User Guide

AWS PCS

AWS PCS: User Guide

Table of Contents

What is AWS Parallel Computing Service?

AWS Parallel Computing Service (AWS PCS) is a managed service that makes it easier to run and scale high performance computing (HPC) workloads, and build scientific and engineering models on AWS using Slurm. Use AWS PCS to build compute clusters that integrate best in class AWS compute, storage, networking, and visualization. Run simulations or build scientific and engineering models. Streamline and simplify your cluster operations using built-in management and observability capabilities. Empower your users to focus on research and innovation by enabling them to run their applications and jobs in a familiar environment.

Topics

Concepts in AWS PCS

A cluster in AWS PCS has 1 or more queues, associated with at least 1 compute node group. Jobs are submitted to queues and run on EC2 instances defined by compute node groups. You can use these foundations to implement sophisticated HPC architectures.

Cluster

A cluster is a resource for managing resources and running workloads. A cluster is an AWS PCS resource that defines an assembly of compute, networking, storage, identity, and job scheduler configuration. You create a cluster by specifying which job scheduler you want to use (Slurm currently), what scheduler configuration you want, what service controller you want to manage the cluster, and in which VPC you want the cluster resources to be launched. The scheduler accepts and schedules jobs, and also launches the compute nodes (EC2 instances) that process those jobs.

Compute node group

A compute node group is a collection of compute nodes that AWS PCS uses to run jobs or provide interactive access to a cluster. When you define a compute node group, you specify common traits such as Amazon EC2 instance types, minimum and maximum instance count, target VPC subnets, Amazon Machine Image (AMI), purchase option, and custom launch configuration. AWS PCS uses these settings to efficiently launch, manage, and terminate compute nodes in a compute node group.

Queue

When you want to run a job on a specific cluster, you submit it to a particular queue (also sometimes called a *partition*). The job remains in the queue until AWS PCS schedules it to run on a compute node group. You associate one or more compute node groups with each queue. A queue is required to schedule and execute jobs on the underlying compute node group resources using various scheduling policies offered by the job scheduler. Users don’t submit jobs directly to a compute node or compute node group.

System administrator

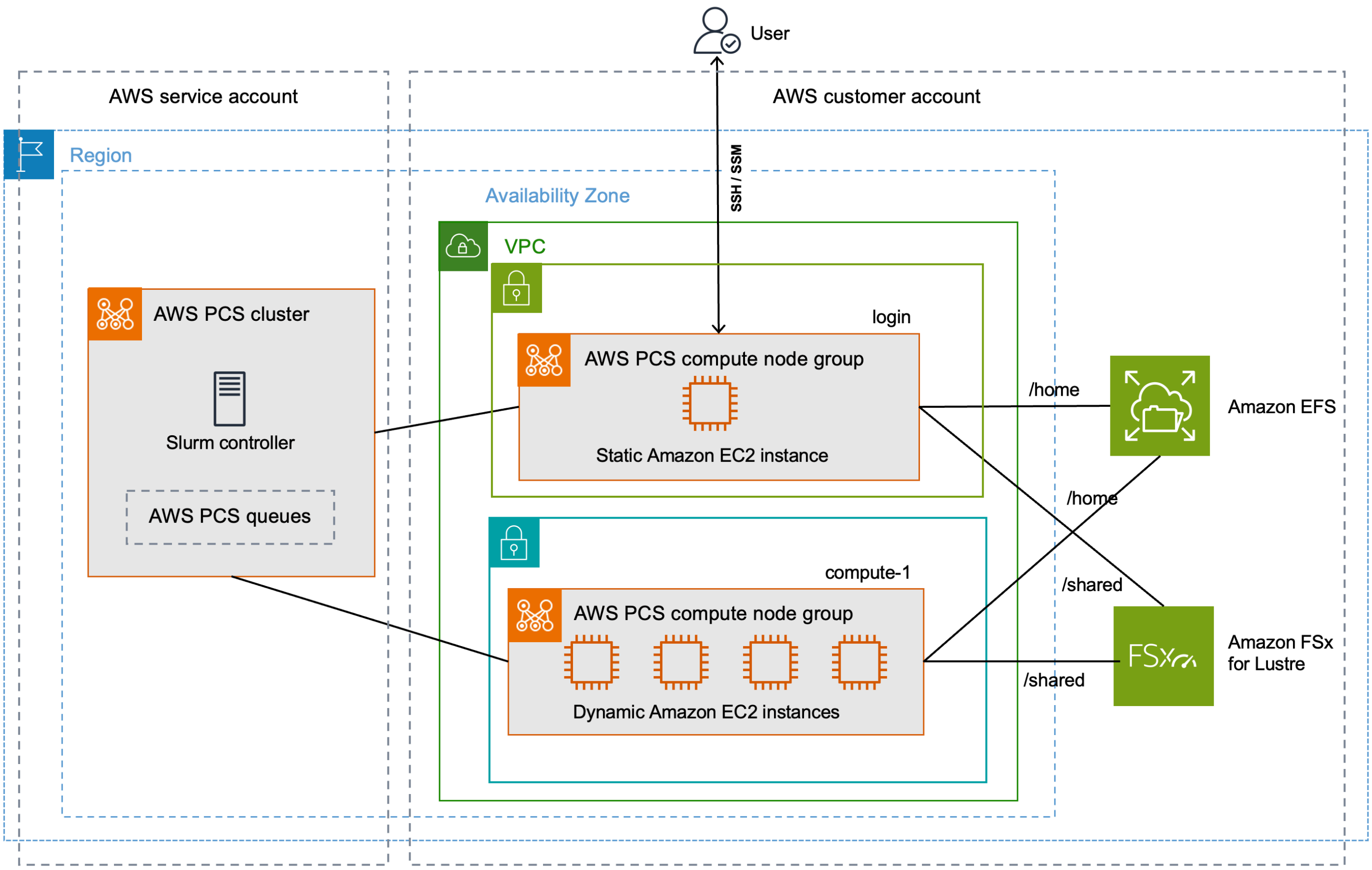
A system administrator deploys, maintains, and operates a cluster. They can access AWS PCS through the AWS Management Console, AWS PCS API, and AWS SDK. They have access to specific clusters through SSH or AWS Systems Manager, where they can run administrative tasks, run jobs, manage data, and perform other shell-based activities. For more information, see *AWS Systems Manager Documentation*.

End user

An end user doesn't have day-to-day responsibility to deploy or operate a cluster. They use a terminal interface (such as SSH) to access cluster resources, run jobs, manage data, and perform other shell-based activities.

Getting started with AWS Parallel Computing Service

This is a tutorial to create a simple cluster that you can use to try AWS PCS. The following figure shows the design of the cluster.



The tutorial cluster design has the following key components:

- A VPC and subnets that meet AWS PCS networking requirements.
- An Amazon EFS file system, which will be used as a shared home directory.
- An Amazon FSx for Lustre file system, which provides a shared high performance directory.
- An AWS PCS cluster, which provides a Slurm controller.
- 2 AWS PCS compute node groups.
    - The `login` node group, which provides shell-based interactive access to the system.
    - The `compute-1` node group provides elastically-scaling instances to run jobs.
- 1 queue that sends jobs to EC2 instances in the `compute-1` node group.

