon S3 ARN. For details about specifying Amazon S3 ARNs, see Amazon S3 ARN Format in the *AWS General Reference*

Examples

The following example gives you read access to any object in the Amazon S3 bucket *my_corporate_bucket*.

```
s3_read_resource = arn:aws:s3:::my_corporate_bucket/*
```

This following example gives you read access to the bucket, but does **not** let you read items from the bucket.

```
s3_read_resource = arn:aws:s3:::my_corporate_bucket
```

This last example gives you read access to the bucket and to the items stored in the bucket..

```
s3_read_resource = arn:aws:s3:::my_corporate_bucket*
```

Working with Spot Instances

AWS ParallelCluster uses Spot Instances if the cluster configuration has set the section called "cluster_type" (p. 59) = spot. The primary feature of Spot Instances is they are available for less than the cost of On-Demand Instances, but it is possible that they might be interrupted. The effect of the interruption varies depending on the scheduler used. It may help to take advantage of *Spot Instance interruption notices*, which provide a two-minute warning before Amazon EC2 must stop or terminate your Spot Instance. For more information, see Spot Instance Interruptions in *Amazon EC2 User Guide for Linux Instances*. The following sections describe three scenarios in which Spot Instances can be interrupted.

Scenario 1: Spot Instance with no running jobs is interrupted

When this interruption occurs, AWS ParallelCluster tries to replace the instance if the scheduler queue has pending jobs that require additional instances, or if the number of active instances is lower than the the section called "initial_queue_size" (p. 63) setting. If AWS ParallelCluster is unable to provision new instances, then a request for new instances is periodically repeated.

Scenario 2: Spot Instance running single node jobs is interrupted

The behavior of this interruption depends on the scheduler being used.

Slurm

The job is terminated and given a state code of `NODE_FAIL`. The compute instance is removed from the scheduler queue.

SGE

The job is terminated. If the job has enabled the rerun flag (using either `qsub -r yes` or `qalter -r yes`) or the queue has the `rerun` configuration set to `TRUE`, then the job is rescheduled. The

compute instance is removed from the scheduler queue. This behavior comes from these SGE configuration parameters:

- `reschedule_unknown 00:00:30`
- `ENABLE_FORCED_QDEL_IF_UNKNOWN`
- `ENABLE_RESCHEDULE_KILL=1`

Torque

The job is removed from the system and the node is removed from the scheduler. The job is not rerun. If multiple jobs are running on the instance when it is interrupted, Torque may time out during node removal. An error may display in the the section called "`sqswatcher`" (p. 86) log file. This does not affect scaling logic, and a proper cleanup is performed by subsequent retries.

Scenario 3: Spot Instance running multi-node jobs is interrupted

The behavior of this interruption depends on the scheduler being used.

Slurm

The job is terminated and given a state code of `NODE_FAIL`. The compute instance is removed from the scheduler queue. Other nodes that were running the terminated jobs may be scaled down after the configured the section called "`scaledown_idletime`" (p. 79) time has passed.

SGE

The job is not terminated and continues to run on the remaining nodes. The compute node is removed from the scheduler queue, but will appear in the hosts list as an orphaned and unavailable node.

The user must delete the job when this occurs (`qdel <jobid>`). The node still displays in the hosts list (`qhost`), although this does not affect AWS ParallelCluster. To remove the host from the list, you could run the following command after replacing the instance.

```
sudo -- bash -c 'source /etc/profile.d/sge.sh; qconf -dattr hostgroup
 hostlist <hostname> @allhosts; qconf -de <hostname>'
```

Torque

The job is removed from the system and the node is removed from the scheduler. The job is not rerun. If multiple jobs are running on the instance when it is interrupted, Torque may time out during node removal. An error may display in the the section called "`sqswatcher`" (p. 86) log file. This does not affect scaling logic, and a proper cleanup is performed by subsequent retries.

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

AWS Identity and Access Management Roles in AWS ParallelCluster

AWS ParallelCluster uses AWS Identity and Access Management (IAM) roles for Amazon EC2 to enable instances to access AWS services for the deployment and operation of a cluster. By default, the IAM role for Amazon EC2 is created as part of the cluster creation by AWS CloudFormation. This means that the

user that creates the cluster must have the appropriate level of permissions, as described in the following sections.

AWS ParallelCluster uses multiple AWS services to deploy and operate a cluster. See the complete list in the AWS Services used in AWS ParallelCluster (p. 89) section.

Defaults

When you use the default settings for cluster creation, an IAM role for Amazon EC2 is created by the cluster. The user that is creating the cluster must have the right level of permissions to create all of the resources required to launch the cluster, including an IAM role for Amazon EC2. Typically, the IAM user must have the permissions of an *AdministratorAccess* managed policy. For details about managed policies, see AWS Managed Policies in the *IAM User Guide*.

Using an Existing IAM Role for Amazon EC2

You can use an existing IAM role for Amazon EC2 when creating a cluster, but you must first define the IAM policy and role before attempting to launch the cluster. Typically, you choose an existing IAM role for Amazon EC2 to reduce the permissions that are granted to users as they launch clusters. The following example shows an IAM policy and role for both Amazon EC2 and the AWS ParallelCluster. You must create both as individual policies in IAM and then attach to the appropriate resources. In both policies, replace *<REGION>*, *<AWS ACCOUNT ID>*, and similar strings with the appropriate values.

ParallelClusterInstancePolicy

The following example sets the `ParallelClusterInstancePolicy`, using SGE, Slurm, or Torque as the scheduler:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes",
                "ec2:AttachVolume",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeInstances",
                "ec2:DescribeRegions"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "Sid": "EC2"
        },
        {
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "Sid": "DynamoDBList"
        },
        {
            "Action": [
                "sqs:SendMessage",
```

```
                "sqs:ReceiveMessage",
                "sqs:ChangeMessageVisibility",
                "sqs:DeleteMessage",
                "sqs:GetQueueUrl"
            ],
            "Resource": [
                "arn:aws:sqs:<REGION>:<AWS ACCOUNT ID>:parallelcluster-*"
            ],
            "Effect": "Allow",
            "Sid": "SQSQueue"
        },
        {
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "autoscaling:SetDesiredCapacity",
                "autoScaling:UpdateAutoScalingGroup",
                "autoscaling:DescribeTags",
                "autoScaling:SetInstanceHealth"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "Sid": "Autoscaling"
        },
        {
            "Action": [
                "cloudformation:DescribeStacks",
                "cloudformation:DescribeStackResource"
            ],
            "Resource": [
                "arn:aws:cloudformation:<REGION>:<AWS ACCOUNT ID>:stack/parallelcluster-*/
*"
            ],
            "Effect": "Allow",
            "Sid": "CloudFormation"
        },
        {
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:GetItem",
                "dynamodb:DeleteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": [
                "arn:aws:dynamodb:<REGION>:<AWS ACCOUNT ID>:table/parallelcluster-*"
            ],
            "Effect": "Allow",
            "Sid": "DynamoDBTable"
        },
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::<REGION>-aws-parallelcluster/*"
            ],
            "Effect": "Allow",
            "Sid": "S3GetObj"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::NONE/batch/*"
            ],
            "Effect": "Allow",
            "Sid": "S3PutObj"
        },
        {
            "Resource": [
                "*"
            ],
            "Action": [
                "sqs:ListQueues"
            ],
            "Effect": "Allow",
            "Sid": "SQSList"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::<AWS ACCOUNT ID>:role/parallelcluster-*"
            ],
            "Effect": "Allow",
            "Sid": "BatchJobPassRole"
        },
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::dcv-license.<REGION>/*"
            ],
            "Effect": "Allow",
            "Sid": "DcvLicense"
        }
    ]
}
```

The following example sets the `ParallelClusterInstancePolicy`, using `awsbatch` as the scheduler. You must include the same policies that are assigned to the `BatchUserRole` that is defined in the AWS Batch AWS CloudFormation nested stack. The `BatchUserRole` ARN is provided as a stack output. Here is an overview of the required permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "batch:SubmitJob",
                "batch:RegisterJobDefinition",
                "cloudformation:DescribeStacks",
                "ecs:ListContainerInstances",
                "ecs:DescribeContainerInstances",
                "logs:GetLogEvents",
                "logs:FilterLogEvents",
                "s3:PutObject",
                "s3:Get*",
                "s3:DeleteObject",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:batch:<REGION>:<AWS ACCOUNT ID>:job-definition/<AWS_BATCH_STACK -
 JOB_DEFINITION_SERIAL_NAME>:1",
```

```
                "arn:aws:batch:<REGION>:<AWS ACCOUNT ID>:job-definition/<AWS_BATCH_STACK -
 JOB_DEFINITION_MNP_NAME>*",
                "arn:aws:batch:<REGION>:<AWS ACCOUNT ID>:job-queue/<AWS_BATCH_STACK -
 JOB_QUEUE_NAME>",
                "arn:aws:cloudformation:<REGION>:<AWS ACCOUNT ID>:stack/<STACK NAME>/*",
                "arn:aws:s3:::<RESOURCES S3 BUCKET>/batch/*",
                "arn:aws:iam::<AWS ACCOUNT ID>:role/<AWS_BATCH_STACK - JOB_ROLE>",
                "arn:aws:ecs:<REGION>:<AWS ACCOUNT ID>:cluster/<ECS COMPUTE ENVIRONMENT>",
                "arn:aws:ecs:<REGION>:<AWS ACCOUNT ID>:container-instance/*",
                "arn:aws:logs:<REGION>:<AWS ACCOUNT ID>:log-group:/aws/batch/job:log-
stream:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::<RESOURCES S3 BUCKET>"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "batch:DescribeJobQueues",
                "batch:TerminateJob",
                "batch:DescribeJobs",
                "batch:CancelJob",
                "batch:DescribeJobDefinitions",
                "batch:ListJobs",
                "batch:DescribeComputeEnvironments",
                "ec2:DescribeInstances"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

ParallelClusterUserPolicy

The following example sets the `ParallelClusterUserPolicy`, using SGE, Slurm, or Torque as the scheduler.

Note
If you use a custom role, the section called “ec2_iam_role” (p. 61) = *<role_name>*, you must change the IAM resource to include the name of that role from:
`"Resource": "arn:aws:iam::<AWS ACCOUNT ID>:role/parallelcluster-*"`
To:
`"Resource": "arn:aws:iam::<AWS ACCOUNT ID>:role/<role_name>"`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EC2Describe",
            "Action": [
                "ec2:DescribeKeyPairs",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
```

```
                "ec2:DescribePlacementGroups",
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeSnapshots",
                "ec2:DescribeVolumes",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeAddresses",
                "ec2:CreateTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeAvailabilityZones"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "NetworkingEasyConfig",
            "Action": [
                "ec2:CreateVpc",
                "ec2:ModifyVpcAttribute",
                "ec2:DescribeNatGateways",
                "ec2:CreateNatGateway",
                "ec2:DescribeInternetGateways",
                "ec2:CreateInternetGateway",
                "ec2:AttachInternetGateway",
                "ec2:DescribeRouteTables",
                "ec2:CreateRouteTable",
                "ec2:AssociateRouteTable",
                "ec2:CreateSubnet",
                "ec2:ModifySubnetAttribute"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "EC2Modify",
            "Action": [
                "ec2:CreateVolume",
                "ec2:RunInstances",
                "ec2:AllocateAddress",
                "ec2:AssociateAddress",
                "ec2:AttachNetworkInterface",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateNetworkInterface",
                "ec2:CreateSecurityGroup",
                "ec2:ModifyVolumeAttribute",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteVolume",
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:DisassociateAddress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:ReleaseAddress",
                "ec2:CreatePlacementGroup",
                "ec2:DeletePlacementGroup"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AutoScalingDescribe",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
```

```
            "autoscaling:DescribeAutoScalingInstances"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "AutoScalingModify",
        "Action": [
            "autoscaling:CreateAutoScalingGroup",
            "ec2:CreateLaunchTemplate",
            "ec2:ModifyLaunchTemplate",
            "ec2:DeleteLaunchTemplate",
            "ec2:DescribeLaunchTemplates",
            "ec2:DescribeLaunchTemplateVersions",
            "autoscaling:PutNotificationConfiguration",
            "autoscaling:UpdateAutoScalingGroup",
            "autoscaling:PutScalingPolicy",
            "autoscaling:DescribeScalingActivities",
            "autoscaling:DeleteAutoScalingGroup",
            "autoscaling:DeletePolicy",
            "autoscaling:DisableMetricsCollection",
            "autoscaling:EnableMetricsCollection"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "DynamoDBDescribe",
        "Action": [
            "dynamodb:DescribeTable"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "DynamoDBModify",
        "Action": [
          "dynamodb:CreateTable",
          "dynamodb:DeleteTable",
          "dynamodb:TagResource"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "SQSDescribe",
        "Action": [
            "sqs:GetQueueAttributes"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "SQSModify",
        "Action": [
            "sqs:CreateQueue",
            "sqs:SetQueueAttributes",
            "sqs:DeleteQueue",
            "sqs:TagQueue"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "SNSDescribe",
        "Action": [
```

```
                "sns:ListTopics",
                "sns:GetTopicAttributes"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "SNSModify",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:DeleteTopic"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "CloudFormationDescribe",
            "Action": [
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStackResource",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks",
                "cloudformation:ListStacks",
                "cloudformation:GetTemplate"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "CloudFormationModify",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:UpdateStack"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "S3ParallelClusterReadOnly",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::<REGION>-aws-parallelcluster*"
            ]
        },
        {
            "Sid": "IAMModify",
            "Action": [
                "iam:PassRole",
                "iam:CreateRole",
                "iam:CreateServiceLinkedRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:TagRole",
                "iam:SimulatePrincipalPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::<AWS ACCOUNT ID>:role/<PARALLELCLUSTER EC2 ROLE
NAME>"
        },
        {
```

```
            "Sid": "IAMCreateInstanceProfile",
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:DeleteInstanceProfile"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::<AWS ACCOUNT ID>:instance-profile/*"
        },
        {
            "Sid": "IAMInstanceProfile",
            "Action": [
                "iam:AddRoleToInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:GetRolePolicy",
                "iam:GetPolicy",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:PutRolePolicy",
                "iam:DeleteRolePolicy"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "EFSDescribe",
            "Action": [
                "elasticfilesystem:DescribeMountTargets",
                "elasticfilesystem:DescribeMountTargetSecurityGroups",
                "ec2:DescribeNetworkInterfaceAttribute"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "SSMDescribe",
            "Action": [
                "ssm:GetParametersByPath"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "FSx",
            "Effect": "Allow",
            "Action": [
                "fsx:*"
            ],
            "Resource": "*"
        }
    ]
}
```

The following example sets the ParallelClusterUserPolicy, using `awsbatch` as the scheduler:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EC2Describe",
      "Action": [
        "ec2:DescribeKeyPairs",
        "ec2:DescribeRegions",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
```