The cluster requires additional AWS resources, such as security groups, IAM roles, and EC2 launch templates, which aren't shown in the diagram.

> **Note**
>
> We recommend that you complete the command line steps in this topic in a Bash shell. If you aren't using a Bash shell, some script commands such as line continuation characters and the way variables are set and used require adjustment for your shell. Additionally, the quoting and escaping rules for your shell might be different. For more information, see Quotation marks and literals with strings in the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

Topics

Prerequisites for getting started with AWS PCS

Refer to the following topics to prepare your AWS account and local development environment for AWS PCS.

Topics

- [Sign up for AWS and create an administrative user]
- [Install the AWS CLI]
- [Required IAM permissions for AWS PCS]

Sign up for AWS and create an administrative user

Complete the following tasks to set up for AWS Parallel Computing Service (AWS PCS).

Topics

- Sign up for an AWS account
- Create a user with administrative access

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

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

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see [Add groups] in the *AWS IAM Identity Center User Guide*.

Install the AWS CLI

You must use the latest version of the AWS CLI. For information, see Install or update to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

You must configure the AWS CLI. For more information, see Configure the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

Enter the following command at a command prompt to check your AWS CLI; it should display help information.

```
aws pcs help
```

Required IAM permissions for AWS PCS

The IAM security principal that you're using must have permissions to work with AWS PCS IAM roles, service linked roles, AWS CloudFormation, a VPC, and related resources. For more information, see Identity and Access Management for AWS Parallel Computing Service, and Create a service-linked role in the *AWS Identity and Access Management User Guide*. You must complete all steps in this guide as the same user. To check the current user, run the following command:

```
aws sts get-caller-identity
```

Create a VPC and subnets for AWS PCS

You can create a VPC and subnets with a CloudFormation template. Use the following URL to download the CloudFormation template, then upload the template in the AWS CloudFormation console to create a new CloudFormation stack. For more information, see Using the AWS CloudFormation console in the *AWS CloudFormation User Guide*.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/net/hpc_large_scale/assets/
main.yaml
```

With the template open in the AWS CloudFormation console, enter the following options. You can use the default values provided in the template.

- Under **Provide a stack name**:
  - Under **Stack name**, enter:

    ```
    hpc-networking
    ```

- Under **Parameters**:
  - Under **VPC**:
    - Under **CidrBlock**, enter:

      ```
      10.3.0.0/16
      ```

  - Under **Subnets A**:
    - Under **CidrPublicSubnetA**, enter:

      ```
      10.3.0.0/20
      ```

    - Under **CidrPrivateSubnetA**, enter:

      ```
      10.3.128.0/20
      ```

  - Under **Subnets B**:
    - Under **CidrPublicSubnetB**, enter:

      ```
      10.3.16.0/20
      ```

    - Under **CidrPrivateSubnetB**, enter:

      ```
      10.3.144.0/20
      ```

  - Under **Subnets C**:
    - For **ProvisionSubnetsC**, select **True**
    - Under **CidrPublicSubnetC**, enter:

      ```
      10.3.32.0/20
      ```


    - Under **CidrPrivateSubnetC**, enter:

```
10.3.160.0/20
```

- Under **Capabilities**:
  - Check the box for **I acknowledge that AWS CloudFormation might create IAM resources**.

Monitor the status of the CloudFormation stack. When it reaches `CREATE_COMPLETE`, find the ID for the default security group in the new VPC. You use the ID later in the tutorial.

Find the default security group for the cluster VPC

To find the ID for the default security group in the new VPC, follow this procedure:

- Navigate to the Amazon VPC console.
- Under the **VPC Dashboard**, select **Filter by VPC**.
  - Choose the VPC where the name starts with `hpc-networking`.
  - Under **Security**, choose **Security groups**.
- Find the **Security group ID** for the group named `default`. It has the description `default VPC security group`. You use the ID later to configure EC2 launch templates.

Create security groups for AWS PCS

AWS PCS relies on security groups to manage network traffic into and out of a cluster and its compute node groups. For detailed information on this topic, see Security group requirements and considerations.

In this step, you will use an CloudFormation template to create two security groups.

- A cluster security group, which enables communications between AWS PCS controller, compute nodes, and login nodes.
- An inbound SSH security group, which you can optionally add to your login nodes to support SSH access

Create the security groups for AWS PCS

You can use a CloudFormation template to create the security groups. Use the following URL to download the CloudFormation template, then upload the template in the AWS CloudFormation console to create a new CloudFormation stack. For more information, see Using the AWS CloudFormation console in the *AWS CloudFormation User Guide*.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/pcs/getting_started/assets/pcs-cluster-sg.yaml
```

With the template open in the AWS CloudFormation console, enter the following options. Note that some options will be pre-populated in the template — you can simply leave them as the default values.

- Under **Provide a stack name**
  - Under **Stack name**, enter:

    ```
    getstarted-sg
    ```
- Under **Parameters**
  - Under **VpcId**, choose the VPC where the name starts with `hpc-networking`.
  - (Optional) Under **ClientIpCidr**, enter a more restrictive IP range for the inbound SSH security group. We recommend that you restrict this with your own IP/subnet (x.x.x.x/32 for your own ip or x.x.x.x/24 for range. Replace x.x.x.x with your own PUBLIC IP. You can get your public IP using tools such as https://ifconfig.co/)

Monitor the status of the CloudFormation stack. When it reaches `CREATE_COMPLETE` the security group resources are ready.

There are two security groups created, with the names:

- `cluster-getstarted-sg` – this is the cluster security group
- `inbound-ssh-getstarted-sg` – this is a security group to allow inbound SSH access

Create a cluster in AWS PCS

In AWS PCS, a cluster is a persistent resource for managing resources and running workloads. You create a cluster for a specific scheduler (AWS PCS currently supports Slurm) in a subnet of a new or existing VPC. The cluster accepts and schedules jobs, and also launches the compute nodes (EC2 instances) that process those jobs.

To create your cluster

1. Open the AWS PCS console and choose **Create cluster**.
2. In the **Cluster details** section, enter the following fields:
   - **Cluster name** – Enter `get-started`
   - **Scheduler** – Select **Slurm Version 24.05**
   - **Controller size** – Select **Small**
3. In the **Networking** section, select values for the following fields:
   - **VPC** – Choose the VPC named `hpc-networking:Large-Scale-HPC`
   - **Subnet** – Select the subnet where the name starts with `hpc-networking:PrivateSubnetA`
   - **Security groups** – Select the cluster security group named `cluster-getstarted-sg`
4. Choose **Create cluster**.

> **Note**
>
> The **Status** field shows **Creating** while the cluster is being provisioned. Cluster creation can take several minutes.

Create shared storage for AWS PCS in Amazon Elastic File System

Amazon Elastic File System (Amazon EFS) is an AWS service that provides serverless, fully elastic file storage so that you can share file data without provisioning or managing storage capacity and performance. For more information, see What is Amazon Elastic File System? in the *Amazon Elastic File System User Guide*.

The AWS PCS demonstration cluster uses an EFS file system to provide a shared home directory between the cluster nodes. Create an EFS file system in the same VPC as your cluster.