```
      "ec2:DescribePlacementGroups",
      "ec2:DescribeImages",
      "ec2:DescribeInstances",
      "ec2:DescribeInstanceStatus",
      "ec2:DescribeSnapshots",
      "ec2:DescribeVolumes",
      "ec2:DescribeVpcAttribute",
      "ec2:DescribeAddresses",
      "ec2:CreateTags",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DescribeAvailabilityZones"
    ],
    "Effect": "Allow",
    "Resource": "*"
  },
  {
      "Sid": "NetworkingEasyConfig",
      "Action": [
          "ec2:CreateVpc",
          "ec2:ModifyVpcAttribute",
          "ec2:DescribeNatGateways",
          "ec2:CreateNatGateway",
          "ec2:DescribeInternetGateways",
          "ec2:CreateInternetGateway",
          "ec2:AttachInternetGateway",
          "ec2:DescribeRouteTables",
          "ec2:CreateRouteTable",
          "ec2:AssociateRouteTable",
          "ec2:CreateSubnet",
          "ec2:ModifySubnetAttribute"
      ],
      "Effect": "Allow",
      "Resource": "*"
  },
  {
    "Sid": "EC2Modify",
    "Action": [
      "ec2:CreateVolume",
      "ec2:RunInstances",
      "ec2:AllocateAddress",
      "ec2:AssociateAddress",
      "ec2:AttachNetworkInterface",
      "ec2:AuthorizeSecurityGroupEgress",
      "ec2:AuthorizeSecurityGroupIngress",
      "ec2:CreateNetworkInterface",
      "ec2:CreateSecurityGroup",
      "ec2:ModifyVolumeAttribute",
      "ec2:ModifyNetworkInterfaceAttribute",
      "ec2:DeleteNetworkInterface",
      "ec2:DeleteVolume",
      "ec2:TerminateInstances",
      "ec2:DeleteSecurityGroup",
      "ec2:DisassociateAddress",
      "ec2:RevokeSecurityGroupIngress",
      "ec2:ReleaseAddress",
      "ec2:CreatePlacementGroup",
      "ec2:DeletePlacementGroup"
    ],
    "Effect": "Allow",
    "Resource": "*"
  },
  {
    "Sid": "DynamoDB",
    "Action": [
      "dynamodb:DescribeTable",
      "dynamodb:CreateTable",
```

```
        "dynamodb:DeleteTable",
        "dynamodb:TagResource"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:dynamodb:<REGION>:<AWS ACCOUNT ID>:table/parallelcluster-*"
    },
    {
      "Sid": "CloudFormation",
      "Action": [
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResource",
        "cloudformation:DescribeStackResources",
        "cloudformation:DescribeStacks",
        "cloudformation:ListStacks",
        "cloudformation:GetTemplate",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:cloudformation:<REGION>:<AWS ACCOUNT ID>:stack/parallelcluster-
*"
    },
    {
      "Sid": "SQS",
      "Action": [
        "sqs:GetQueueAttributes",
        "sqs:CreateQueue",
        "sqs:SetQueueAttributes",
        "sqs:DeleteQueue",
        "sqs:TagQueue"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "SQSQueue",
      "Action": [
        "sqs:SendMessage",
        "sqs:ReceiveMessage",
        "sqs:ChangeMessageVisibility",
        "sqs:DeleteMessage",
        "sqs:GetQueueUrl"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:<REGION>:<AWS ACCOUNT ID>:parallelcluster-*"
    },
    {
      "Sid": "SNS",
      "Action": [
        "sns:ListTopics",
        "sns:GetTopicAttributes",
        "sns:CreateTopic",
        "sns:Subscribe",
        "sns:DeleteTopic"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "IAMRole",
      "Action": [
        "iam:PassRole",
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:GetRole",
        "iam:TagRole",
```

```
      "iam:SimulatePrincipalPolicy"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::<AWS ACCOUNT ID>:role/parallelcluster-*"
  },
  {
    "Sid": "IAMInstanceProfile",
    "Action": [
      "iam:CreateInstanceProfile",
      "iam:DeleteInstanceProfile",
      "iam:GetInstanceProfile",
      "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::<AWS ACCOUNT ID>:instance-profile/*"
  },
  {
    "Sid": "IAM",
    "Action": [
      "iam:AddRoleToInstanceProfile",
      "iam:RemoveRoleFromInstanceProfile",
      "iam:GetRolePolicy",
      "iam:PutRolePolicy",
      "iam:DeleteRolePolicy",
      "iam:GetPolicy",
      "iam:AttachRolePolicy",
      "iam:DetachRolePolicy"
    ],
    "Effect": "Allow",
    "Resource": "*"
  },
  {
    "Sid": "S3ResourcesBucket",
    "Action": ["s3:*"],
    "Effect": "Allow",
    "Resource": ["arn:aws:s3:::parallelcluster-*"]
  },
  {
    "Sid": "S3ParallelClusterReadOnly",
    "Action": [
      "s3:Get*",
      "s3:List*"
    ],
    "Effect": "Allow",
    "Resource": ["arn:aws:s3:::<REGION>-aws-parallelcluster/*"]
  },
  {
    "Sid": "Lambda",
    "Action": [
      "lambda:CreateFunction",
      "lambda:DeleteFunction",
      "lambda:GetFunctionConfiguration",
      "lambda:InvokeFunction",
      "lambda:AddPermission",
      "lambda:RemovePermission"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:lambda:<REGION>:<AWS ACCOUNT ID>:function:parallelcluster-*"
  },
  {
    "Sid": "Logs",
    "Effect": "Allow",
    "Action": ["logs:*"],
    "Resource": "arn:aws:logs:<REGION>:<AWS ACCOUNT ID>:*"
  },
  {
```

```
      "Sid": "CodeBuild",
      "Effect": "Allow",
      "Action": ["codebuild:*"],
      "Resource": "arn:aws:codebuild:<REGION>:<AWS ACCOUNT ID>:project/parallelcluster-*"
    },
    {
      "Sid": "ECR",
      "Effect": "Allow",
      "Action": ["ecr:*"],
      "Resource": "*"
    },
    {
      "Sid": "Batch",
      "Effect": "Allow",
      "Action": ["batch:*"],
      "Resource": "*"
    },
    {
      "Sid": "AmazonCloudWatchEvents",
      "Effect": "Allow",
      "Action": ["events:*"],
      "Resource": "*"
    }
  ]
}
```

AWS ParallelCluster Batch CLI Commands

When you use the `awsbatch` scheduler, the AWS ParallelCluster batch CLI commands are automatically installed in the AWS ParallelCluster master node. The CLI uses AWS Batch APIs and permits to:

- Submit and manage jobs
- Monitor jobs, queues, and hosts
- Mirror traditional scheduler commands

Topics

awsbsub

Submits jobs to the cluster's job queue.

```
awsbsub [-h] [-jn JOB_NAME] [-c CLUSTER] [-cf] [-w WORKING_DIR]
        [-pw PARENT_WORKING_DIR] [-if INPUT_FILE] [-p VCPUS] [-m MEMORY]
        [-e ENV] [-eb ENV_BLACKLIST] [-r RETRY_ATTEMPTS] [-t TIMEOUT]
        [-n NODES] [-a ARRAY_SIZE] [-d DEPENDS_ON]
        [command] [arguments [arguments ...]]
```

Positional Arguments

command

Submits the job (the command specified must be available on the compute instances), or specifies the file name to be transferred (also see the `--command-file` option).

arguments

(Optional) Specifies arguments for the command or the command-file.

Named Arguments

-jn *JOB_NAME*, --job-name *JOB_NAME*

Names the job. The first character must be alphanumeric. The job name can contain up to 128 characters. Letters (both uppercase and lowercase), numbers, hyphens, and underscores are allowed.

-c *CLUSTER*, --cluster *CLUSTER*

Specifies the cluster to use.

-cf, --command-file

Indicates that the command is a file to be transferred to the compute instances.

Default: False

-w *WORKING_DIR*, --working-dir *WORKING_DIR*

Specifies the folder to use as the job's working directory. If a working directory is not specified, the job is executed in the `job-<AWS_BATCH_JOB_ID>` subfolder of the user's home directory. You can use either this parameter or the `--parent-working-dir` parameter.

-pw *PARENT_WORKING_DIR*, --parent-working-dir *PARENT_WORKING_DIR*

Specfies the parent folder of the job's working directory. If a parent working directory is not specified, it defaults to the user's home directory. A subfolder named `job-<AWS_BATCH_JOB_ID>` is created in the parent working directory. You can use either this parameter or the `--working-dir` parameter.

-if *INPUT_FILE*, --input-file *INPUT_FILE*

Specifies the file to be transferred to the compute instances, in the job's working directory. You can specify multiple input file parameters.

-p *VCPUS*, --vcpus *VCPUS*

Specifies the number of vCPUs to reserve for the container. When used together with `-nodes`, it identifies the number of vCPUs per node.

Default: 1

-m *MEMORY*, --memory *MEMORY*

Specifies the hard limit of memory (in MiB) to provide for the job. If your job attempts to exceed the memory limit specified here, the job is killed.

Default: 128

-e *ENV*, --env *ENV*

Specifies a comma-separated list of environment variable names to export to the job environment. To export all environment variables, specify 'all'. Note that a list of 'all' environment variables will not include those listed in the `-env-blacklist` parameter, or variables starting with the `PCLUSTER_*` or `AWS_*` prefix.

-eb *ENV_BLACKLIST*, --env-blacklist *ENV_BLACKLIST*

Specifies a comma-separated list of environment variable names to **not** export to the job environment. By default, `HOME`, `PWD`, `USER`, `PATH`, `LD_LIBRARY_PATH`, `TERM`, and `TERMCAP` are not exported.

-r *RETRY_ATTEMPTS*, --retry-attempts *RETRY_ATTEMPTS*

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

Default: 1

-t *TIMEOUT*, --timeout *TIMEOUT*

Specifies the time duration in seconds (measured from the job attempt's startedAt timestamp) after which AWS Batch terminates your job if it has not finished. The timeout value must be at least 60 seconds.

-n *NODES*, --nodes *NODES*

Specifies the number of nodes to reserve for the job. Specify a value for this parameter to enable multi-nNode parallel submission.

-a *ARRAY_SIZE*, --array-size *ARRAY_SIZE*

Indicates the size of the array. You can specify a value between 2 and 10,000. If you specify array properties for a job, it becomes an array job.

-d *DEPENDS_ON*, --depends-on *DEPENDS_ON*

Specifies a semicolon-separated list of dependencies for a job. A job can depend upon a maximum of 20 jobs. You can specify a SEQUENTIAL type dependency without specifying a job ID for array jobs. A sequential dependency allows each child array job to complete sequentially, starting at index 0. You can also specify an N_TO_N type dependency with a job ID for array jobs. An N_TO_N dependency means that each index child of this job must wait for the corresponding index child of each dependency to complete before it can begin. The syntax for this parameter is "jobId=*<string>*,type=<string>;...".

awsbstat

Shows the jobs that are submitted in the cluster's job queue.

```
awsbstat [-h] [-c CLUSTER] [-s STATUS] [-e] [-d] [job_ids [job_ids ...]]
```

Positional Arguments

job_ids

Specifies the space-separated list of job IDs to show in the output. If the job is a job array, all of the children are displayed. If a single job is requested, it is shown in a detailed version.

Named Arguments

-c *CLUSTER*, --cluster *CLUSTER*

Indicates the cluster to use.

-s *STATUS*, --status *STATUS*

Specifies a comma-separated list of job statuses to include. The default job status is “active.”. Accepted values are: SUBMITTED, PENDING, RUNNABLE, STARTING, RUNNING, SUCCEEDED, FAILED, and ALL.

Default: “SUBMITTED,PENDING,RUNNABLE,STARTING,RUNNING”

-e, --expand-children

Expands jobs with children (both array and multi-node parallel).

Default: False

-d, --details

Shows jobs details.

Default: False

awsbout

Shows the output of a given job.

```
awsbout [ - h ] [ - c CLUSTER ] [ - hd HEAD ] [ - t TAIL ] [ - s ] [ - sp STREAM_PERIOD
 ] job_id
```

Positional Arguments

job_id

Specifies the job ID.

Named Arguments

-c *CLUSTER*, --cluster *CLUSTER*

Indicates the cluster to use.

-hd *HEAD*, --head *HEAD*

Gets the first *HEAD* lines of the job output.

-t *TAIL*, --tail *TAIL*

Gets the last <tail> lines of the job output.

-s, --stream

Gets the job output, and then waits for additional output to be produced. This argument can be used together with –tail to start from the latest <tail> lines of the job output.

Default: False

-sp *STREAM_PERIOD*, --stream-period *STREAM_PERIOD*

Sets the streaming period.

Default: 5

awsbkill

Cancels or terminates jobs submitted in the cluster.

```
awsbkill [ - h ] [ - c CLUSTER ] [ - r REASON ] job_ids [ job_ids ... ]
```

Positional Arguments

job_ids

Specifies the space-separated list of job IDs to cancel or terminate.

Named Arguments

-c *CLUSTER*, --cluster *CLUSTER*

Indicates the name of the cluster to use.

-r *REASON*, --reason *REASON*

Indicates the message to attach to a job, explaining the reason for canceling it.

Default: “Terminated by the user”

awsbqueues

Shows the job queue that is associated with the cluster.

```
awsbqueues [ - h ] [ - c CLUSTER ] [ - d ] [ job_queues [ job_queues ... ]]
```

Positional Arguments

job_queues

Specifies the space-separated list of queue names to show. If a single queue is requested, it is shown in a detailed version.

Named Arguments

-c *CLUSTER*, --cluster *CLUSTER*

Specifies the name of the cluster to use.

-d, --details

Indicates whether to show the details of the queues.

Default: False

awsbhosts

Shows the hosts that belong to the cluster’s compute environment.

```
awsbhosts [ - h ] [ - c CLUSTER ] [ - d ] [ instance_ids [ instance_ids ... ]]
```

Positional Arguments

instance_ids

Specifies a space-separated list of instances IDs. If a single instance is requested, it is shown in a detailed version.

Named Arguments

-c *CLUSTER*, --cluster *CLUSTER*

Specifies the name of the cluster to use.

-d, --details

Indicates whether to show the details of the hosts.

Default: False

Elastic Fabric Adapter

Elastic Fabric Adapter (EFA) is a network device that has OS-bypass capabilities for low-latency network communications with other instances on the same subnet. EFA is exposed by using libfabric, and can be used by applications using the Messaging Passing Interface (MPI). To use EFA with AWS ParallelCluster, add the line `enable_efa = compute` to the [cluster] section (p. 57). EFA is supported by specific instance types (the section called "compute_instance_type" (p. 60) is `c5n.18xlarge`, `c5n.metal`, `i3en.24xlarge`, or `p3dn.24xlarge`) on specific operating systems (the section called "base_os" (p. 59) is `alinux`, `centos7`, `ubuntu1604`, or `ubuntu1804`). For more information about the `enable_efa` setting, see the section called "enable_efa" (p. 62). A cluster placement group should be used to minimize latencies between instances. See the section called "placement" (p. 64) and the section called "placement_group" (p. 65).

For more information, see Elastic Fabric Adapter in the *Amazon EC2 User Guide for Linux Instances* and Scale HPC Workloads with Elastic Fabric Adapter and AWS ParallelCluster in the *AWS Open Source Blog*.

Enable Intel MPI

Intel MPI is available on the AWS ParallelCluster AMIs for `alinux`, `centos7`, and `ubuntu1604` the section called "base_os" (p. 59). Open MPI is placed on the path by default. To enable Intel MPI instead of Open MPI, the Intel MPI module must be loaded. The exact name of the module changes with every update. To see which modules are available, run `module avail`.

```
$ module avail

---------------------------------------------- /usr/share/Modules/modulefiles
 ----------------------------------------------
dot                 module-git          modules             use.own         openmpi/3.1.4
intelmpi/2019.4.243 module-info         null
```

To load a module, run `module load` *modulename*. You can add this to the script used to run `mpirun`.

```
$ module load intelmpi
```

To see what modules are loaded, run `module list`.

```
$ module list
Currently Loaded Modulefiles:
  1) intelmpi/2019.4.243
```

To verify that Intel MPI is enabled, run `mpirun --version`.