To create your Amazon EFS file system

1. Go to the Amazon EFS console.

2. Make sure it's set to the same AWS Region where you will try AWS PCS.
3. Choose **Create file system**.
4. On the **Create file system** page, set the following parameters:
   - For **Name**, enter `getstarted-efs`
   - Under **Virtual Private Cloud (VPC)**, choose the VPC named `hpc-networking:Large-Scale-HPC`
   - Choose **Create**. This returns you to the **File systems** page.
5. Make a note of the **File system ID** for the `getstarted-efs` file system. You use this information later.

Create shared storage for AWS PCS in Amazon FSx for Lustre

Amazon FSx for Lustre makes it easy and cost-effective to launch and run the popular, high-performance Lustre file system. You use Lustre for workloads where speed matters, such as machine learning, high performance computing (HPC), video processing, and financial modeling. For more information, see [What is Amazon FSx for Lustre?] in the *Amazon FSx for Lustre User Guide*.

The AWS PCS demonstration cluster can use an FSx for Lustre file system to provide a high-performance shared directory between the cluster nodes. Create an FSx for Lustre file system in the same VPC as your cluster.

To create your FSx for Lustre file system

1. Go to the Amazon FSx console.
2. Make sure the console is set to use the same AWS Region as your cluster.
3. Choose **Create file system**.
   - For **Select file system type**, choose **Amazon FSx for Lustre**, then choose **Next**.
4. On the **Specify file system details** page, set the following parameters:
   - Under **File system details**
     - For **Name**, enter `getstarted-fsx`
     - For **Deployment and storage type**, choose **Persistent, SSD**
     - For **Throughput per unit of storage**, choose **125 MB/s/TiB**
     - For **Storage capacity**, enter 1.2 TiB

    - For **Metadata Configuration**, choose **Automatic**
    - For **Data compression type**, choose **LZ4**
  - Under **Network & security**
    - For **Virtual Private Cloud (VPC)**, choose the VPC named `hpc-networking:Large-Scale-HPC`
    - For **VPC Security Groups**, leave the security group named `default`
    - For **Subnet**, choose the subnet where the name starts with `hpc-networking:PrivateSubnetA`
  - Leave the other options set to their default values.
  - Choose **Next**.
5. On the **Review and create** page, choose **Create file system**. This returns you to the **File systems** page.
6. Navigate to the details page for the FSx for Lustre file system you created.
7. Make a note of the **File system ID** and the **Mount name**. You use this information later.

> **Note**
>
> The **Status** field shows **Creating** while the file system is being provisioned. File system creation can take several minutes. Wait until it completes before proceeding with the rest of the tutorial.

Create compute node groups in AWS PCS

A compute node group is virtual collection of compute nodes (EC2 instances) that AWS PCS launches and manages. When you define a compute node group, you specify common traits such as EC2 instance types, minimum and maximum instance count, target VPC subnets, preferred purchase option, and custom launch configuration. AWS PCS efficiently launches, manages, and terminates compute nodes in a compute node group, according to these settings. The demonstration cluster uses a compute node group to provide login nodes for user access, and a separate compute node group to process jobs. The following topics describe the procedures to set up these compute node groups in your cluster.

Topics

- [Create an instance profile for AWS PCS]
- [Create launch templates for AWS PCS]
- [Create compute node group for login nodes in AWS PCS]
- [Create compute node group for running compute jobs in AWS PCS]

Create an instance profile for AWS PCS

Compute node groups require an instance profile when they are created. If you use the AWS Management Console to create a role for Amazon EC2, the console automatically creates an instance profile and gives it the same name as the role. For more information, see Using instance profiles in the *AWS Identity and Access Management User Guide*.

In the following procedure, you use the AWS Management Console to create a role for Amazon EC2, which also creates the instance profile for your compute node groups.

To create the role and instance profile

- Navigate to the IAM console.
- Under **Access management**, choose **Policies**.
  - Choose **Create policy**.
  - Under **Specify permissions**, for **Policy editor**, choose **JSON**.
  - Replace the contents of the text editor with the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "pcs:RegisterComputeNodeGroupInstance"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

  - Choose **Next**.
  - Under **Review and create**, for **Policy name**, enter `AWSPCS-getstarted-policy`.

  - Choose **Create policy**.
- Under **Access management**, choose **Roles**.
- Choose **Create role**.
- Under **Select trusted entity**:
  - For **Trusted entity type**, select **AWS service**
  - Under **Use case**, select **EC2**.
    - Then, under **Choose a use case** for the specified service, choose **EC2**.
  - Choose **Next**.
- Under **Add permissions**:
  - In **Permissions policies**, search for **AWSPCS-getstarted-policy**.
  - Check the box beside **AWSPCS-getstarted-policy** to add it to the role.
  - In **Permissions policies**, search for **AmazonSSMManagedInstanceCore**.
  - Check the box beside **AmazonSSMManagedInstanceCore** to add it to the role.
  - Choose **Next**.
- Under **Name, review, and create**:
  - Under **Role details**:
    - For **Role name**, enter `AWSPCS-getstarted-role`.
  - Choose **Create role**.

Create launch templates for AWS PCS

When you create a compute node group, you provide an EC2 launch template that AWS PCS uses to configure EC2 instances it launches. This includes settings such as security groups and scripts that run when the instance launches.

In this step, one CloudFormation template will be used to create two EC2 launch templates. One template will be used to create login nodes, and the other will be used to create compute nodes. The key difference between them is that the login nodes can be configured to allow inbound SSH access.

Access the CloudFormation template

Use the following URL to download the CloudFormation template, then upload the template in the AWS CloudFormation console to create a new CloudFormation stack. For more information, see Using the AWS CloudFormation console in the *AWS CloudFormation User Guide*.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/pcs/getting_started/assets/pcs-lt-efs-fsxl.yaml
```

Use the CloudFormation template to create EC2 launch templates

Use the following procedure to complete the CloudFormation template in the AWS CloudFormation console

- Under **Provide a stack name**:
  - Under **Stack name**, enter `getstarted-lt`.
- Under **Parameters**:
  - Under **Security**
    - For **VpcSecurityGroupId**, select the security group named `default` in your cluster VPC.
    - For **ClusterSecurityGroupId**, select the group named `cluster-getstarted-sg`
    - For **SshSecurityGroupId**, select the group named `inbound-ssh-getstarted-sg`
    - For **SshKeyName**, select your preferred SSH key pair.
  - Under **File systems**
    - For **EfsFilesystemId**, enter the file system ID from the EFS file system you created earlier in the tutorial.
    - For **FSxLustreFilesystemId**, enter the file system ID from the FSx for Lustre file system you created earlier in the tutorial.
    - For **FSxLustreFilesystemMountName**, enter the mount name for that same FSx for Lustre file system.
- Choose **Next**, then choose **Next** again.
- Choose **Submit**.

Monitor the status of the CloudFormation stack. When it reaches `CREATE_COMPLETE` the launch template is ready to be used.

> **Note**
>
> To see all the resources the CloudFormation template created, open the AWS CloudFormation console. Choose the `getstarted-lt` stack and then choose the **Resources** tab.

Create compute node group for login nodes in AWS PCS

A compute node group is virtual collection of compute nodes (EC2 instances) that AWS PCS launches and manages. When you define a compute node group, you specify common traits such as EC2 instance types, minimum and maximum instance count, target VPC subnets, preferred purchase option, and custom launch configuration. AWS PCS efficiently launches, manages, and terminates compute nodes in a compute node group, according to these settings.

In this step, you will launch a static compute node group that provides interactive access to the cluster. You can use SSH or Amazon EC2 Systems Manager (SSM) to log in to it, then run shell commands and manage Slurm jobs.

To create the compute node group

- Open the AWS PCS console and navigate to **Clusters**.
- Select the cluster named `get-started`
- Navigate to **Compute node groups** and choose **Create**.
- In the **Compute node group setup** section, provide the following:
  - **Compute node group name** – Enter `login`.
- Under **Computing configuration**, enter or select these values:
  - **EC2 launch template** – Choose the launch template where the name is `login-getstarted-lt`
  - **IAM instance profile** – Choose the instance profile named `AWSPCS-getstarted-role`
  - **Subnets** – Select the subnet where the name starts with `hpc-networking:PublicSubnetA`.
  - **Instances** – Select `c6i.xlarge`.
  - **Scaling configuration** – For **Min. instance count**, enter 1. For **Max. instance count**, enter 1.
- Under **Additional settings**, specify the following:
  - **AMI ID** – Select an AMI you want to use, that has a name in the following format:

```
aws-pcs-sample_ami-amzn2-platform-slurm-version
```

For more information about the sample AMIs, see [Using sample Amazon Machine Images (AMIs) with AWS PCS](#).

- Choose **Create compute node group**.

The **Status** field shows **Creating** while the compute node group is being provisioned. You can proceed to the next step in the tutorial while it is in progress.

Create compute node group for running compute jobs in AWS PCS

In this step, you will launch a compute node group that scales elastically to run jobs submitted to the cluster.

To create the compute node group

- Open the [AWS PCS console](#) and navigate to **Clusters**.
- Select the cluster named `get-started`
- Navigate to **Compute node groups** and choose **Create**.
- In the **Compute node group setup** section, provide the following:
  - **Compute node group name** – Enter `compute-1`.
- Under **Computing configuration**, enter or select these values:
  - **EC2 launch template** – Choose the launch template where the name is `compute-getstarted-lt`
  - **IAM instance profile** – Choose the instance profile named `AWSPCS-getstarted-role`
  - **Subnets** – Select the subnet where the name starts with `hpc-networking:PrivateSubnetA`.
  - **Instances** – Select `c6i.xlarge`.
  - **Scaling configuration** – For **Min. instance count**, enter `0`. For **Max. instance count**, enter `4`.
- Under **Additional settings**, specify the following:
  - **AMI ID** – Select an AMI you want to use, that has a name in the following format:

```
aws-pcs-sample_ami-amzn2-platform-slurm-version
```

For more information about the sample AMIs, see Using sample Amazon Machine Images (AMIs) with AWS PCS.

- Choose **Create compute node group**.

The **Status** field shows **Creating** while the compute node group is being provisioned.

> **Important**
>
> Wait for the **Status** field to show **Active** before proceeding to the next step in this tutorial.

Create a queue to manage jobs in AWS PCS

You submit a job to a queue to run it. The job remains in the queue until AWS PCS schedules it to run on a compute node group. Each queue is associated with one or more compute node groups, which provide the necessary EC2 instances to do the processing.

In this step, you will create a queue that uses the compute node group to process jobs.

To create a queue

- Open the AWS PCS console.
- Select the cluster named `get-started`.
- Navigate to **Compute node groups** and make sure the status of the `compute-1` group is **Active**.

  > **Important**
  >
  > The status of the `compute-1` group must be **Active** before you proceed to the next step.

- Navigate to **Queues** and choose **Create queue**.
  - In the **Queue configuration** section, provide the following values:
    - **Queue name** – Enter the following: demo
    - **Compute node groups** – Select the compute node group named `compute-1`.
- Choose **Create queue**.

The **Status** field shows **Creating** while the queue is being created.

> **Important**
>
> Wait for the **Status** field to show **Active** before proceeding to the next step in this tutorial.

Connect to your AWS PCS cluster

After the status of the `login` compute node group becomes **Active**, you can connect to the EC2 instance it created.

To connect to the login node

- Open the AWS PCS console and navigate to **Clusters**.
- Select the cluster named `get-started`.
- Choose **Compute node groups**.
- Navigate to the compute node group named `login`.
- Find the **Compute node group ID**.
- In another browser window or tab, open the Amazon EC2 console.
  - Choose **Instances**.
  - Search for EC2 instances with the following tag. Replace *`node-group-id`* with the value of the **Compute node group ID** from the previous step. There should be 1 instance.

    ```
    aws:pcs:compute-node-group-id=node-group-id
    ```

  - Connect to the EC2 instance. You can use Session Manager or SSH.

    Session Manager

    - Select the instance.
    - Choose **Connect**.
    - Under **Connect to instance**, select **Session Manager**.
    - Choose **Connect**.
    - Choose **Connect**. An interactive terminal launches in your browser.

    SSH

    - Select the instance.
    - Choose **Connect**.

- Under **Connect to instance**, select **SSH client**.
- Follow the instructions provided by the console.

> **Note**
>
> The the user name for the instance is **ec2-user** not root.

Explore the cluster environment in AWS PCS

After you have logged into the cluster, you can run shell commands. For instance, you can change users, work with data on shared filesystems, and interact with Slurm.

Change user

If you have logged in to the cluster using Session Manager, you may be connected as ssm-user. This is an special user that is created for Session Manager. Switch to the default user on Amazon Linux 2 using the following command. You will not need to do this if you connected using SSH.

```
sudo su - ec2-user
```

Work with shared file systems

You can confirm that the EFS filesystem and FSx for Lustre file systems are available with the command df -h. Output on your cluster should resemble the following:

```
[ec2-user@ip-10-3-6-103 ~]$ df -h
Filesystem                  Size  Used Avail Use% Mounted on
devtmpfs                    3.8G     0  3.8G   0% /dev
tmpfs                       3.9G     0  3.9G   0% /dev/shm
tmpfs                       3.9G  556K  3.9G   1% /run
tmpfs                       3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/nvme0n1p1               24G   18G  6.6G  73% /
127.0.0.1:/                 8.0E     0  8.0E   0% /home
10.3.132.79@tcp:/zlshxbev  1.2T  7.5M  1.2T   1% /shared
tmpfs                       780M     0  780M   0% /run/user/0
tmpfs                       780M     0  780M   0% /run/user/1000
```

The /home filesystem mounts 127.0.0.1 and has a very large capacity. This is the EFS file system that you created earlier in the tutorial. Any files written here will be available under /home on all nodes in the cluster.

The /shared filesystem mounts a private IP and has a capacity of 1.2 TB. This is the FSx for Lustre file system that you created earlier in the tutorial. Any files written here will be available under /shared on all nodes in the cluster.

Interact with Slurm

Topics

List queues and nodes

You can list the queues and the nodes they are associated with using `sinfo`. Output from your cluster should resemble the following:

```
[ec2-user@ip-10-3-6-103 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
demo         up   infinite      4  idle~ compute-1-[1-4]
[ec2-user@ip-10-3-6-103 ~]$
```

Note the partition named demo. Its status is up and it has a maximum of 4 nodes. It is associated with nodes in the `compute-1` node group. If you edit the compute node group and increase the maximum number of instances to 8, the number of nodes would read 8 and the node list would read `compute-1-[1-8]`. If you created a second compute node group named `test` with 4 nodes, and added it to the demo queue, those nodes would show up in the node list as well.

Show jobs

You can list all jobs, in any state, on the system with `squeue`. Output from your cluster should resemble the following:

```
[ec2-user@ip-10-3-6-103 ~]$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```

Try running `squeue` again later, when you have a Slurm job pending or running.

Run a single node job in AWS PCS

To run a job using Slurm, you prepare a submission script specifying job requirements and submit it to a queue with the `sbatch` command. Typically, this is done from a shared directory so the login and compute nodes have a common space for accessing files.

Connect to the login node of your cluster and run the following commands at its shell prompt.

- Become the default user. Change to the shared directory.

```
sudo su - ec2-user
cd /shared
```

- Use the following commands to create an example job script:

```
cat << EOF > job.sh
#!/bin/bash
#SBATCH -J single
#SBATCH -o single.%j.out
#SBATCH -e single.%j.err

echo "This is job \${SLURM_JOB_NAME} [\${SLURM_JOB_ID}] running on \
${SLURMD_NODENAME}, submitted from \${SLURM_SUBMIT_HOST}" && sleep 60 && echo "Job
 complete"
EOF
```

- Submit the job script to the Slurm scheduler:

```
sbatch -p demo job.sh
```

- When the job is submitted, it will return a job ID as a number. Use that ID to check the job status. Replace *job-id* in the following command with the number returned from `sbatch`.

```
squeue --job job-id
```

Example

```
squeue --job 1
```

The `squeue` command returns output similar to the following:

```
JOBID PARTITION NAME USER     ST TIME NODES NODELIST(REASON)
1     demo      test ec2-user CF 0:47 1     compute-1
```

- Continue to check the status of the job until it reaches the R (running) status. The job is done when `squeue` doesn't return anything.
- Inspect the contents of the `/shared` directory.

  ```
  ls -alth /shared
  ```

  The command output is similar to the following:

  ```
  -rw-rw-r- 1 ec2-user ec2-user 107 Mar 19 18:33 single.1.out
  -rw-rw-r- 1 ec2-user ec2-user 0 Mar 19 18:32 single.1.err
  -rw-rw-r- 1 ec2-user ec2-user 381 Mar 19 18:29 job.sh
  ```

  The files named `single.1.out` and `single.1.err` were written by one of your cluster's compute nodes. Because the job was run in a shared directory (`/shared`), they are also available on your login node. This is why you configured an FSx for Lustre file system for this cluster.
- Inspect the contents of the `single.1.out` file.

  ```
  cat /shared/single.1.out
  ```

  The output is similar to the following:

  ```
  This is job test [1] running on compute-1, submitted from ip-10-3-13-181
  Job complete
  ```

Run a multi-node MPI job with Slurm in AWS PCS

These instructions demonstrate using Slurm to run a message passing interface (MPI) job in AWS PCS.

Run the following commands at a shell prompt of your login node.

- Become the default user. Change to its home directory.

  ```
  sudo su - ec2-user
  ```

```
cd ~/
```

- Create source code in the C programming language.

```
cat > hello.c << EOF
// * mpi-hello-world - https://www.mpitutorial.com
// Released under MIT License
//
// Copyright (c) 2014 MPI Tutorial.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to
// deal in the Software without restriction, including without limitation the
// rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
// sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
// FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
// DEALINGS IN THE SOFTWARE.

#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();
}
EOF
```

- Load the OpenMPI module.

```
module load openmpi
```

- Compile the C program.

```
mpicc -o hello hello.c
```

- Write a Slurm job submission script.

```
cat > hello.sh << EOF
#!/bin/bash
#SBATCH -J multi
#SBATCH -o multi.out
#SBATCH -e multi.err
#SBATCH --exclusive
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1

srun $HOME/hello
EOF
```

- Change to the shared directory.

```
cd /shared
```

- Submit the job script.

```
sbatch -p demo ~/hello.sh
```

- Use `squeue` to monitor the job until it's done.
- Check the contents of `multi.out`:

```
cat multi.out
```

The output is similar to the following. Note that each rank has its own IP address because it ran on a different node.

```
Hello world from processor ip-10-3-133-204, rank 0 out of 4 processors
Hello world from processor ip-10-3-128-219, rank 2 out of 4 processors
Hello world from processor ip-10-3-141-26, rank 3 out of 4 processors
Hello world from processor ip-10-3-143-52, rank 1 out of 4 processor
```

Delete your AWS resources for AWS PCS

After you are done with the cluster and node groups that you created for this tutorial, you should delete the resources that you created.

> **Important**
>
> You get billing charges for all resources running in your AWS account

To delete AWS PCS resources that you created for this tutorial

- Open the AWS PCS console.
- Navigate to the cluster named **get-started**.
- Navigate to the **Queues** section.
- Select the queue named **demo**.
- Choose **Delete**.

> **Important**
>
> Wait until the queue has been deleted before proceeding.

- Navigate to the **Compute node groups** section.
- Select the compute node group named **compute-1**.
- Choose **Delete**.
- Select the compute node group named **login**.
- Choose **Delete**.

> **Important**
>
> Wait until both compute node groups have been deleted before proceeding.

- In the cluster detail page for **get-started**, choose **Delete**.

> **Important**
>
> Wait until the cluster has been deleted before proceeding with subsequent steps.

To delete other AWS resources you created for this tutorial

- Open the [IAM console](#).
  - Choose **Roles**.
  - Select the role named **AWSPCS-getstarted-role** then choose **Delete**.
  - After the role has been deleted, choose **Policies**.
  - Select the policy named **AWSPCS-getstarted-policy** then choose **Delete**.
- Open the [AWS CloudFormation console](#).
  - Select the stack named **getstarted-lt**.
  - Choose **Delete**.

  > **Important**
  >
  > Wait for the stack to delete before proceeding.

- Open the Amazon EFS console.
  - Choose **File systems**.
  - Select the file system named **getstarted-efs**.
  - Choose **Delete**.

  > **Important**
  >
  > Wait for the file system to delete before proceeding.

- Open the Amazon FSx console.
  - Choose **File systems**.
  - Select the file system named **getstarted-fsx**.
  - Choose **Delete**.

  > **Important**
  >
  > Wait for the file system to delete before proceeding.

- Open the AWS CloudFormation console.
  - Select the stack named **getstarted-sg**.
  - Choose **Delete**.
- Open the AWS CloudFormation console.
  - Select the stack named **hpc-networking**.
  - Choose **Delete**.