```
$ mpirun --version
Intel(R) MPI Library for Linux* OS, Version 2019 Update 4 Build 20190430 (id: cbdd16069)
Copyright 2003-2019, Intel Corporation.
```

After the Intel MPI module has been loaded, multiple paths are changed to use the Intel MPI tools. To run code that was compiled by the Intel MPI tools, the Intel MPI module must be loaded first.

Note
Prior to AWS ParallelCluster 2.5.0, Intel MPI is not available on the AWS ParallelCluster AMIs in China (Beijing) and China (Ningxia).

Intel HPC Platform Specification

AWS ParallelCluster is compliant with the Intel HPC Platform Specification. Intel's HPC Platform Specification provides a set of compute, fabric, memory, storage, and software requirements to ensure a high standard of quality and compatibility with HPC workloads.

In order to meet Intel's HPC Platform Specification, these requirements must be met.

- The OS must be Centos 7 (the section called "base_os" (p. 59) = centos7).
- The instance type for the compute nodes must have an Intel CPU and at least 64 GB of memory. For the c5 family of instance types, this means the instance type must at least a c5.9xlarge (the section called "compute_instance_type" (p. 60) = c5.9xlarge).
- The master node must have at least 200 GB of storage.
- The End User License Agreement for Intel Parallel Studio must be accepted (the section called "enable_intel_hpc_platform" (p. 62) = true).
- Each compute node must have at least 80 GB of storage (the section called "compute_root_volume_size" (p. 60) = 80).

The storage can be local or network (NFS shared from the master node, EBS or Amazon FSx for Lustre, and it can be shared.

Connect to the master instance through NICE DCV

NICE DCV is a remote visualization technology that enables users to securely connect to graphic-intensive 3D applications hosted on a remote high-performance server. For more information, see NICE DCV.

The NICE DCV software is automatically installed on the master instance when using the section called "base_os" (p. 59) = centos7.

To enable NICE DCV on the master instance, the section called "dcv_settings" (p. 60) must contain the name of a [dcv] section (p. 69) that has the section called "enable" (p. 69) = master and the section called "base_os" (p. 59) must be set to centos7.

```
[cluster custom-cluster]
...
dcv_settings = custom-dcv
...
[dcv custom-dcv]
enable = master
```

For more information about NICE DCV configuration parameters, see the section called "dcv_settings" (p. 60). To connect to the NICE DCV session, use the the section called "pcluster dcv" (p. 21) command.

Note
Support for NICE DCV was added in AWS ParallelCluster 2.5.0.

NICE DCV HTTPS Certificate

NICE DCV automatically generates a self-signed certificate to secure traffic between the NICE DCV client and NICE DCV server.

To replace the default self-signed NICE DCV certificate with another certificate, first connect to the master instance. Then, copy both the certificate and key to the /etc/dcv folder before running the the section called "pcluster dcv" (p. 21) command.

For more information, see Changing the TLS Certificate in the *NICE DCV Administrator Guide*.

Licensing NICE DCV

The NICE DCV server does not require a license server when running on Amazon EC2 instances. However, the NICE DCV server must periodically connect to an Amazon S3 bucket to determine whether a valid license is available.

AWS ParallelCluster automatically adds the required permissions to the the section called "ParallelClusterInstancePolicy" (p. 36). When using a custom IAM Instance Policy, use the permissions described in NICE DCV on Amazon EC2 in the *NICE DCV Administrator Guide*.

Configuration

Topics

By default, AWS ParallelCluster uses the file `~/.parallelcluster/config` for all configuration parameters.

An example configuration file is installed with AWS ParallelCluster in the Python directory at `site-packages/aws-parallelcluster/examples/config`. The example configuration file is also available on GitHub, at https://github.com/aws/aws-parallelcluster/blob/release/cli/pcluster/examples/config.

Layout

An AWS ParallelCluster configuration is defined in multiple sections.

The following sections are required: `[global] section (p. 55)` and `[aws] section (p. 56)`.

You also must include at least one `[cluster] section (p. 57)` and one `[vpc] section (p. 79)`.

A section starts with the section name in brackets, followed by parameters and configuration.

```
[global]
cluster_template = default
update_check = true
sanity_check = true
```

[global] Section

Topics

Specifies global configuration options related to `pcluster`.

```
[global]
```

cluster_template

Defines the name of the cluster section that is used by default for the cluster.

See Cluster Definition (p. 57).

For example, the following setting specifies that the section that starts `[cluster default]` is used by default.

```
cluster_template = default
```

update_check

Checks for updates to `pcluster`.

```
update_check = true
```

sanity_check

Attempts to validate the existence of the resources that are defined in the cluster parameters.

The default value is `true`.

```
sanity_check = true
```

Note
Prior to AWS ParallelCluster 2.5.0, `sanity_check` defaults to `false`.

[aws] Section

Specifies AWS Region information.

These settings apply to all clusters and they are required.

To store credentials, you can use the environment, IAM roles for Amazon EC2, or the AWS CLI, rather than saving credentials into the AWS ParallelCluster config file.

```
[aws]
# Defaults to us-east-1 if not defined in environment or below
aws_region_name = #region
```

[aliases] Section

Specifies aliases, and enables you to customize the `ssh` command.

Note the following default settings:

- *CFN_USER* is set to the default user name for the OS
- *MASTER_IP* is set to the IP address of the master instance
- *ARGS* is set to whatever arguments the user provides after *pcluster ssh cluster_name*

```
[aliases]
# This is the aliases section, you can configure
# ssh alias here
ssh = ssh {CFN_USER}@{MASTER_IP} {ARGS}
```

[cluster] Section

Topics

Defines one or more clusters for different job types or workloads.

Each cluster can have its own configuration.

The format is `[cluster <clustername>]`. The [cluster] section (p. 57) named by the the section called "cluster_template" (p. 56) setting in the [global] section (p. 55) is used.

```
[cluster default]
```

additional_cfn_template

Defines an additional AWS CloudFormation template to launch along with the cluster. This additional template is used for the creation of resources that exist outside of the cluster but are part of the cluster's lifecycle.

When set to a value other than `NONE`, it must be an HTTP URL to a public template, with all parameters provided.

The default value is `NONE`.

```
additional_cfn_template = NONE
```

additional_iam_policies

Specifies a comma-separated list of Amazon Resource Names (ARNs) of IAM policies for Amazon EC2. This list is attached to the root role used in the cluster, in addition to the permissions required by AWS ParallelCluster. An IAM policy name and its ARN are different. Names cannot be used as an argument to `additional_iam_policies`. `additional_iam_policies` should be used instead of the `ec2_iam_role`. This is because `additional_iam_policies` are added to the permissions that AWS ParallelCluster requires, and the `ec2_iam_role` must include all permissions required. The permissions required often change from release to release as features are added.

The default value is `NONE`.

```
additional_iam_policies = arn:aws:iam::aws:policy/AdministratorAccess
```

Note
Support for `additional_iam_policies` was added in AWS ParallelCluster 2.5.0.

base_os

Specifies which OS type is used in the cluster.

Available options are:

- alinux
- centos6
- centos7
- ubuntu1604
- ubuntu1804

Note
Support for ubuntu1804 was added and support for ubuntu1404 was removed in AWS ParallelCluster 2.5.0.

Supported operating systems by Region are listed in the following table. Note that "commercial" entails all other supported Regions including us-east-1, us-west-2, and so on.

Partition (Regions)	alinux	centos6	centos7	ubuntu1604	ubuntu1804
Commercial (All Regions not mentioned below)	True	True	True	True	True
AWS GovCloud (US-East) (us-gov-east-1)	True	False	False	True	True
AWS GovCloud (US-West) (us-gov-west-1)	True	False	False	True	True
China (Beijing) (cn-north-1)	True	False	False	True	True
China (Ningxia) (cn-northwest-1)	True	False	False	True	True

Note: The base_os parameter also determines the user name that is used to log into the cluster.

- centos6 and centos7: centos
- ubuntu1604, and ubuntu1804: ubuntu
- alinux: ec2-user

The default value is alinux.

```
base_os = alinux
```

cluster_type

Defines the type of cluster to launch.

Valid options are: ondemand, and spot.

The default value is ondemand.

For more information about Spot Instances, see the section called "Working with Spot Instances" (p. 34).

```
cluster_type = ondemand
```

compute_instance_type

Defines the Amazon EC2 instance type that is used for the cluster compute nodes.

If you are using the `awsbatch` scheduler, see the Compute Environments creation in the AWS Batch UI for a list of supported instance types.

Defaults to `t2.micro`, `optimal` when the scheduler is `awsbatch`.

```
compute_instance_type = t2.micro
```

A1 instances are not supported.

compute_root_volume_size

Specifies the ComputeFleet root volume size in GB. The AMI must support growroot.

The default value is `25`.

Note
Prior to AWS ParallelCluster. 2.5.0 the default was 20.

```
compute_root_volume_size = 20
```

custom_ami

Specifies the ID of a custom AMI to use instead of the default published AMIs.

The default value is `NONE`.

```
custom_ami = NONE
```

dcv_settings

Identifies the `[dcv]` section with the NICE DCV configuration.

For more information, see the [dcv] section (p. 69).

For example, the following setting specifies that the section that starts `[dcv custom-dcv]` is used for the NICE DCV configuration.

```
dcv_settings = custom-dcv
```

Note
Support for `dcv_settings` was added in AWS ParallelCluster 2.5.0.

desired_vcpus

Specifies the desired number of vCPUs in the compute environment. Used only if the scheduler is `awsbatch`.

The default value is 4.

```
desired_vcpus = 4
```

disable_hyperthreading

Disables hyperthreading on the master and compute nodes. Not all instance types can disable hyperthreading. For a list of instance types that support disabling hyperthreading, see CPU Cores and Threads Per CPU Core Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.

```
disable_hyperthreading = true
```

Note
Support for disable_hyperthreading was added in AWS ParallelCluster 2.5.0.

ebs_settings

Identifies the [ebs] sections with the Amazon EBS volumes that are mounted on the master instance. When using multiple Amazon EBS volumes, enter these parameters as a comma-separated list.

Up to five (5) additional Amazon EBS volumes are supported.

For more information, see the [ebs] section (p. 69).

For example, the following setting specifies that the sections that start [ebs custom1] and [ebs custom2] are used for the Amazon EBS volumes.

```
ebs_settings = custom1, custom2
```

ec2_iam_role

Defines the name of an existing IAM role for Amazon EC2 that is attached to all instances in the cluster. An IAM role name and its Amazon Resource Name (ARN) are different. ARNs cannot be used as an argument to ec2_iam_role. If this option is specified, the additional_iam_policies setting is ignored. AWS recommends using additional_iam_policies rather than the ec2_iam_role, because features added to AWS ParallelCluster often require new permissions.

The default value is NONE.

```
ec2_iam_role = NONE
```

efs_settings

Specifies settings related to the Amazon EFS filesystem.

For more information, see the [efs] section (p. 72).

For example, the following setting specifies that the section that starts [efs customfs] is used for the Amazon EFS filesystem configuration.

```
efs_settings = customfs
```

enable_efa

If present, specifies that Elastic Fabric Adapter (EFA) is enabled for the compute nodes. EFA is supported by specific instance types (`c5n.18xlarge`, `c5n.metal`, `i3en.24xlarge`, `p3dn.24xlarge`). For more information, see the section called "Elastic Fabric Adapter" (p. 52).

```
enable_efa = compute
```

enable_intel_hpc_platform

If present, indicates that the End User License Agreement for Intel Parallel Studio is accepted. This will cause Intel Parallel Studio to be installed on the master node and shared with the compute nodes. This adds several minutes to the time it takes the master node to bootstrap.

```
enable_intel_hpc_platform = true
```

Note
Support for `enable_intel_hpc_platform` was added in AWS ParallelCluster 2.5.0.

encrypted_ephemeral

Encrypts the ephemeral instance store volumes with non-recoverable in-memory keys, using LUKS (Linux Unified Key Setup).

For more information, see https://gitlab.com/cryptsetup/cryptsetup/blob/master/README.md.

The default value is `false`.

```
encrypted_ephemeral = false
```

ephemeral_dir

Defines the path where instance store volumes are mounted, if they are used.

The default value is `/scratch`.

```
ephemeral_dir = /scratch
```

extra_json

Defines the extra JSON that is merged into the `dna.json` that is used by Chef.

The default value is `{}`.

```
extra_json = {}
```

fsx_settings

Specifies the section that defines the Amazon FSx for Lustre configuration.

For more information, see the [fsx] section (p. 74).

```
fsx_settings = fs
```

For example, the following setting specifies that the section that starts `[fsx fs]` is used for the Amazon FSx for Lustre configuration.

```
fsx_settings = fs
```

initial_queue_size

Sets the initial number of Amazon EC2 instances to launch as compute nodes in the cluster.

This setting is applicable only for traditional schedulers (SGE, Slurm, and Torque).

If the scheduler is `awsbatch`, use the section called "min_vcpus" (p. 64) instead.

Defaults to `2`.

```
initial_queue_size = 2
```

key_name

Names an existing Amazon EC2 key pair with which to enable SSH access to the instances.

```
key_name = mykey
```

maintain_initial_size

Maintains the initial size of the Auto Scaling group for traditional schedulers (SGE, Slurm, and Torque).

If the scheduler is `awsbatch`, use the section called "desired_vcpus" (p. 60) instead.

This setting is a Boolean flag. If set to `true`, the Auto Scaling group never has fewer members than the value of the section called "initial_queue_size" (p. 63). The cluster can still scale up to the value of the section called "max_queue_size" (p. 64). If `cluster_type = spot` then the Auto Scaling group can have instances interrupted and the size can drop below `initial_queue_size`.

If set to `false`, the Auto Scaling group can scale down to zero (0) members to prevent resources from sitting idle when they are not needed.

Defaults to `false`.

```
maintain_initial_size = false
```

master_instance_type

Defines the Amazon EC2 instance type that is used for the master node.

Defaults to `t2.micro`.

```
master_instance_type = t2.micro
```

A1 instances are not supported.

master_root_volume_size

Specifies the MasterServer root volume size in GB. The AMI must support `growroot`.

The default value is `25`.

Note
Prior to AWS ParallelCluster. 2.5.0 the default was 20.

```
master_root_volume_size = 20
```

max_queue_size

Sets the maximum number of Amazon EC2 instances that can be launched in the cluster.

This setting is applicable only for traditional schedulers (SGE, Slurm, and Torque).

If the scheduler is `awsbatch`, use the section called "max_vcpus" (p. 64) instead.

Defaults to `10`.

```
max_queue_size = 10
```

max_vcpus

Specifies the maximum number of vCPUs in the compute environment. Used only if the scheduler is `awsbatch`.

The default value is `20`.

```
max_vcpus = 20
```

min_vcpus

Maintains the initial size of the Auto Scaling group for the `awsbatch` scheduler.

If the scheduler is SGE, Slurm, or Torque, use the section called "maintain_initial_size" (p. 63) instead.

The compute environment never has fewer members than the value of `min_vcpus`.

Defaults to `0`.

```
min_vcpus = 0
```

placement

Defines the cluster placement group logic, enabling either the whole cluster or only the compute instances to use the cluster placement group.

Valid options are `cluster` or `compute`.

This parameter is not used when the scheduler is `awsbatch`.

The default value is `compute`.

```
placement = compute
```

placement_group

Defines the cluster placement group.

Valid options are:

- NONE
- DYNAMIC
- An existing Amazon EC2 cluster placement group name

When set to DYNAMIC, a unique placement group is created and deleted as part of the cluster stack.

This parameter is not used when the scheduler is awsbatch.

For more information about placement groups, see Placement Groups in the *Amazon EC2 User Guide for Linux Instances*.

The default value is NONE.

Not all instance types support cluster placement groups. For example, the default instance type of t2.micro does not support cluster placement groups. For information about the list of instance types that support cluster placement groups, see Cluster Placement Group Rules and Limitations in the *Amazon EC2 User Guide for Linux Instances*. See the section called "Placement Groups and Instance Launch Issues" (p. 108) for tips when working with placement groups.

```
placement_group = NONE
```

post_install

Specifies the URL of a postinstall script that is executed after all of the boot_as_* scripts are run.

When using awsbatch as the scheduler, the postinstall script is executed only on the master node.

The parameter format can be either http://*hostname/path/to/script.sh* or s3://*bucketname/path/to/script.sh*.

The default value is NONE.