AWS PCS clusters

An AWS PCS cluster consists of the following components:

- Managed instances of the HPC system scheduler software, such as the Slurm control daemon (`slurmctld`).
- Components that integrate with the HPC system scheduler to provision and manage Amazon EC2 instances.
- Components that integrate with the HPC system scheduler to transmit logs and metrics to Amazon CloudWatch.

These components run in an account managed by AWS. They work together to manage Amazon EC2 instances in your customer account. AWS PCS provisions elastic network interfaces in your Amazon VPC subnet to provide connectivity from the scheduler software to Amazon EC2 instances (for example, to support scheduling batch jobs on them and enabling users to run scheduler commands to list and manage those jobs).

Topics

Creating a cluster in AWS Parallel Computing Service

This topic provides an overview of available options and describes what to consider when you create a cluster in AWS Parallel Computing Service (AWS PCS). If this is your first time creating an AWS PCS cluster, we recommend you follow Getting started with AWS Parallel Computing Service. The tutorial can help you create a working HPC system without expanding into all the available options and system architectures that are possible.

Prerequisites

- An existing VPC and subnet that meet AWS PCS Networking requirements. Before you deploy a cluster for production use, we recommend that you have a thorough understanding of the VPC and subnet requirements. To create a VPC and subnet, see Creating a VPC for your AWS PCS cluster.

- An IAM principal with permissions to create and manage AWS PCS resources. For more information, see Identity and Access Management for AWS Parallel Computing Service.

Create an AWS PCS cluster

You can use the AWS Management Console or AWS CLI to create a cluster.

AWS Management Console

To create a cluster

1. Open the AWS PCS console at https://console.aws.amazon.com/pcs/home#/clusters and choose **Create cluster**.
2. In the **Cluster setup** section, enter the following fields:
   - **Cluster name** – A name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 40 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - **Scheduler** – Choose a scheduler and version. AWS PCS currently supports Slurm 24.05 and 23.11. For more information, see Slurm versions in AWS PCS.
   - **Controller size** – Choose a size for your controller. This determines how many concurrent jobs and compute nodes can be managed by the AWS PCS cluster. You can only set the controller size when the cluster is created. For more information on sizing, see Cluster size in AWS PCS.
3. In the **Networking** section, select values for the following fields:
   - **VPC** – Choose an existing VPC that meets AWS PCS requirements. For more information, see AWS PCS VPC and subnet requirements and considerations. After you create the cluster, you can't change its VPC. If no VPCs are listed, you must create one first.
   - **Subnet** – All available subnets in the selected VPC are listed. Choose two in different Availability Zones. Each subnet must meet the AWS PCS subnet requirements. For more information, see AWS PCS VPC and subnet requirements and considerations. We recommend you select a private subnet to avoid exposing your scheduler endpoints to the public internet.

- **Security groups** – Specify the security group(s) that you want AWS PCS to associate with the network interfaces it creates for your cluster. You must select at least one security group that allows communication between your cluster and its compute nodes. For more information, see Security group requirements and considerations.

4. (Optional) Under **Encryption**, you can define a custom key to encrypt your controller data by setting these fields:

   - **KMS key ID** – Leave as `aws/pcs` to use the KMS key that PCS creates. Select an existing KMS key alias to use a custom KMS key. Note that the account used to create the cluster must have `kms:Decrypt` privileges on the custom KMS key.

5. (Optional) In the **Slurm configuration** section, you can specify Slurm configuration options that override defaults set by AWS PCS:

   - **Scale down idle time** – This controls how long dynamically-provisioned compute nodes stay active after jobs placed on them complete or terminate. Setting this to a longer value can make it more likely that a subsequent job can run on the node, but may lead to increased costs. A shorter value will decrease costs, but may increase the proportion of time your HPC system spends provisioning nodes as opposed to running jobs on them.
   - **Prolog** – This is a fully-qualified path to a prolog scripts directory on your compute node group instances. This corresponds to the Prolog setting in Slurm. Note that this must be a directory, not a path to a specific executable.
   - **Epilog** – This is a fully-qualified path to an epilog scripts directory on your compute node group instances. This corresponds to the Epilog setting in Slurm. Note that this must be a directory, not a path to a specific executable.
   - **Select type parameters** – This helps control the resource selection algorithm used by Slurm. Setting this value to `CR_CPU_Memory` will activate memory-aware scheduling, while setting it to `CR_CPU` will activate CPU-only scheduling. This parameter corresponds to the SelectTypeParameters setting in Slurm where `SelectType` is set to `select/cons_tres` by AWS PCS.

6. (Optional) Under **Tags**, add any tags to your AWS PCS cluster.

7. Choose **Create cluster**. The **Status** field shows `Creating` while the AWS PCS creates the cluster. This process can take several minutes.

> **Important**
>
> There can only be 1 cluster in a `Creating` state per AWS Region per AWS account. AWS PCS returns an error if there is already a cluster in a `Creating` state when you try to create a cluster.

AWS CLI

To create a cluster

1. Create your cluster with the command that follows. Before running the command, make the following replacements:

   - Replace *`region`* with the ID of the AWS Region that you want to create your cluster in, such as `us-east-1`.
   - Replace *`my-cluster`* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 40 characters. The name must be unique within the AWS Region and AWS account where you're creating the cluster.
   - Replace *`24.05`* with any supported version of Slurm.

     > **Note**
     >
     > AWS PCS currently supports Slurm 24.05 and 23.11.

   - Replace *`SMALL`* with any supported cluster size. This determines how many concurrent jobs and compute nodes can be managed by the AWS PCS cluster. It can only be set when the cluster is created. For more information on sizing, see Cluster size in AWS PCS.
   - Replace the value for `subnetIds` with your own. We recommend you select a private subnet to avoid exposing your scheduler endpoints to the public internet.
   - Specify the `securityGroupIds` that you want AWS PCS to associate with the network interfaces it creates for your cluster. The security groups must be in the same VPC as the cluster. You must select at least one security group that allows communication between your cluster and its compute nodes. For more information, see Security group requirements and considerations.

- Optionally, you can fine-tune Slurm behavior by adding a `--slurm-configration` option. For example, you can set the scale-down idle time to 60 minutes (3600 seconds) with `--slurm configuration scaleDownIdeTime=3600`.
- Optionally, you can provide a custom KMS key to encrypt your controller's data using `--kms-key-id` *`kms-key`*. Replace *`kms-key`* with an existing KMS ARN, key ID, or alias. Note that the account used to create the cluster must have `kms:Decrypt` privileges on the custom KMS key.

```
aws pcs create-cluster --region region \
    --cluster-name my-cluster \
    --scheduler type=SLURM,version=24.05 \
    --size SMALL \
    --networking subnetIds=subnet-ExampleId1,securityGroupIds=sg-ExampleId1
```

2. It can take several minutes to provision the cluster. You can query the status of your cluster with the following command. Don't proceed to creating queues or compute node groups until the cluster's status field is `ACTIVE`.

```
aws pcs get-cluster --region region --cluster-identifier my-cluster
```

> **Important**
>
> There can only be 1 cluster in a `Creating` state per AWS Region per AWS account. AWS PCS returns an error if there is already a cluster in a `Creating` state when you try to create a cluster.

Recommended next steps for your cluster

- Add compute node groups.
- Add queues.
- Enable logging.

Deleting a cluster in AWS PCS

This topic provides an overview of how to delete an AWS PCS cluster.

Considerations when deleting an AWS PCS cluster

- All queues associated with the cluster must be deleted before the cluster can be deleted. For more information, see Deleting a queue in AWS PCS.
- All compute node groups associated with the cluster must be deleted before the cluster can be deleted. For more information, see Deleting a compute node group in AWS PCS.

Delete the cluster

You can use the AWS Management Console or AWS CLI to delete a cluster.

AWS Management Console

To delete a cluster

1. Open the AWS PCS console.
2. Select the cluster to delete.
3. Choose **Delete**.
4. The cluster **Status** field shows `Deleting`. It can take several minutes to complete.

AWS CLI

To delete a cluster

1. Use the following command to delete a cluster, with these replacements:
   - Replace *region-code* with the AWS Region your cluster is in.
   - Replace *my-cluster* with the name or ID of your cluster.

```
aws pcs delete-cluster --region region-code --cluster-identifier my-cluster
```

2. It can take several minutes to delete the cluster. You can check the status of your cluster with the following command.

```
aws pcs get-cluster --region region-code --cluster-identifier my-cluster
```

Cluster size in AWS PCS

AWS PCS provides highly available and secure clusters, while automating key tasks such as patching, node provisioning, and updates.

When you create a cluster, you select a size for it based on two factors:

- The number of compute nodes it will manage
- The number of active and queued jobs that you expect to run on the cluster

> **Important**
>
> You can't change the cluster size after you create the cluster. If you need to change the size, you must create a new cluster.

Slurm cluster size	Number of instances managed	Number of active and queued jobs
Small	Up to 32	Up to 256
Medium	Up to 512	Up to 8192
Large	Up to 2048	Up to 16384

Examples

- If your cluster will have up to 24 managed instances and run up to 100 jobs, choose **Small**.
- If your cluster will have up to 24 managed instances and run up to 1000 jobs, choose **Medium**.
- If your cluster will have up to 1000 managed instances and run up to 100 jobs, choose **Large**.
- If your cluster will have up to 1000 managed instances and run up to 10,000 jobs, choose **Large**.

Working with cluster secrets in AWS PCS

As part of creating a cluster, AWS PCS creates a cluster secret that is required to connect to the job scheduler on the cluster. You also create AWS PCS compute node groups, which define sets of instances to launch in response to scaling events. AWS PCS configures instances launched by those compute node groups with the cluster secret so they can connect to the job scheduler. There are cases where you might want to configure Slurm clients manually. Examples include building a persistent login node or setting up a workflow manager with job management capabilities.

AWS PCS stores the cluster secret as a managed secret with the prefix `pcs!` in AWS Secrets Manager. The cost of the secret is included in the charge for using AWS PCS.

> **Warning**
>
> Don't modify your cluster secret. AWS PCS won't be able to communicate with your cluster if you modify your cluster secret. AWS PCS doesn't support rotation of the cluster secret. You must create a new cluster if you need to modify your cluster secret.

Contents

Use AWS Secrets Manager to find the cluster secret

AWS Management Console

1. Navigate to the Secrets Manager console.
2. Choose **Secrets**, then search for the `pcs!` prefix.

> **Note**
>
> A AWS PCS cluster secret has a name in the form `pcs!slurm-secret-`*`cluster-id`* where *`cluster-id`* is the AWS PCS cluster ID.

AWS CLI

Each AWS PCS cluster secret is also tagged with `aws:pcs:`*`cluster-id`*. You can get the secret ID for a cluster with the command that follows. Make these substitutions before running the command:

- Replace *`region`* with the AWS Region to create your cluster in, such as `us-east-1`.
- Replace *`cluster-id`* with the ID of the AWS PCS cluster to find the cluster secret for.

```
aws secretsmanager list-secrets \
    --region region  \
    --filters Key=tag-key,Values=aws:pcs:cluster-id \
            Key=tag-value,Values=cluster-id
```

Use AWS PCS to find the cluster secret

You can use the AWS CLI to find the ARN for an AWS PCS cluster secret. Enter the command that follows, making the following substitutions:

- Replace *`region`* with the AWS Region to create your cluster in, such as `us-east-1`.
- Replace *`my-cluster`* with the name or identifier for your cluster.

```
aws pcs get-cluster --region region --cluster-identifier my-cluster
```

The following example output is from the `get-cluster` command. You can use `secretArn` and `secretVersion` together to get the secret.

```
{
    "cluster": {
        "name": "get-started",
        "id": "pcs_123456abcd",
        "arn": "arn:aws:pcs:us-east-1:111122223333:cluster/pcs_123456abcd",
        "status": "ACTIVE",
        "createdAt": "2024-12-17T21:03:52+00:00",
        "modifiedAt": "2024-12-17T21:03:52+00:00",
        "scheduler": {
            "type": "SLURM",
```

```
            "version": "24.05"
        },
        "size": "SMALL",
        "slurmConfiguration": {
            "authKey": {
                "secretArn": "arn:aws:secretsmanager:us-east-1:111122223333:secret:pcs!
slurm-secret-pcs_123456abcd-a12ABC",
                "secretVersion": "ef232370-d3e7-434c-9a87-ec35c1987f75"
            }
        },
        "networking": {
            "subnetIds": [
                "subnet-0123456789abcdef0"
            ],
            "securityGroupIds": [
                "sg-0123456789abcdef0"
            ]
        },
        "endpoints": [
            {
                "type": "SLURMCTLD",
                "privateIpAddress": "10.3.149.220",
                "port": "6817"
            }
        ]
    }
}
```


Get the Slurm cluster secret

You can use Secrets Manager to get the current base64-encoded version of a Slurm cluster secret The following example uses the AWS CLI. Make the following substitutions before running the command.

- Replace *region* with the AWS Region to create your cluster in, such as `us-east-1`.
- Replace *secret-arn* with the `secretArn` from an AWS PCS cluster.

```
aws secretsmanager get-secret-value \
    --region region \
    --secret-id 'secret-arn' \
    --version-stage AWSCURRENT \
```

```
    --query 'SecretString' \
    --output text
```

For information about how to use the Slurm cluster secret, see Using standalone instances as AWS PCS login nodes.

Permissions

You use an IAM principal to get the Slurm cluster secret. The IAM principal must have permission to read the secret. For more information, see Roles terms and concepts in the *AWS Identity and Access Management User Guide*.