```
post_install = NONE
```

post_install_args

Specifies a quoted list of arguments to pass to the postinstall script.

The default value is NONE.

```
post_install_args = "NONE"
```

pre_install

Specifies the URL of a preinstall script that is executed before any of the boot_as_* scripts are run.

When using `awsbatch` as the scheduler, the preinstall script is executed only on the master node.

The parameter format can be either `http://`*`hostname/path/to/script.sh`* or `s3://`*`bucketname/path/to/script.sh`*.

The default value is `NONE`.

```
pre_install = NONE
```

pre_install_args

Specifies a quoted list of arguments to pass to the preinstall script.

The default value is `NONE`.

```
pre_install_args = "NONE"
```

proxy_server

Defines an HTTP or HTTPS proxy server, typically `http://`*`x.x.x.x:8080`*.

The default value is `NONE`.

```
proxy_server = NONE
```

raid_settings

Identifies the `[raid]` section with the Amazon EBS volume RAID configuration.

For more information, see the [raid] section (p. 76).

For example, the following setting specifies that the section that starts `[raid rs]` be used for the Auto Scaling configuration.

```
raid_settings = rs
```

s3_read_resource

Specifies an Amazon S3 resource to which AWS ParallelCluster nodes are granted read-only access.

For example, `arn:aws:s3:::`*`my_corporate_bucket`*`/*` provides read-only access to all objects in the *`my_corporate_bucket`* bucket.

See working with Amazon S3 (p. 33) for details on format.

The default value is `NONE`.

```
s3_read_resource = NONE
```

s3_read_write_resource

Specifies an Amazon S3 resource to which AWS ParallelCluster nodes are granted read/write access.

For example, `arn:aws:s3:::my_corporate_bucket/Development/*` provides read/write access to all objects in the Development folder of the *my_corporate_bucket* bucket.

See working with Amazon S3 (p. 33) for details on format.

The default value is `NONE`.

```
s3_read_write_resource = NONE
```

scaling_settings

Identifies the `[scaling]` section with the Auto Scaling configuration.

For more information, see the [scaling] section (p. 79).

For example, the following setting specifies that the section that starts `[scaling custom]` is used for the Auto Scaling configuration.

```
scaling_settings = custom
```

scheduler

Defines the cluster scheduler.

Valid options are:

- `sge`
- `torque`
- `slurm`
- `awsbatch`

For more information about the `awsbatch` scheduler, see networking setup (p. 30).

The default value is `sge`.

```
scheduler = sge
```

shared_dir

Defines the path where the shared Amazon EBS volume is mounted.

Do not use this option with multiple Amazon EBS volumes. Instead, provide `shared_dir` values under each Amazon EBS [ebs] section (p. 69).

See the Amazon EBS Section (p. 69) for details on working with multiple Amazon EBS volumes.

The default value is `/shared`.

The following example shows a shared Amazon EBS volume mounted at `/myshared`.

```
shared_dir = myshared
```

spot_bid_percentage

Optionally sets the on-demand bid percentage used to calculate the maximum Spot price for the ComputeFleet, when `awsbatch` is the scheduler.

If unspecified, the current spot market price is selected, capped at the On-Demand price.

```
spot_bid_percentage = 85
```

spot_price

Optionally sets the maximum Spot price for the ComputeFleet on traditional schedulers (SGE, Slurm, and Torque). Used only when the `cluster_type` is set to `spot`. If you do not specify a value, you are charged the Spot price, capped at the On-Demand price.

If the scheduler is `awsbatch`, use spot_bid_percentage (p. 68) instead.

For assistance finding a bid price that meets your needs, see the Spot Bid Advisor.

```
spot_price = 1.50
```

tags

Defines tags to be used by AWS CloudFormation.

If command line tags are specified via *--tags*, they are merged with config tags.

Command line tags overwrite config tags that have the same key.

Tags are JSON formatted. Do not use quotes outside of the curly braces.

For more information, see AWS CloudFormation Resource Tags Type in the *AWS CloudFormation User Guide*.

```
tags = {"key" : "value", "key2" : "value2"}
```

template_url

Defines the path to the AWS CloudFormation template that is used to create the cluster.

Updates use the template that was originally used to create the stack.

Defaults to `https://<aws_region_name>-aws-parallelcluster.s3.amazonaws.com/templates/aws-parallelcluster-<version>.cfn.json`.

```
template_url = https://us-east-1-aws-parallelcluster.s3.amazonaws.com/templates/aws-parallelcluster.cfn.json
```

vpc_settings

Identifies the `[vpc]` section with the Amazon VPC configuration where the cluster is deployed.

For more information, see the [vpc] section (p. 79).

For example, the following setting specifies that the section that starts `[vpc public]` is used for the Amazon VPC configuration.

```
vpc_settings = public
```

[dcv] Section

Defines configuration settings for the NICE DCV server running on the master instance.

To create and configure a NICE DCV server, specify a the section called "dcv_settings" (p. 60) with the name of your section, with the section called "enable" (p. 69) set to `master`, and a the section called "base_os" (p. 59) set to `centos7`.

```
[dcv custom-dcv]
enable = master
port = 8443
access_from = 0.0.0.0/0
```

Note
Support for the [dcv] section (p. 69) was added in AWS ParallelCluster 2.5.0.

enable

(Required) Indicates whether NICE DCV is enabled on the master node. To enable NICE DCV on the master node, set `enable` to `master`.

The default value is `NONE`.

```
enable = master
```

port

(Optional) Specifies the port for NICE DCV.

The default value is `8443`.

```
port = 8443
```

access_from

(Optional) Specifies the CIDR for connections to NICE DCV.

The default value is `0.0.0.0/0`.

```
access_from = 0.0.0.0/0
```

[ebs] Section

Topics

Defines Amazon EBS volume configuration settings for volumes that are mounted on the master instance and shared via NFS to the compute nodes.

The format is `[ebs <ebsname>]`.

```
[ebs custom1]
shared_dir = vol1
ebs_snapshot_id = snap-xxxxx
volume_type = io1
volume_iops = 200
...

[ebs custom2]
shared_dir = vol2
...

...
```

shared_dir

Specifies the path where the shared Amazon EBS volume is mounted.

This parameter is required when using multiple Amazon EBS volumes.

When using one (1) Amazon EBS volume, this option overwrites the the section called "shared_dir" (p. 67) that is specified under the [cluster] section (p. 57). In the following example, the volume mounts to `/vol1`.

```
shared_dir = vol1
```

ebs_snapshot_id

Defines the Amazon EBS snapshot Id, if you are using a snapshot as the source for the volume.

The default value is `NONE`.

```
ebs_snapshot_id = snap-xxxxx
```

volume_type

Specifies the Amazon EBS volume type of the volume that you want to launch.

Valid options are:

- gp2
- io1
- st1
- sc1

The default value is gp2.

```
volume_type = io1
```

volume_size

Specifies the size of the volume to be created, in GiB (if not using a snapshot).

The default value is 20.

```
volume_size = 20
```

volume_iops

Defines the number of IOPS for io1-type volumes.

```
volume_iops = 200
```

encrypted

Specifies whether the Amazon EBS volume is encrypted. Note: Do *not* use with snapshots.

The default value is false.

```
encrypted = false
```

ebs_kms_key_id

Specifies a custom AWS KMS key to use for encryption.

This parameter must be used together with encrypted = true. It also must have a custom ec2_iam_role.

For more information, see the section called "Disk Encryption with a Custom KMS Key" (p. 104).

```
ebs_kms_key_id = xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

ebs_volume_id

Defines the volume Id of an existing Amazon EBS volume to attach to the master instance.

The default value is NONE.

```
ebs_volume_id = vol-xxxxxx
```

[efs] Section

Topics

Defines configuration settings for the Amazon EFS that is mounted on the master and compute instances. For more information, see CreateFileSystem in the Amazon EFS documentation.

The format is `[efs <efsname>]`.

```
[efs customfs]
shared_dir = efs
encrypted = false
performance_mode = generalPurpose
```

shared_dir

Defines the Amazon EFS mount point on the master and compute nodes.

This parameter is required. The Amazon EFS section is used only if `shared_dir` is specified.

Do not use `NONE` or `/NONE` as the shared directory.

The following example mounts Amazon EFS at `/efs`.

```
shared_dir = efs
```

encrypted

Indicates whether the file system is encrypted.

The default value is `false`.

```
encrypted = false
```

performance_mode

Defines the performance mode of the file system.

Valid choices are:

- `generalPurpose`
- `maxIO`

Both values are case-sensitive.

We recommend the `generalPurpose` performance mode for most file systems.

File systems that use the `maxIO` performance mode can scale to higher levels of aggregate throughput and operations per second. However, there is a trade-off of slightly higher latencies for most file operations.

This parameter cannot be changed after the file system has been created.

The default value is `generalPurpose`.

```
performance_mode = generalPurpose
```

throughput_mode

Defines the throughput mode of the file system.

Valid options are:

- `bursting`
- `provisioned`

```
throughput_mode = provisioned
```

provisioned_throughput

Defines the provisioned throughput of the file system, measured in MiB/s.

If you use this parameter, you must set the section called “throughput_mode” (p. 73) to `provisioned`.

The limit on throughput is `1024` MiB/s. To request a limit increase, contact AWS Support.

The minimum value is `0.0` MiB/s

```
provisioned_throughput = 1024
```

efs_fs_id

Defines the Amazon EFS file system ID for an existing file system.

Specifying this option voids all other Amazon EFS options except for `shared_dir`.

If you set this option to `config_sanity`, it only supports file systems:

- That do not have a mount target in the stack's Availability Zone

  OR
- That do have an existing mount target in the stack's Availability Zone, with inbound and outbound NFS traffic allowed from `0.0.0.0/0`.

The sanity check for validating `efs_fs_id` requires the IAM role to have the following permissions:

- `elasticfilesystem:DescribeMountTargets`
- `elasticfilesystem:DescribeMountTargetSecurityGroups`

- ec2:DescribeSubnets
- ec2:DescribeSecurityGroups
- ec2:DescribeNetworkInterfaceAttribute

To avoid errors, you must add these permissions to your IAM role, or set sanity_check = false.

CAUTION: When you set a mount target with inbound and outbound NFS traffic allowed from 0.0.0.0/0, it exposes the file system to NFS mounting requests from anywhere in the mount target's Availability Zone. AWS recommends that you *not* create a mount target in the stack's Availability Zone, and instead let AWS handle this step. If you must have a mount target in the stack's Availability Zone, consider using a custom security group by providing a vpc_security_group_id option under the [vpc] section (p. 79). Then add that security group to the mount target, and turn off config sanity to create the cluster.

The default value is NONE.

```
efs_fs_id = fs-12345
```

[fsx] Section

Topics

Defines configuration settings for an attached Amazon FSx for Lustre file system. For more information about Amazon FSx for Lustre, see Amazon FSx CreateFileSystem.

Amazon FSx for Lustre is supported when the section called "base_os" (p. 59) is either centos7 or alinux.

When using Amazon Linux, the kernel must be >= 4.14.104-78.84.amzn1.x86_64. For detailed instructions, see Installing the Lustre Client in the *Amazon FSx for Lustre User Guide*.

Note
Amazon FSx for Lustre is not currently supported when using awsbatch as a scheduler.

If using an existing file system, it must be associated to a security group that allows inbound TCP traffic to port 988. Setting the source to 0.0.0.0/0 on a security group rule provides client access from all IP ranges within your VPC security group for the protocol and port range for that rule. To further limit access to your file systems we recommend using more restrictive sources for your security group rules, for example more specific CIDR ranges, IP addresses, or security group IDs. This is done automatically when not using the section called "vpc_security_group_id" (p. 81).

To use an existing Amazon FSx file system, specify the section called "fsx_fs_id" (p. 75).

The format is [fsx *<fsxname>*].

```
[fsx fs]
shared_dir = /fsx
fsx_fs_id = fs-073c3803dca3e28a6
```

To create and configure a new file system, use the following parameters:

```
[fsx fs]
shared_dir = /fsx
storage_capacity = 3600
imported_file_chunk_size = 1024
export_path = s3://bucket/folder
import_path = s3://bucket
weekly_maintenance_start_time = 1:00:00
```

shared_dir

(Required) Defines the mount point for the Amazon FSx for Lustre file system on the master and compute nodes.

Do not use `NONE` or `/NONE` as the shared directory.

The following example mounts the file system at `/fsx`.

```
shared_dir = /fsx
```

fsx_fs_id

(Optional) Attaches an existing Amazon FSx file system.

If this option is specified, only the the section called "`shared_dir`" (p. 75) and the section called "`fsx_fs_id`" (p. 75) settings in the [fsx] section (p. 74) are used and any other settings in the [fsx] section (p. 74) are ignored.

```
fsx_fs_id = fs-073c3803dca3e28a6
```

storage_capacity

(Optional) Specifies the storage capacity of the file system, in GiB.

The storage capacity has a minimum of 3,600 GiB and is provisioned in increments of 3,600 GiB.

The default value is `3600`.

```
storage_capacity = 3600
```

fsx_kms_key_id

(Optional) Specifies the ID of your AWS Key Management Service (AWS KMS) key.

This ID is used to encrypt the data in your file system at rest.

This must be used with a custom `ec2_iam_role`. For more information, see the section called "Disk Encryption with a Custom KMS Key" (p. 104).

.

```
fsx_kms_key_id = xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

imported_file_chunk_size

(Optional) Determines the stripe count and the maximum amount of data per file (in MiB) stored on a single physical disk, for files that are imported from a data repository (using `import_path`). The maximum number of disks that a single file can be striped across is limited by the total number of disks that make up the file system.

The chunk size default is `1024` (1 GiB), and it can go as high as 512,000 MiB (500 GiB). Amazon S3 objects have a maximum size of 5 TB.

```
imported_file_chunk_size = 1024
```

export_path

(Optional) Specifies the Amazon S3 path where the root of your file system is exported. The path **must** be in the same Amazon S3 bucket as the `import_path` parameter.

The default value is `s3://`*`import-bucket`*`/FSxLustre`*`[creation-timestamp]`*, where *`import-bucket`* is the bucket provided in the the section called “import_path” (p. 76) parameter.

```
export_path = s3://bucket/folder
```

import_path

(Optional) Specifies the S3 bucket to load data from into the file system. Also serves as the export bucket. For more information, see the section called “export_path” (p. 76).

Import occurs on cluster creation. For more information, see Importing Data from your Amazon S3 Bucket in the *Amazon FSx for Lustre User Guide*.

If a value is not provided, the file system is empty.

```
import_path =  s3://bucket
```

weekly_maintenance_start_time

(Optional) Specifies a preferred time to perform weekly maintenance, in the UTC time zone.

The format is [day of week]:[hour of day]:[minute of hour]. For example, Monday at Midnight is:

```
weekly_maintenance_start_time = 1:00:00
```

[raid] Section

Topics

- raid_type (p. 77)
- num_of_raid_volumes (p. 77)
- volume_type (p. 78)
- volume_size (p. 78)
- volume_iops (p. 78)
- encrypted (p. 78)
- ebs_kms_key_id (p. 78)

Defines configuration settings for a RAID array that is built from a number of identical Amazon EBS volumes. The RAID drive is mounted on the master node and is exported to compute nodes via NFS.

The format is `[raid <raidname>]`.

```
[raid rs]
shared_dir = raid
raid_type = 1
num_of_raid_volumes = 2
encrypted = true
```

shared_dir

Defines the mount point for the RAID array on the master and compute nodes.

The RAID drive is created only if this parameter is specified.

Do not use `NONE` or `/NONE` as the shared directory.

The following example mounts the array at `/raid`.

```
shared_dir = raid
```

raid_type

Defines the RAID type for the RAID array.

The RAID drive is created only if this parameter is specified.

Valid options are:

- `0`
- `1`

For more information on RAID types, see: RAID info in the *Amazon EC2 User Guide for Linux Instances*.

The following example creates a RAID `0` array:

```
raid_type = 0
```

num_of_raid_volumes

Defines the number of Amazon EBS volumes to assemble the RAID array from.

Minimum number of volumes = 2.

Maximum number of volumes = 5.

The default value is 2.

```
num_of_raid_volumes = 2
```

volume_type

Defines the type of volume to build.

Valid options are:

- gp2
- io1
- st1
- sc1

For more information, see Amazon EBS Volume Types in the *Amazon EC2 User Guide for Linux Instances*.

The default value is gp2.

```
volume_type = io1
```

volume_size

Defines the size of the volume to be created, in GiB.

The default value is 20.

```
volume_size = 20
```

volume_iops

Defines the number of IOPS for io1 type volumes.

```
volume_iops = 500
```

encrypted

Specifies whether the file system is encrypted.

The default value is false.

```
encrypted = false
```

ebs_kms_key_id

Specifies a custom AWS KMS key to use for encryption.

This parameter must be used together with `encrypted = true`, and it must have a custom `ec2_iam_role`.

For more information, see the section called "Disk Encryption with a Custom KMS Key" (p. 104).

```
ebs_kms_key_id = xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

[scaling] Section

Topics

Specifies settings that define how the compute nodes scale.

The format is `[scaling <scalingname>]`.

```
[scaling custom]
scaledown_idletime = 10
```

scaledown_idletime

Specifies the amount of time in minutes without a job, after which the compute node terminates.

This parameter is not used if `awsbatch` is the scheduler.

The default value is `10`.

```
scaledown_idletime = 10
```

[vpc] Section

Topics

Specifies Amazon VPC configuration settings.

The format is `[vpc <vpcname>]`.

```
[vpc public]
vpc_id = vpc-xxxxxx
```

```
master_subnet_id = subnet-xxxxxx
```

additional_sg

Provides an additional Amazon VPC security group Id for all instances.

The default value is `NONE`.

```
additional_sg = sg-xxxxxx
```

compute_subnet_cidr

Specifies a CIDR block. Use this parameter if you want AWS ParallelCluster to create a compute subnet.

```
compute_subnet_cidr = 10.0.100.0/24
```

compute_subnet_id

Specifies the ID of an existing subnet in which to provision the compute nodes.

If not specified, `compute_subnet_id` uses the value of `master_subnet_id`.

If the subnet is private, you must set up NAT for web access.

```
compute_subnet_id = subnet-xxxxxx
```

master_subnet_id

Specifies the ID of an existing subnet in which to provision the master node.

```
master_subnet_id = subnet-xxxxxx
```

ssh_from

Specifies a CIDR-formatted IP range to allow SSH access from.

This parameter is used only when AWS ParallelCluster creates the security group.

The default value is `0.0.0.0/0`.

```
ssh_from = 0.0.0.0/0
```

use_public_ips

Defines whether to assign public IP addresses to compute instances.

If set to `true`, an Elastic IP is associated to the master instance.

If set to `false`, the master instance has a public IP (or not) according to the value of the "Auto-assign Public IP" subnet configuration parameter.

For examples, see networking configuration (p. 27).

The default value is `true`.

```
use_public_ips = true
```

vpc_id

Specifies the ID of the Amazon VPC in which to provision the cluster.

```
vpc_id = vpc-xxxxxx
```

vpc_security_group_id

Specifies the use of an existing security group for all instances.

The default value is `NONE`.

```
vpc_security_group_id = sg-xxxxxx
```

Example

The following example launches a cluster with the `awsbatch` scheduler. It is set to pick the optimal instance type, based on your job resource needs.

The example configuration allows a maximum of 40 concurrent vCPUs, and scales down to zero when no jobs have run for 10 minutes.

```
[global]
update_check = true
sanity_check = true
cluster_template = awsbatch

[aws]
aws_region_name = [your_aws_region]

[cluster awsbatch]
scheduler = awsbatch
compute_instance_type = optimal # optional, defaults to optimal
min_vcpus = 0                   # optional, defaults to 0
desired_vcpus = 0               # optional, defaults to 4
max_vcpus = 40                  # optional, defaults to 20
base_os = alinux                # optional, defaults to alinux, controls the base_os of the
 master instance and the docker image for the compute fleet
key_name = [your_ec2_keypair]
vpc_settings = public

[vpc public]
master_subnet_id = [your_subnet]
vpc_id = [your_vpc]
```

How AWS ParallelCluster Works

AWS ParallelCluster was built not only as a way to manage clusters, but as a reference on how to use AWS services to build your HPC environment.

Topics

AWS ParallelCluster Processes

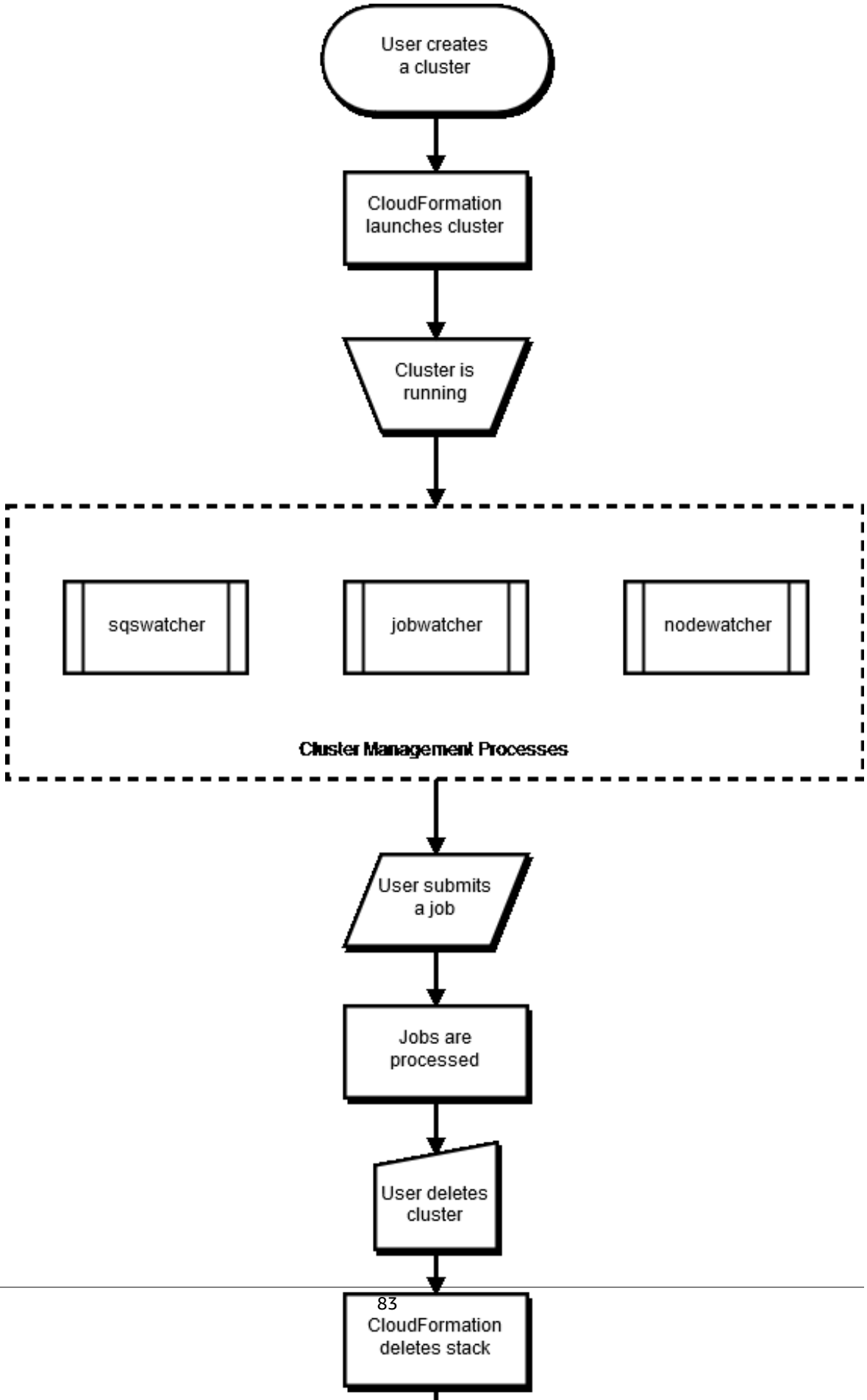
This section applies only to HPC clusters that are deployed with one of the supported traditional job schedulers (SGE, Slurm, or Torque). When used with these schedulers, AWS ParallelCluster manages the compute node provisioning and removal by interacting with both the Auto Scaling Group (ASG) and the underlying job scheduler.

For HPC clusters that are based on AWS Batch, AWS ParallelCluster relies on the capabilities provided by the AWS Batch for the compute node management.

Topics

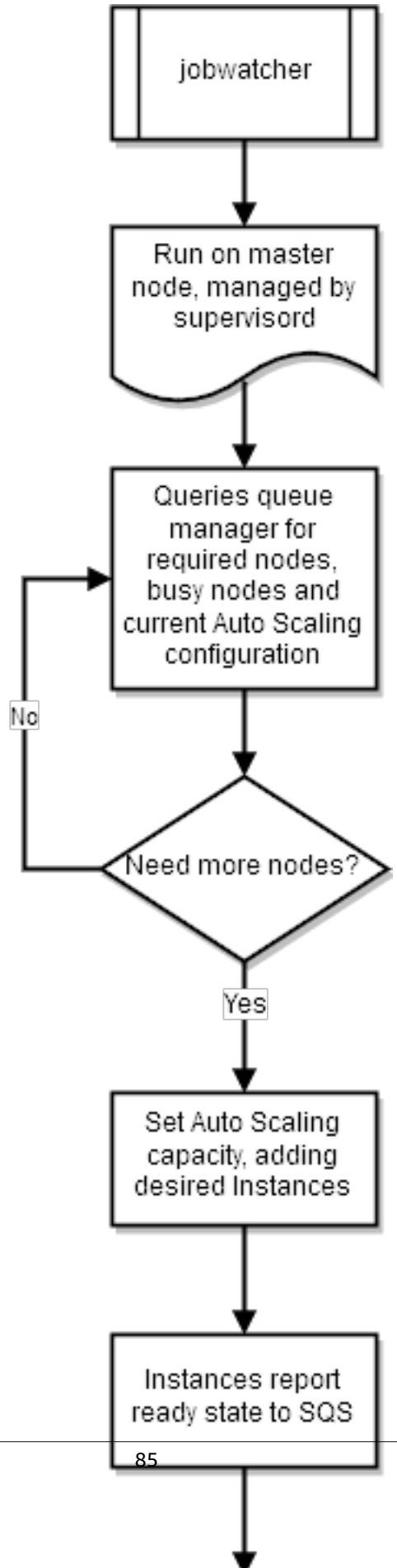
General Overview

A cluster's lifecycle begins after it is created by a user. Typically, a cluster is created from the Command Line Interface (CLI). After it's created, a cluster exists until it's deleted. AWS ParallelCluster daemons run on the cluster nodes, mainly to manage the HPC cluster elasticity. The following diagram shows a user workflow and the cluster lifecycle. The sections that follow describe the AWS ParallelCluster daemons that are used to manage the cluster.


User creates a cluster
CloudFormation launches cluster
Cluster is running
sqswatcher
jobwatcher
nodewatcher
Cluster Management Processes
User submits a job
Jobs are processed
User deletes cluster
CloudFormation deletes stack

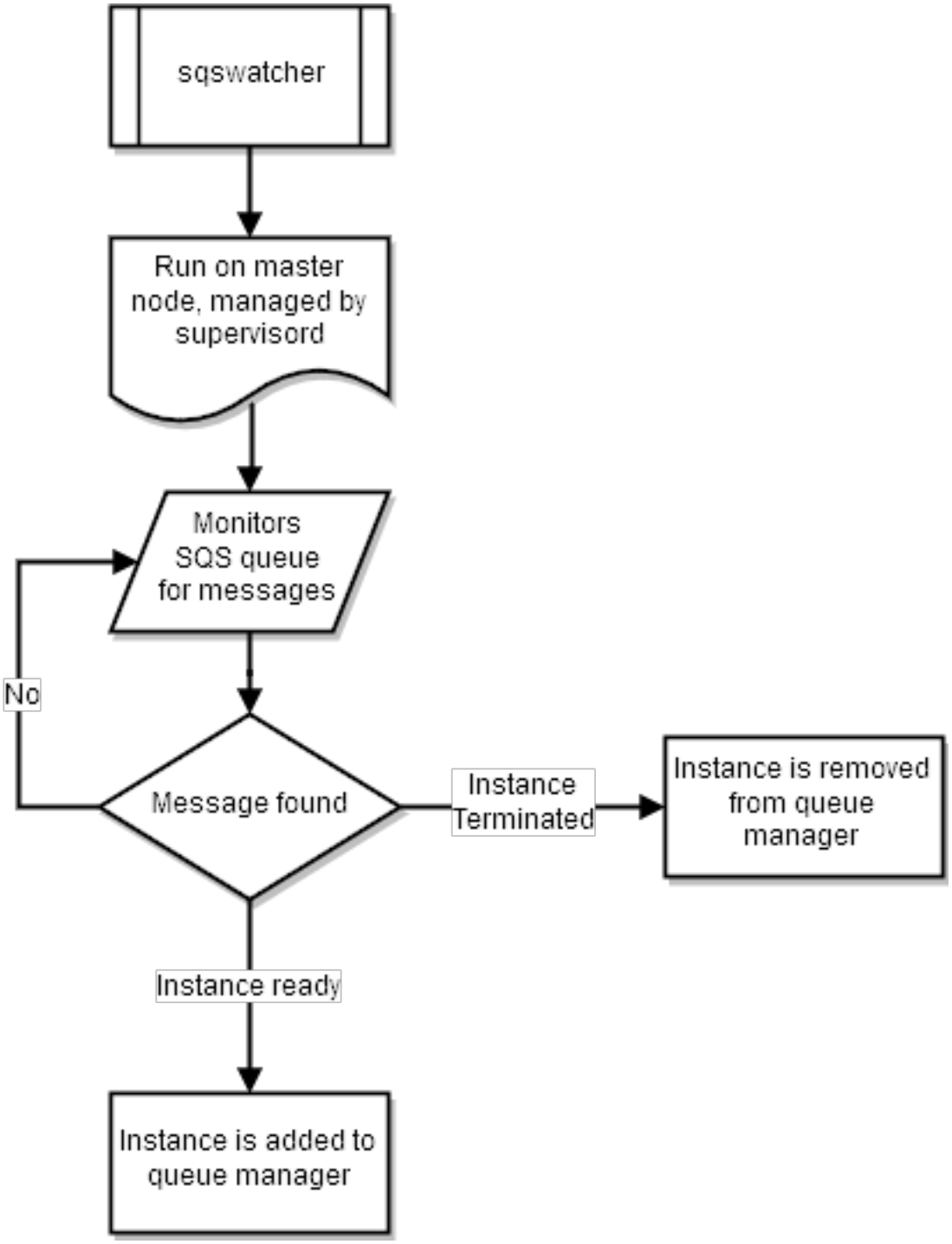
jobwatcher

When a cluster is running, a process owned by the root user monitors the configured scheduler (SGE, Slurm, or Torque) and each minute, it evaluates the queue in order to decide when to scale up.


jobwatcher
Run on master node, managed by supervisord
Queries queue manager for required nodes, busy nodes and current Auto Scaling configuration
No
Need more nodes?
Yes
Set Auto Scaling capacity, adding desired Instances
Instances report ready state to SQS

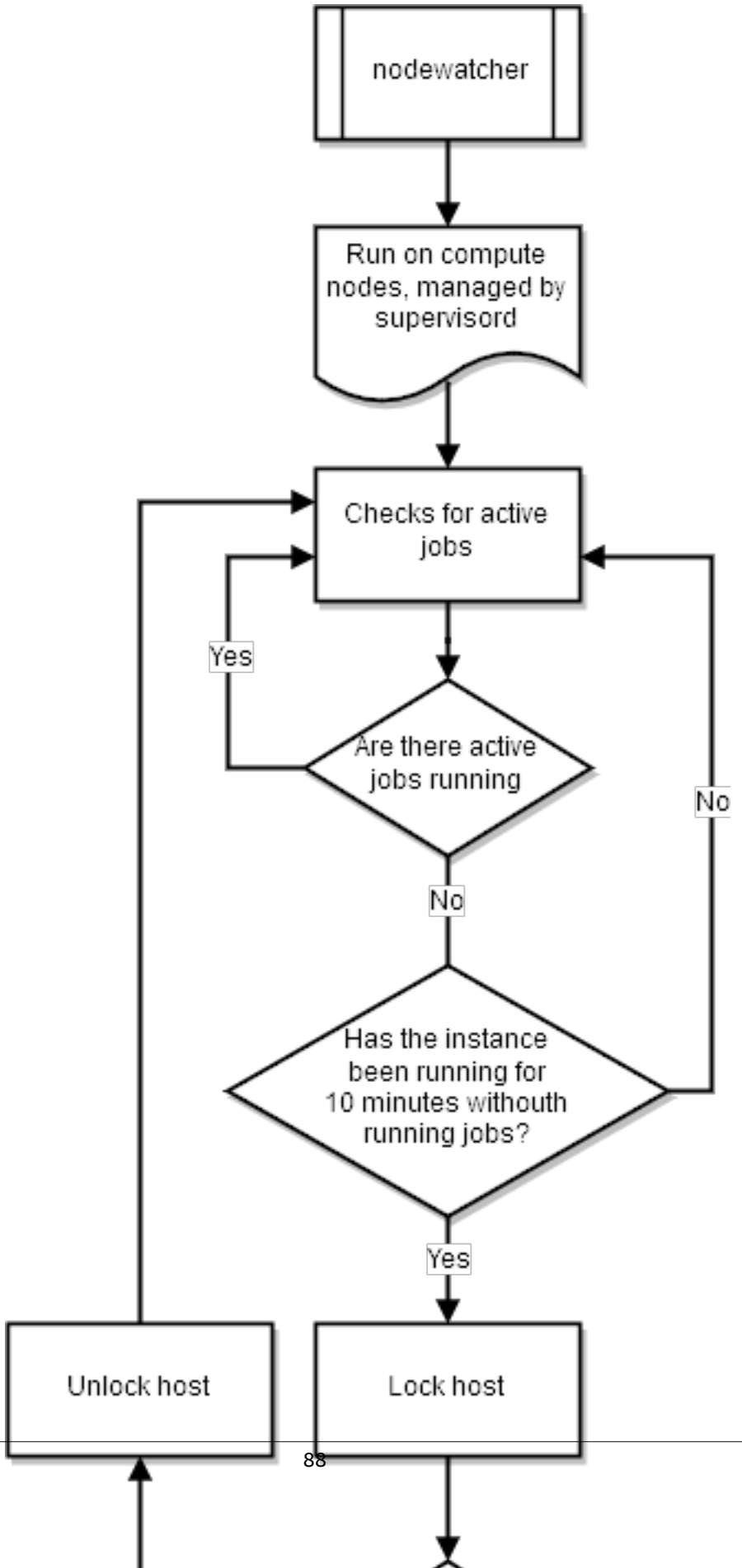
sqswatcher

The `sqswatcher` process monitors for Amazon SQS messages that are sent by Auto Scaling, to notify you of state changes within the cluster. When an instance comes online, it submits an "instance ready" message to Amazon SQS. This message is picked up by `sqs_watcher`, running on the master node. These messages are used to notify the queue manager when new instances come online or are terminated, so they can be added or removed from the queue.



nodewatcher

The `nodewatcher` process runs on each node in the compute fleet. After the `scaledown_idletime` period, as defined by the user, the instance is terminated.


nodewatcher
Run on compute nodes, managed by supervisord
Checks for active jobs
Yes
Are there active jobs running
No
No
Has the instance been running for 10 minutes withouth running jobs?
Yes
Unlock host
Lock host

AWS Services used in AWS ParallelCluster

The following Amazon Web Services (AWS) services are used in AWS ParallelCluster.

Topics

AWS Auto Scaling

AWS Auto Scaling is used to manage the ComputeFleet instances. These instances are managed as an AutoScaling Group, and can be elastically driven by workload, or can be static and driven by the configuration.

AWS Auto Scaling is not used with AWS Batch clusters.

For more details about AWS Auto Scaling, see https://aws.amazon.com/autoscaling/.

AWS Batch

AWS Batch is the AWS-managed job scheduler that dynamically provisions the optimal quantity and type of compute resources (for example, CPU or memory-optimized instances). It provisions resources based on the volume and the requirements of the batch jobs that are submitted. With AWS Batch, there is no need to install and manage batch computing software or server clusters to run your jobs.

AWS Batch is used only with AWS Batch clusters.

For more details, see https://aws.amazon.com/batch/.

AWS CloudFormation

AWS CloudFormation is the core service used by AWS ParallelCluster. Each cluster is represented as a stack. All resources required by the cluster are defined within the AWS ParallelCluster AWS CloudFormation template. AWS ParallelCluster CLI commands typically map to AWS CloudFormation stack commands, such as create, update, and delete. Instances that are launched within a cluster make HTTPS calls to the AWS CloudFormation endpoint for the region in which the cluster is launched.

For more details about AWS CloudFormation, see https://aws.amazon.com/cloudformation/.

Amazon CloudWatch

Amazon CloudWatch (CloudWatch) is used to log Docker image build steps and the standard output and error of the AWS Batch jobs.

CloudWatch is used only with AWS Batch clusters.

For more details, see https://aws.amazon.com/cloudwatch/.

AWS CodeBuild

AWS CodeBuild (CodeBuild) is used to automatically and transparently build Docker images at cluster creation time.

CodeBuild is used only with AWS Batch clusters.

For more details, see https://aws.amazon.com/codebuild/.

Amazon DynamoDB

Amazon DynamoDB (DynamoDB) is used to store minimal state of the cluster. The MasterServer tracks provisioned instances in a DynamoDB table.

DynamoDB is not used with AWS Batch clusters.

For more details, see https://aws.amazon.com/dynamodb/.

Amazon Elastic Block Store

Amazon Elastic Block Store (Amazon EBS) provides persistent storage for shared volumes. All Amazon EBS settings can be passed through the configuration. Amazon EBS volumes can either be initialized empty, or from an existing Amazon EBS snapshot.

For more details about Amazon EBS, see https://aws.amazon.com/ebs/.

Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (Amazon EC2) provides the computing capacity for AWS ParallelCluster. The MasterServer and ComputeFleet are Amazon EC2 instances. Any instance type that support HVM can be selected. The MasterServer and ComputeFleet can be different instance types, and the ComputeFleet can also be launched as a Spot instance. Instance store volumes found on the instances are mounted as striped LVM volumes.

For more details about Amazon EC2, see https://aws.amazon.com/ec2/.

Amazon Elastic Container Registry

Amazon Elastic Container Registry (Amazon ECR) stores the Docker images built at cluster creation time. The Docker images are then used by AWS Batch to run the containers for the submitted jobs.

Amazon ECR is used only with AWS Batch clusters.

For more details, see https://aws.amazon.com/ecr/.

AWS Identity and Access Management

AWS Identity and Access Management (IAM) is used within AWS ParallelCluster. It provides a least privileged IAM role for Amazon EC2 for the instance that is specific to each individual cluster. AWS ParallelCluster instances are given access only to the specific API calls that are required to deploy and manage the cluster.

With AWS Batch clusters, IAM roles are also created for the components that are involved with the Docker image building process at cluster creation time. These components include the Lambda functions that are allowed to add and delete Docker images to and from the Amazon ECR repository, and to delete the Amazon S3 bucket that is created for the cluster and CodeBuild project. There are also roles for AWS Batch resources, instances, and jobs.

For more details about IAM, see https://aws.amazon.com/iam/.

AWS Lambda

AWS Lambda (Lambda) runs the functions that orchestrate Docker image creation. Lambda also manages the cleanup of custom cluster resources, such as Docker images stored in the Amazon ECR repository and on Amazon S3.

Lambda is used only with AWS Batch clusters.

For more details, see https://aws.amazon.com/lambda/.

Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) is used to receive notifications from Auto Scaling. These events are called life cycle events, and are generated when an instance launches or terminates in an Autoscaling Group. Within AWS ParallelCluster, the Amazon SNS topic for the Autoscaling Group is subscribed to an Amazon SQS queue.

Amazon SNS is not used with AWS Batch clusters.

For more details about Amazon SNS, see https://aws.amazon.com/sns/.

Amazon Simple Queue Service

Amazon Simple Queue Service (Amazon SQS) is used to hold notification messages from Auto Scaling, sent through Amazon SNS, and notifications from the ComputeFleet instances. Using Amazon SQS decouples the sending of notifications from receiving them, and allows the Master to handle them through polling. The MasterServer runs Amazon SQSwatcher and polls the queue. Auto Scaling and the ComputeFleet instances post messages to the queue.

Amazon SQS is not used with AWS Batch clusters.

For more details about Amazon SQS, see https://aws.amazon.com/sqs/.

Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) is used to store the AWS ParallelCluster templates used in each region. AWS ParallelCluster can be configured to allow CLI/SDK tools to use Amazon S3.

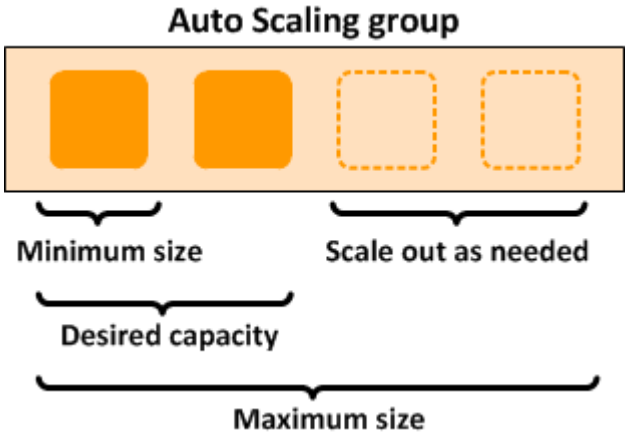
When an AWS Batch cluster is used, an Amazon S3 bucket in the customer's account is used for storage. For example, it stores artifacts used by the Docker image creation, and scripts from submitted jobs.

For more details, see https://aws.amazon.com/s3/.

AWS ParallelCluster Auto Scaling

The auto scaling strategy described here applies to HPC clusters that are deployed with one of the supported traditional job schedulers, either SGE, Slurm or Torque. When deployed with one of these schedulers, AWS ParallelCluster implements the scaling capabilities by managing the Auto Scaling Group (ASG) of the compute nodes, and then changing the scheduler configuration as needed. For HPC clusters that are based on AWS Batch, AWS ParallelCluster relies on the elastic scaling capabilities provided by the AWS-managed job scheduler. For more information, see What Is Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Clusters deployed with AWS ParallelCluster are elastic in several ways. Setting the `initial_queue_size` specifies the minimum size value of the ComputeFleet ASG, and also the desired capacity value. Setting the `max_queue_size` specifies the maximum size value of the ComputeFleet ASG.



Scaling Up

Every minute, a process called `jobwatcher` runs on the master instance. It evaluates the current number of instances required by the pending jobs in the queue. If the total number of busy nodes and requested nodes is greater than the current desired value in the ASG, it adds more instances. If you submit more jobs, the queue is re-evaluated and the ASG is updated, up to the specified `max_queue_size`.

With an SGE scheduler, each job requires a number of slots to run (one slot corresponds to one processing unit, for example, a vCPU). To evaluate the number of instances that are required to serve the currently pending jobs, the `jobwatcher` divides the total number of requested slots by the capacity of a single compute node. The capacity of a compute node that corresponds to the number of available vCPUs depends on the Amazon EC2 instance type that is specified in the cluster configuration.

With Slurm and Torque schedulers, each job might require both a number of nodes and a number of slots per node, depending on circumstance. For each request, the `jobwatcher` determines the number of compute nodes that are needed to fulfill the new computational requirements. For example, let's assume a cluster with `c5.2xlarge` (8 vCPU) as the compute instance type, and three queued pending jobs with the following requirements:

- job1: 2 nodes / 4 slots each
- job2: 3 nodes / 2 slots each
- job3: 1 node / 4 slots each

In this example, the `jobwatcher` requires three new compute instances in the ASG to serve the three jobs.

Current limitation: auto scale up logic does not consider partially loaded busy nodes. i.e. A node that is running a job is considered busy even if there are empty slots.

Scaling Down

On each compute node, a process called `nodewatcher` runs and evaluates the idle time of the node. An instance is terminated when both of the following conditions are met:

- An instance has no jobs for a period of time longer than the `scaledown_idletime` (the default setting is 10 minutes)
- There are no pending jobs in the cluster

To terminate an instance, `nodewatcher` calls the TerminateInstanceInAutoScalingGroup API call, which removes an instance if the size of the ASG is at least the minimum ASG size. This process scales down a cluster without affecting running jobs. It also enables an elastic cluster, with a fixed base number of instances.

Static Cluster

The value of auto scaling is the same for HPC as with any other workloads. The only difference is that AWS ParallelCluster has code that makes it interact more intelligently. For example, if a static cluster is required, you set the `initial_queue_size` and `max_queue_size` parameters to the exact size of cluster that is required,. and then you set the `maintain_initial_size` parameter to true. This causes the ComputeFleet ASG to have the same value for minimum, maximum, and desired capacity.

Tutorials

The following tutorials show you how to get started with AWS ParallelCluster, and provide best practice guidance for some common tasks.

Topics

Running Your First Job on AWS ParallelCluster

This tutorial walks you through running your first Hello World job on AWS ParallelCluster.

If you haven't yet completed installation of AWS ParallelCluster, and configured your CLI, follow the instructions in the getting started (p. 2) guide before continuing with this tutorial.

Verifying Your Installation

First, we verify that AWS ParallelCluster is correctly installed and configured.

```
$ pcluster version
```

This returns the running version of AWS ParallelCluster. If the output gives you a message about configuration, you need to run the following to configure AWS ParallelCluster:

```
$ pcluster configure
```

Creating Your First Cluster

Now it's time to create your first cluster. Because the workload for this tutorial isn't performance intensive, we can use the default instance size of `t2.micro`. (For production workloads, you choose an instance size that best fits your needs.)

Let's call your cluster hello-world.

```
$ pcluster create hello-world
```

When the cluster is created, you see output similar to the following:

```
Starting: hello-world
Status: parallelcluster-hello-world - CREATE_COMPLETE
MasterPublicIP = 54.148.x.x
ClusterUser: ec2-user
MasterPrivateIP = 192.168.x.x
GangliaPrivateURL = http://192.168.x.x/ganglia/
GangliaPublicURL = http://54.148.x.x/ganglia/
```

The message `CREATE_COMPLETE` shows that the cluster created successfully. The output also provides us with the public and private IP addresses of our master node. We need this IP to log in.

Logging into Your Master Instance

Use your OpenSSH pem file to log into your master instance.

```
pcluster ssh hello-world -i /path/to/keyfile.pem
```

After you log in, run the command `qhost` to verify that your compute nodes are set up and configured.

```
$ qhost
HOSTNAME                ARCH         NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO
 SWAPUS
----------------------------------------------------------------------------------------------
global                  -               -    -    -    -     -       -       -       -
   -
ip-192-168-1-125        lx-amd64        2    1    2    2  0.15    3.7G  130.8M 1024.0M
 0.0
ip-192-168-1-126        lx-amd64        2    1    2    2  0.15    3.7G  130.8M 1024.0M
 0.0
```

The output shows that we have two compute nodes in our cluster, both with two threads available to them.

Running Your First Job Using SGE

Next, we create a job that sleeps for a little while and then outputs its own hostname.

Create a file called `hellojob.sh`, with the following contents.

```
#!/bin/bash
sleep 30
echo "Hello World from $(hostname)"
```

Next, submit the job using `qsub`, and verify that it runs.

```
$ qsub hellojob.sh
Your job 1 ("hellojob.sh") has been submitted
```

Now, you can view your queue and check the status of the job.

```
$ qstat
job-ID  prior   name       user         state submit/start at     queue
      slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
      1 0.55500 hellojob.s ec2-user     r     03/24/2015 22:23:48
 all.q@ip-192-168-1-125.us-west     1
```

The output shows that the job is currently in a running state. Wait 30 seconds for the job to finish, and then run `qstat` again.

```
$ qstat
$
```

Now that there are no jobs in the queue, we can check for output in our current directory.

```
$ ls -l
total 8
-rw-rw-r-- 1 ec2-user ec2-user 48 Mar 24 22:34 hellojob.sh
-rw-r--r-- 1 ec2-user ec2-user  0 Mar 24 22:34 hellojob.sh.e1
-rw-r--r-- 1 ec2-user ec2-user 34 Mar 24 22:34 hellojob.sh.o1
```

In the output, we see an "`e1`" and "`o1`" file in our job script. Because the `e1` file is empty, there was no output to stderr. If we view the `o1` file, we can see output from our job.

```
$ cat hellojob.sh.o1
Hello World from ip-192-168-1-125
```

The output also shows that our job ran successfully on instance `ip-192-168-1-125`.

Building a Custom AWS ParallelCluster AMI

Important
Building a custom AMI is not the recommended approach for customizing AWS ParallelCluster. After you build your own AMI, you no longer receive updates or bug fixes with future releases of AWS ParallelCluster. Also, you must repeat the steps used to create your custom AMI with each new AWS ParallelCluster release.

Before reading any further, take a look at the Custom Bootstrap Actions (p. 31) section to determine if the modifications you want to make can be scripted and supported with future AWS ParallelCluster releases.

While building a custom AMI is not ideal, there are scenarios in which building a custom AMI for AWS ParallelCluster is necessary. This tutorial guides you through the process.

How to Customize the AWS ParallelCluster AMI

There are three ways to use a custom AWS ParallelCluster AMI. Two methods require you to build a new AMI that is available under your AWS account. The third method—Use a Custom AMI at Runtime—does not require you to build anything in advance. Select the appropriate method based on your needs.

Customization Methods

- Modify an AWS ParallelCluster AMI (p. 96)
- Build a Custom AWS ParallelCluster AMI (p. 97)
- Use a Custom AMI at Runtime (p. 98)

Modify an AWS ParallelCluster AMI

This is the safest method, because the base AWS ParallelCluster AMI is often updated with new releases. This AMI has all of the components required for AWS ParallelCluster to function as installed and configured. You can start with this as the base.

1. In the AMI list, find the AMI that corresponds to the Region you are using. The AMI list to use must match the version of AWS ParallelCluster you are using. Run `pcluster version` to verify the version. For example:
   - For AWS ParallelCluster 2.4.1 -> https://github.com/aws/aws-parallelcluster/blob/v2.4.1/amis.txt
   - For AWS ParallelCluster 2.4.0 -> https://github.com/aws/aws-parallelcluster/blob/v2.4.0/amis.txt

- For AWS ParallelCluster 2.3.1 -> https://github.com/aws/aws-parallelcluster/blob/v2.3.1/amis.txt
- For AWS ParallelCluster 2.2.1 -> https://github.com/aws/aws-parallelcluster/blob/v2.2.1/amis.txt
- For AWS ParallelCluster 2.1.1 -> https://github.com/aws/aws-parallelcluster/blob/v2.1.1/amis.txt
- For CfnCluster 1.6.1 -> https://github.com/aws/aws-parallelcluster/blob/v1.6.1/amis.txt

2. Within the Amazon EC2 console, choose **Launch Instance**.
3. Navigate to **Community AMIs**, and enter the AMI id for your Region into the search box.
4. Select the AMI, choose your instance type and properties, and launch your instance.
5. Log into your instance using the OS user and your SSH key.
6. Customize your instance as required.
7. Run the following command to prepare your instance for AMI creation:

```
sudo /usr/local/sbin/ami_cleanup.sh
```

8. Stop the instance.
9. Create a new AMI from the instance.
10. Enter the AMI id in the custom_ami (p. 60) field within your cluster configuration.

Build a Custom AWS ParallelCluster AMI

If you have a customized AMI and software already in place, you can apply the changes needed by AWS ParallelCluster on top of it.

1. Install the following tools in your local system, together with the AWS ParallelCluster CLI:
   - Packer: find the latest version for your OS from the Packer website, and install it.
   - ChefDK: find the latest version for your OS from the ChefDK website, and install it.
2. Verify that the `packer` and `berks` commands are available in your PATH after you have installed the tools in Step 1.
3. Configure your AWS account credentials so that Packer can make calls to AWS API operations on your behalf. The minimal set of required permissions necessary for Packer to work are documented in the Packer doc.
4. Use the command `createami` in the AWS ParallelCluster CLI to build an AWS ParallelCluster AMI starting from the one that you provide as base:

```
pcluster createami --ami-id <BASE AMI> --os <BASE OS AMI>
```

   Important
   You cannot use an AWS ParallelCluster AMI as *<BASE AMI>* for the `createami` command. If you do, the command fails.

   For other parameters, consult the command help:

```
pcluster createami -h
```

5. The command in Step 4 executes Packer, which does the following:
   a. Launches an instance using the base AMI provided.
   b. Applies the AWS ParallelCluster cookbook to the instance, in order to install software and perform other necessary configuration tasks.
   c. Stops the instance.
   d. Creates an new AMI from the instance.
   e. Terminates the instance after the AMI is created.

f. Outputs the new AMI ID string to use to create your cluster.

6. To create your cluster, enter the AMI ID in the custom_ami (p. 60) field within your cluster configuration.

Note
The instance type used to build a custom AWS ParallelCluster AMI is a `t2.xlarge`. It does not qualify for the AWS free tier. You are charged for any instances that are created when building this AMI.

Use a Custom AMI at Runtime

If you don't want to create anything in advance, you can use your AMI and create an AWS ParallelCluster from that.

Note that with this method, the AWS ParallelCluster creation time is longer, because all software that is needed by AWS ParallelCluster at cluster creation time must be installed. Also, scaling up for every new node takes more time.

- Enter the AMI id in the custom_ami (p. 60) field within your cluster configuration.

Running an MPI Job with AWS ParallelCluster and awsbatch Scheduler

This tutorial walks you through running an MPI job with `awsbatch` as a scheduler.

If you haven't yet installed AWS ParallelCluster and configured your CLI, follow the instructions in the getting started (p. 2) guide before continuing with this tutorial. Also, make sure to read through the awsbatch networking setup (p. 30) documentation before moving to the next step.

Creating the Cluster

First, let's create a configuration for a cluster that uses `awsbatch` as the scheduler. Make sure to insert the missing data in the `vpc` section and the `key_name` field with the resources that you created at configuration time.

```
[global]
sanity_check = true

[aws]
aws_region_name = us-east-1

[cluster awsbatch]
base_os = alinux
# Replace with the name of the key you intend to use.
key_name = key-#######
vpc_settings = my-vpc
scheduler = awsbatch
compute_instance_type = optimal
min_vcpus = 2
desired_vcpus = 2
max_vcpus = 24

[vpc  my-vpc]
# Replace with the id of the vpc you intend to use.
vpc_id = vpc-#######
```

```
# Replace with id of the subnet for the Master node.
master_subnet_id = subnet-########
# Replace with id of the subnet for the Compute nodes.
# A NAT Gateway is required for MNP.
compute_subnet_id = subnet-#######
```

You can now start the creation of the cluster. Let's call our cluster *awsbatch-tutorial*.

```
$ pcluster create -c /path/to/the/created/config/aws_batch.config -t awsbatch awsbatch-tutorial
```

When the cluster is created, you see output similar to the following:

```
Beginning cluster creation for cluster: awsbatch-tutorial
Creating stack named: parallelcluster-awsbatch
Status: parallelcluster-awsbatch - CREATE_COMPLETE
MasterPublicIP: 54.160.xxx.xxx
ClusterUser: ec2-user
MasterPrivateIP: 10.0.0.15
```

Logging into Your Master Instance

The AWS ParallelCluster Batch CLI (p. 47) commands are all available on the client machine where AWS ParallelCluster is installed. However, we are going to SSH into the master node and submit the jobs from there. This allows us to take advantage of the NFS volume that is shared between the master and all Docker instances that run AWS Batch jobs.

Use your SSH pem file to log into your master instance.

```
$ pcluster ssh awsbatch-tutorial -i /path/to/keyfile.pem
```

When you are logged in, run the commands `awsbqueues` and `awsbhosts` to show the configured AWS Batch queue and the running Amazon ECS instances.

```
[ec2-user@ip-10-0-0-111 ~]$ awsbqueues
jobQueueName                    status
------------------------------  --------
parallelcluster-awsbatch-tutorial  VALID

[ec2-user@ip-10-0-0-111 ~]$ awsbhosts
ec2InstanceId        instanceType    privateIpAddress    publicIpAddress      runningJobs
-------------------  --------------  ------------------  -----------------  -------------
i-0d6a0c8c560cd5bed  m4.large        10.0.0.235          34.239.174.236                 0
```

As you can see from the output, we have one single running host. This is due to the value we chose for the section called "`min_vcpus`" (p. 64) in the configuration. If you want to display additional details about the AWS Batch queue and hosts, add the `-d` flag to the command.

Running Your First Job Using AWS Batch

Before moving to MPI, let's create a dummy job that sleeps for a little while and then outputs its own hostname, greeting the name passed as a parameter.

Create a file called "hellojob.sh" with the following content.

```
#!/bin/bash
```

```
sleep 30
echo "Hello $1 from $(hostname)"
echo "Hello $1 from $(hostname)" > "/shared/secret_message_for_${1}_by_${AWS_BATCH_JOB_ID}"
```

Next, submit the job using `awsbsub` and verify that it runs.

```
$ awsbsub -jn hello -cf hellojob.sh Luca
Job 6efe6c7c-4943-4c1a-baf5-edbfeccab5d2 (hello) has been submitted.
```

View your queue, and check the status of the job.

```
$ awsbstat
jobId                                jobName      status    startedAt            stoppedAt
    exitCode
-------------------------------------  -----------  --------  -------------------
 -----------  ----------
6efe6c7c-4943-4c1a-baf5-edbfeccab5d2  hello        RUNNING   2018-11-12 09:41:29  -
    -
```

The output provides detailed information for the job.

```
$ awsbstat 6efe6c7c-4943-4c1a-baf5-edbfeccab5d2
jobId                    : 6efe6c7c-4943-4c1a-baf5-edbfeccab5d2
jobName                  : hello
createdAt                : 2018-11-12 09:41:21
startedAt                : 2018-11-12 09:41:29
stoppedAt                : -
status                   : RUNNING
statusReason             : -
jobDefinition            : parallelcluster-myBatch:1
jobQueue                 : parallelcluster-myBatch
command                  : /bin/bash -c 'aws s3 --region us-east-1 cp s3://parallelcluster-
mybatch-lui1ftboklhpns95/batch/job-hellojob_sh-1542015680924.sh /tmp/batch/job-
hellojob_sh-1542015680924.sh; bash /tmp/batch/job-hellojob_sh-1542015680924.sh Luca'
exitCode                 : -
reason                   : -
vcpus                    : 1
memory[MB]               : 128
nodes                    : 1
logStream                : parallelcluster-myBatch/default/c75dac4a-5aca-4238-
a4dd-078037453554
log                      : https://console.aws.amazon.com/cloudwatch/home?region=us-
east-1#logEventViewer:group=/aws/batch/job;stream=parallelcluster-myBatch/default/
c75dac4a-5aca-4238-a4dd-078037453554
--------------------------
```

Note that the job is currently in a `RUNNING` state. Wait 30 seconds for the job to finish, and then run `awsbstat` again.

```
$ awsbstat
jobId                                jobName      status    startedAt            stoppedAt
    exitCode
-------------------------------------  -----------  --------  -------------------
 -----------  ----------
```

Now you can see that the job is in the `SUCCEEDED` status.

```
$ awsbstat -s SUCCEEDED
```

```
jobId                                    jobName      status     startedAt
 stoppedAt              exitCode
---------------------------------------  -----------  ---------  -------------------
 -------------------  ----------
6efe6c7c-4943-4c1a-baf5-edbfeccab5d2  hello        SUCCEEDED  2018-11-12 09:41:29
 2018-11-12 09:42:00           0
```

Because there are no jobs in the queue now, we can check for output through the `awsbout` command.

```
$ awsbout 6efe6c7c-4943-4c1a-baf5-edbfeccab5d2
2018-11-12 09:41:29: Starting Job 6efe6c7c-4943-4c1a-baf5-edbfeccab5d2
download: s3://parallelcluster-mybatch-lui1ftboklhpns95/batch/job-
hellojob_sh-1542015680924.sh to tmp/batch/job-hellojob_sh-1542015680924.sh
2018-11-12 09:42:00: Hello Luca from ip-172-31-4-234
```

We can see that our job successfully ran on instance "ip-172-31-4-234".

If you look into the `/shared` directory, you find a secret message for you.

To explore all of the available features that are not part of this tutorial, see the AWS ParallelCluster Batch CLI documentation (p. 47). When you are ready to continue the tutorial, let's move on and see how to submit an MPI job.

Running an MPI Job in a Multi-Node Parallel Environment

While still logged into the master node, create a file in the `/shared` directory named `mpi_hello_world.c`. Add the following MPI program to the file:

```
// Copyright 2011 www.mpitutorial.com
//
// An intro MPI hello world program that uses MPI_Init, MPI_Comm_size,
// MPI_Comm_rank, MPI_Finalize, and MPI_Get_processor_name.
//
#include <mpi.h>
#include <stdio.h>
#include <stddef.h>

int main(int argc, char** argv) {
  // Initialize the MPI environment. The two arguments to MPI Init are not
  // currently used by MPI implementations, but are there in case future
  // implementations might need the arguments.
  MPI_Init(NULL, NULL);

  // Get the number of processes
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of the process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);

  // Print off a hello world message
  printf("Hello world from processor %s, rank %d out of %d processors\n",
         processor_name, world_rank, world_size);
```

```
  // Finalize the MPI environment. No more MPI calls can be made after this
  MPI_Finalize();
}
```

Now save the following code as `submit_mpi.sh`:

```
#!/bin/bash
echo "ip container: $(/sbin/ip -o -4 addr list eth0 | awk '{print $4}' | cut -d/ -f1)"
echo "ip host: $(curl -s "http://169.254.169.254/latest/meta-data/local-ipv4")"

# get shared dir
IFS=',' _shared_dirs=(${PCLUSTER_SHARED_DIRS})
_shared_dir=${_shared_dirs[0]}
_job_dir="${_shared_dir}/${AWS_BATCH_JOB_ID%#*}-${AWS_BATCH_JOB_ATTEMPT}"
_exit_code_file="${_job_dir}/batch-exit-code"

if [[ "${AWS_BATCH_JOB_NODE_INDEX}" -eq  "${AWS_BATCH_JOB_MAIN_NODE_INDEX}" ]]; then
    echo "Hello I'm the main node $(hostname)! I run the mpi job!"

    mkdir -p "${_job_dir}"

    echo "Compiling..."
    /usr/lib64/openmpi/bin/mpicc -o "${_job_dir}/mpi_hello_world" "${_shared_dir}/
mpi_hello_world.c"

    echo "Running..."
    /usr/lib64/openmpi/bin/mpirun --mca btl_tcp_if_include eth0 --allow-run-as-root --
machinefile "${HOME}/hostfile" "${_job_dir}/mpi_hello_world"

    # Write exit status code
    echo "0" > "${_exit_code_file}"
    # Waiting for compute nodes to terminate
    sleep 30
else
    echo "Hello I'm the compute node $(hostname)! I let the main node orchestrate the mpi
 execution!"
    # Since mpi orchestration happens on the main node, we need to make sure the containers
 representing the compute
    # nodes are not terminated. A simple trick is to wait for a file containing the status
 code to be created.
    # All compute nodes are terminated by Batch if the main node exits abruptly.
    while [ ! -f "${_exit_code_file}" ]; do
        sleep 2
    done
    exit $(cat "${_exit_code_file}")
fi
```

We are now ready to submit our first MPI job and make it run concurrently on three nodes:

```
$ awsbsub -n 3 -cf submit_mpi.sh
```

Now let's monitor the job status, and wait for it to enter the `RUNNING` status:

```
$ watch awsbstat -d
```

When the job enters the `RUNNING` status, we can look at its output. To show the output of the main node, append `#0` to the job id. To show the output of the compute nodes, use `#1` and `#2`:

```
[ec2-user@ip-10-0-0-111 ~]$ awsbout -s 5b4d50f8-1060-4ebf-ba2d-1ae868bbd92d#0
2018-11-27 15:50:10: Job id: 5b4d50f8-1060-4ebf-ba2d-1ae868bbd92d#0
```

```
2018-11-27 15:50:10: Initializing the environment...
2018-11-27 15:50:10: Starting ssh agents...
2018-11-27 15:50:11: Agent pid 7
2018-11-27 15:50:11: Identity added: /root/.ssh/id_rsa (/root/.ssh/id_rsa)
2018-11-27 15:50:11: Mounting shared file system...
2018-11-27 15:50:11: Generating hostfile...
2018-11-27 15:50:11: Detected 1/3 compute nodes. Waiting for all compute nodes to start.
2018-11-27 15:50:26: Detected 1/3 compute nodes. Waiting for all compute nodes to start.
2018-11-27 15:50:41: Detected 1/3 compute nodes. Waiting for all compute nodes to start.
2018-11-27 15:50:56: Detected 3/3 compute nodes. Waiting for all compute nodes to start.
2018-11-27 15:51:11: Starting the job...
download: s3://parallelcluster-awsbatch-tutorial-iwyl4458saiwgwvg/batch/job-
submit_mpi_sh-1543333713772.sh to tmp/batch/job-submit_mpi_sh-1543333713772.sh
2018-11-27 15:51:12: ip container: 10.0.0.180
2018-11-27 15:51:12: ip host: 10.0.0.245
2018-11-27 15:51:12: Compiling...
2018-11-27 15:51:12: Running...
2018-11-27 15:51:12: Hello I'm the main node! I run the mpi job!
2018-11-27 15:51:12: Warning: Permanently added '10.0.0.199' (RSA) to the list of known
 hosts.
2018-11-27 15:51:12: Warning: Permanently added '10.0.0.147' (RSA) to the list of known
 hosts.
2018-11-27 15:51:13: Hello world from processor ip-10-0-0-180.ec2.internal, rank 1 out of 6
 processors
2018-11-27 15:51:13: Hello world from processor ip-10-0-0-199.ec2.internal, rank 5 out of 6
 processors
2018-11-27 15:51:13: Hello world from processor ip-10-0-0-180.ec2.internal, rank 0 out of 6
 processors
2018-11-27 15:51:13: Hello world from processor ip-10-0-0-199.ec2.internal, rank 4 out of 6
 processors
2018-11-27 15:51:13: Hello world from processor ip-10-0-0-147.ec2.internal, rank 2 out of 6
 processors
2018-11-27 15:51:13: Hello world from processor ip-10-0-0-147.ec2.internal, rank 3 out of 6
 processors

[ec2-user@ip-10-0-0-111 ~]$ awsbout -s 5b4d50f8-1060-4ebf-ba2d-1ae868bbd92d#1
2018-11-27 15:50:52: Job id: 5b4d50f8-1060-4ebf-ba2d-1ae868bbd92d#1
2018-11-27 15:50:52: Initializing the environment...
2018-11-27 15:50:52: Starting ssh agents...
2018-11-27 15:50:52: Agent pid 7
2018-11-27 15:50:52: Identity added: /root/.ssh/id_rsa (/root/.ssh/id_rsa)
2018-11-27 15:50:52: Mounting shared file system...
2018-11-27 15:50:52: Generating hostfile...
2018-11-27 15:50:52: Starting the job...
download: s3://parallelcluster-awsbatch-tutorial-iwyl4458saiwgwvg/batch/job-
submit_mpi_sh-1543333713772.sh to tmp/batch/job-submit_mpi_sh-1543333713772.sh
2018-11-27 15:50:53: ip container: 10.0.0.199
2018-11-27 15:50:53: ip host: 10.0.0.227
2018-11-27 15:50:53: Compiling...
2018-11-27 15:50:53: Running...
2018-11-27 15:50:53: Hello I'm a compute note! I let the main node orchestrate the mpi
 execution!
```

We can now confirm that the job completed successfully:

```
[ec2-user@ip-10-0-0-111 ~]$ awsbstat -s ALL
jobId                                 jobName        status     startedAt
 stoppedAt            exitCode
------------------------------------  -------------  ---------  -------------------
 -------------------  ----------
5b4d50f8-1060-4ebf-ba2d-1ae868bbd92d  submit_mpi_sh  SUCCEEDED  2018-11-27 15:50:10
 2018-11-27 15:51:26  -
```

Note: if you want to terminate a job before it ends, you can use the `awsbkill` command.

Disk Encryption with a Custom KMS Key

AWS ParallelCluster supports the configuration options `ebs_kms_key_id` and `fsx_kms_key_id`. These options allow you to provide a custom AWS KMS key for Amazon EBS Disk encryption or Amazon FSx for Lustre. To use them, you specify an `ec2_iam_role`.

In order for the cluster to create, the AWS KMS key must know the name of the cluster's role. This prevents you from using the role created on cluster create, requiring a custom `ec2_iam_role`.

Creating the Role

First you create a policy:

1. Go to the IAM Console: https://console.aws.amazon.com/iam/home.
2. Under **Policies**, **Create policy**, click the **JSON** tab.
3. As the policy's body, paste in the Instance Policy (p. 35). Make sure to replace all occurrences of *<AWS ACCOUNT ID>* and *<REGION>*.
4. Name the policy `ParallelClusterInstancePolicy`, and then click **Create Policy**.

Next create a role:

1. Under **Roles**, create a role.
2. Click `EC2` as the trusted entity.
3. Under **Permissions**, search for the `ParallelClusterInstancePolicy` role that you just created, and attach it.
4. Name the role `ParallelClusterInstanceRole`, and then click **Create Role**.

Give Your Key Permissions

In the IAM Console > **Encryption Keys** > click your key.

Click **Add User**, and search for the *ParallelClusterInstanceRole* you just created. Attach it.

Creating the Cluster

Now create a cluster. The following is an example of a cluster with encrypted `Raid 0` drives:

```
[cluster default]
...
raid_settings = rs
ec2_iam_role = ParallelClusterInstanceRole

[raid rs]
shared_dir = raid
raid_type = 0
num_of_raid_volumes = 2
volume_size = 100
encrypted = true
ebs_kms_key_id = xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The following is an example with the Amazon FSx for Lustre file system:

```
[cluster default]
```

```
...
fsx_settings = fs
ec2_iam_role = ParallelClusterInstanceRole

[fsx fs]
shared_dir = /fsx
storage_capacity = 3600
imported_file_chunk_size = 1024
export_path = s3://bucket/folder
import_path = s3://bucket
weekly_maintenance_start_time = 1:00:00
fsx_kms_key_id = xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Similar configurations apply to Amazon EBS and Amazon FSx based file systems.

Development

You can use the following sections to get started with the development of AWS ParallelCluster.

Warning
The following sections include instructions for using a custom version of the cookbook recipes and a custom AWS ParallelCluster node package. This information covers an advanced method of customizing AWS ParallelCluster, with potential issues that can be hard to debug. The AWS ParallelCluster team highly recommends using the scripts in Custom Bootstrap Actions (p. 31) for customization, because post-install hooks are generally easier to debug and more portable across releases of AWS ParallelCluster.

Topics

Setting Up a Custom AWS ParallelCluster Cookbook

Warning
The following are instructions for using a custom version of the AWS ParallelCluster cookbook recipes. This is an advanced method of customizing AWS ParallelCluster, with potential issues that can be hard to debug. The AWS ParallelCluster team highly recommends using the scripts in Custom Bootstrap Actions (p. 31) for customization, because post-install hooks are generally easier to debug and more portable across releases of AWS ParallelCluster.

Steps

1. Identify the AWS ParallelCluster Cookbook working directory where you have cloned the AWS ParallelCluster Cookbook code.

```
_cookbookDir=<path to cookbook>
```

2. Detect the current version of the AWS ParallelCluster Cookbook.

```
_version=$(grep version ${_cookbookDir}/metadata.rb|awk '{print $2}'| tr -d \')
```

3. Create an archive of the AWS ParallelCluster Cookbook and calculate its md5.

```
cd "${_cookbookDir}"
_stashName=$(git stash create)
git archive --format tar --prefix="aws-parallelcluster-cookbook-${_version}/"
 "${_stashName:-HEAD}" | gzip > "aws-parallelcluster-cookbook-${_version}.tgz"
md5sum "aws-parallelcluster-cookbook-${_version}.tgz" > "aws-parallelcluster-cookbook-
${_version}.md5"
```

4. Create an Amazon S3 bucket and upload the archive, its md5, and its last modified date into the bucket. Give public readable permission through a public-read ACL.

```
_bucket=<the bucket name>
aws s3 cp --acl public-read aws-parallelcluster-cookbook-${_version}.tgz s3://${_bucket}/
cookbooks/aws-parallelcluster-cookbook-${_version}.tgz
```

```
aws s3 cp --acl public-read aws-parallelcluster-cookbook-${_version}.md5 s3://${_bucket}/
cookbooks/aws-parallelcluster-cookbook-${_version}.md5
aws s3api head-object --bucket ${_bucket} --key cookbooks/aws-parallelcluster-cookbook-
${_version}.tgz --output text --query LastModified > aws-parallelcluster-cookbook-
${_version}.tgz.date
aws s3 cp --acl public-read aws-parallelcluster-cookbook-${_version}.tgz.date s3://
${_bucket}/cookbooks/aws-parallelcluster-cookbook-${_version}.tgz.date
```

5. Add the following variable to the AWS ParallelCluster configuration file, under the [cluster] section (p. 57).

```
custom_chef_cookbook = https://${_bucket}.s3.<the bucket region>.amazonaws.com/cookbooks/
aws-parallelcluster-cookbook-${_version}.tgz
```

Setting Up a Custom AWS ParallelCluster Node Package

Warning
The following are instructions for using a custom version of the AWS ParallelCluster node package. This is an advanced method of customizing AWS ParallelCluster, with potential issues that can be hard to debug. The AWS ParallelCluster team highly recommends using the scripts in Custom Bootstrap Actions (p. 31) for customization, because post-install hooks are generally easier to debug and more portable across releases of AWS ParallelCluster.

Steps

1. Identify the AWS ParallelCluster node working directory where you have cloned the AWS ParallelCluster node code.

```
_nodeDir=<path to node package>
```

2. Detect the current version of the AWS ParallelCluster node.

```
_version=$(grep "version = \"" ${_nodeDir}/setup.py |awk '{print $3}' | tr -d \")
```

3. Create an archive of the AWS ParallelCluster Node.

```
cd "${_nodeDir}"
_stashName=$(git stash create)
git archive --format tar --prefix="aws-parallelcluster-node-${_version}/" "${_stashName:-
HEAD}" | gzip > "aws-parallelcluster-node-${_version}.tgz"
```

4. Create an Amazon S3 bucket and upload the archive into the bucket. Give public readable permission through a public-read ACL.

```
_bucket=<the bucket name>
aws s3 cp --acl public-read aws-parallelcluster-node-${_version}.tgz s3://${_bucket}/
node/aws-parallelcluster-node-${_version}.tgz
```

5. Add the following variable to the AWS ParallelCluster configuration file, under the "*[cluster ...]* section".

```
extra_json = { "cluster" : { "custom_node_package" : "https://${_bucket}.s3.<the bucket
 region>.amazonaws.com/node/aws-parallelcluster-node-${_version}.tgz" } }
```

AWS ParallelCluster Troubleshooting

The AWS ParallelCluster community maintains a wiki with many troubleshooting tips at the aws-parallelcluster wiki.

Failure submitting AWS Batch multi-node parallel jobs

If you have problems submitting multi-node parallel jobs when using AWS Batch as the job scheduler, it's recommend to upgrade to AWS ParallelCluster 2.5.0. If that is not feasible, a workaround can be found at Self patch a Cluster Used for Submitting Multi node Parallel Jobs through AWS Batch.

Placement Groups and Instance Launch Issues

In order to get the lowest inter-node latency, AWS recommends that you use a placement group. A placement group guarantees that your instances will be on the same networking backbone. If not enough instances are available when the request is made, an `InsufficientInstanceCapacity` error is returned. To reduce the possibility of receiving an `InsufficientInstanceCapacity` error when using cluster placement groups, set the `the section called “placement_group” (p. 65)` parameter to `DYNAMIC` and set the `the section called “placement” (p. 64)` parameter to `compute`.

If a high performance shared filesystem is needed, consider using Amazon FSx for Lustre.

If the master node must be in the placement group, use the same instance type and subnet for both the master and compute nodes. In other words, the `the section called “compute_instance_type” (p. 60)` parameter has the same value as the `the section called “master_instance_type” (p. 63)` parameter, the `the section called “placement” (p. 64)` parameter is set to `cluster` and the `the section called “compute_subnet_id” (p. 80)` parameter is not specified (the value of the `the section called “master_subnet_id” (p. 80)` parameter is used for the compute notes.)

For more information, see Troubleshooting Instance Launch issues and Placement Groups Roles and Limitations in the *Amazon EC2 User Guide for Linux Instances*

Document History

The following table describes the major updates and new features for the *AWS ParallelCluster User Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
AWS ParallelCluster 2.5.0 released.	AWS ParallelCluster 2.5.0 introduces support for Ubuntu 18.04, scheduling with GPU options in Slurm, and NICE DCV on Centos 7. For more details on the other changes made for AWS ParallelCluster 2.5.0, see AWS ParallelCluster 2.5.0.	18 November 2019
AWS ParallelCluster introduces support for Intel MPI.	AWS ParallelCluster 2.4.1 introduces support for Intel MPI. For more information on, see the section called "Enable Intel MPI" (p. 52). For more details on the other changes made for AWS ParallelCluster 2.4.1, see AWS ParallelCluster 2.4.1.	29 July 2019
AWS ParallelCluster introduces support for EFA.	AWS ParallelCluster 2.4.0 introduces support for EFA. For more information, see the section called "Elastic Fabric Adapter" (p. 52). For more details on the other changes made for AWS ParallelCluster 2.4.0, see AWS ParallelCluster 2.4.0.	11 June 2019
AWS ParallelCluster documentation initial release on AWS Documentation website.	The AWS ParallelCluster documentation is now available in 10 languages and in both HTML and PDF formats.	11 June 2019