The following sample IAM policy allows access to an example cluster secret.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSecretValueRetrievalAndVersionListing",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-east-1:012345678901:secret:pcs!
slurm-secret-s3431v9rx2-FN7tJF"
        }
    ]
}
```

AWS PCS compute node groups

An AWS PCS compute node group is a logical collection of nodes (Amazon EC2 instances). These nodes can be used to run computing jobs, as well as to provide interactive, shell-based access to an HPC system. A compute node group consists of rules for creating nodes, including which Amazon EC2 instances types to use, how many instances to run, whether to use Spot Instances or On-demand Instances, which subnets and security groups to use, and how to configure each instance when it launches. When those rules are updated, AWS PCS updates resources associated with the compute node group to match.

Topics

Creating a compute node group in AWS PCS

This topic provides an overview of available options and describes what to consider when you create a compute node group in AWS Parallel Computing Service (AWS PCS). If this is your first time creating a compute node group in AWS PCS, we recommend you follow the tutorial in Getting started with AWS Parallel Computing Service. The tutorial can help you create a working HPC system without expanding into all the available options and system architectures that are possible.

Prerequisites

- Sufficient service quotas to launch the desired number of EC2 instances in your AWS Region. You can use the AWS Management Console to check and request increases to your service quotas.
- An existing VPC and subnet(s) that meet AWS PCS networking requirements. We recommend that you thoroughly understand these requirements before you deploy a cluster for production use. For more information, see AWS PCS VPC and subnet requirements and considerations. You can also use a CloudFormation template to create a VPC and subnets. AWS provides an HPC recipe for the CloudFormation template. For more information, see aws-hpc-recipes on GitHub.
- An IAM instance profile with permissions to call the AWS PCS `RegisterComputeNodeGroupInstance` API action and access to any other AWS resources

required for your node group instances. For more information, see IAM instance profiles for AWS Parallel Computing Service.

- A launch template for your node group instances. For more information, see Using Amazon EC2 launch templates with AWS PCS.
- To create a compute node group that uses Amazon EC2 **Spot** instances, you must have the **AWSServiceRoleForEC2Spot** service-linked role in your AWS account. For more information, see Amazon EC2 Spot role for AWS PCS.

Create a compute node group in AWS PCS

You can create a compute node group using the AWS Management Console or the AWS CLI.

AWS Management Console

To create your compute node group using the console

1. Open the AWS PCS console.
2. Select the cluster where you want to create a compute node group. Navigate to **Compute node groups** and choose **Create**.
3. In the **Compute node group setup** section, provide a name for your node group. The name can only contain case-sensitive alphanumeric characters and hyphens. It must start with an alphabetic character and can't be longer than 25 characters. The name must be unique within the cluster.
4. Under **Computing configuration**, enter or select these values:

   a. **EC2 launch template** – Select a custom launch template to use for this node group. Launch templates can be used to customize network settings such as subnet, and security groups, monitoring configuration, and instance-level storage. If you don't have a launch template prepared, see Using Amazon EC2 launch templates with AWS PCS to learn how to create one.

   > **Important**
   >
   > AWS PCS creates a managed launch template for each compute node group. These are named `pcs-`*`identifier`*`-do-not-delete`. Don't select these

> when you create or update a compute node group, or the node group won't function correctly.

b. **EC2 launch template version** – You must select a version of your custom launch template. If you change the version later, you must update the compute node group to detect changes in the launch template. For more information, see Updating an AWS PCS compute node group.

c. **AMI ID** – if your launch template doesn't include an AMI ID, or if you want to override the value in the launch template, provide an AMI ID here. Note that the AMI used for the node group must be compatible with AWS PCS. You can also select a sample AMI provided by AWS. For more information on this topic, see Amazon Machine Images (AMIs) for AWS PCS.

d. **IAM instance profile** – Choose an instance profile for the node group. An instance profile grants the instance permissions to access AWS resources and services securely. If you don't have one prepared, see IAM instance profiles for AWS Parallel Computing Service to learn how to create one.

e. **Subnets** – Choose one or more subnets in the VPC where your AWS PCS cluster is deployed. If you select multiple subnets, EFA communications won't be available between nodes, and communication between nodes in different subnets might have increased latency. Make sure the subnets you specify here match any that you define in the EC2 launch template.

f. **Instances** – Choose one or more instance types to fulfill scaling requests in the node group. All instance types must have the same processor architecture (x86_64 or arm64) and number of vCPUs. If the instances have GPUs, all instance types must have the same number of GPUs.

g. **Scaling configuration** – Specify the minimum and maximum number of instances for the node group. You can define either a static configuration, where there is a fixed number of nodes running, or a dynamic configuration, where up to the maximum count of nodes can run. For a static configuration, set minimum and maximum to the same, greater than zero number. For a dynamic configuration, set minimum instances to zero and maximum instances to a number greater than zero. AWS PCS doesn't support compute node groups with a mix of static and dynamic instances.

5. (Optional) Under **Additional settings**, specify the following:

   a. **Purchase option** – select between Spot and On-demand instances.

b. **Allocation strategy** – if you have selected the Spot purchase option, you can specify how Spot capacity pools are chosen when launching instances in the node group. For more information, see Allocation strategies for Spot Instances in the *Amazon Elastic Compute Cloud User Guide*. This option has no effect if you have selected the On-demand purchase option.

6. (Optional) In the **Slurm custom settings** section, provide these values:

    a. **Weight** – This value sets the priority of nodes in the group for scheduling purposes. Nodes with lower weights have higher priority, and the units are arbitrary. For more information, see Weight in the Slurm documentation.

    b. **Real memory** – This value sets the size (in GB) of real memory on nodes in the node group. It is meant to be used in conjunction with the `CR_CPU_Memory` option in the Cluster Slurm configuration in AWS PCS. For more information, see RealMemory in the Slurm documentation.

7. (Optional) Under **Tags**, add any tags to your compute node group.

8. Choose **Create compute node group**. The **Status** field shows `Creating` while AWS PCS provisions the node group. This can take several minutes.

Recommended next step

- Add your node group to an queue in AWS PCS to enable it to process jobs.

AWS CLI

To create your compute node group using AWS CLI

Create your queue with the command that follows. Before running the command, make the following replacements:

1. Replace *`region`*with the ID of the AWS Region to create your cluster in, such as `us-east-1`.
2. Replace *`my-cluster`* with the name or `clusterId`of your cluster.
3. Replace *`my-node-group`*with the name for your compute node group. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 25 characters. The name must be unique within the cluster.

4. Replace *subnet-ExampleID1* with one or more subnets IDs from your cluster VPC.
5. Replace *lt-ExampleID1* with the ID for your custom launch template. If you don't have one prepared, see Using Amazon EC2 launch templates with AWS PCS to learn how to create one.

> **Important**
>
> AWS PCS creates a managed launch template for each compute node group. These are named `pcs-identifier-do-not-delete`. Don't select these when you create or update a compute node group, or the node group won't function correctly.

6. Replace *launch-template-version* with a specific launch template version. AWS PCS associates your node group with that specific version of the launch template.
7. Replace *arn:InstanceProfile*with the ARN of your IAM instance profile. If you don't have one prepared, see Using Amazon EC2 launch templates with AWS PCS for guidance.
8. Replace *min-instances* and *max-instances* with integer values. You can define either a static configuration, where there is a fixed number of nodes running, or a dynamic configuration, where up to the maximum count of nodes can run. For a static configuration, set minimum and maximum to the same, greater than zero number. For a dynamic configuration, set minimum instances to zero and maximum instances to a number greater than zero. AWS PCS doesn't support compute node groups with a mix of static and dynamic instances.
9. Replace *t3.large* with another instance type. You can add more instance types by specifying a list of `instanceType` settings. For example, *--instance-configs instanceType=c6i.16xlarge,instanceType=c6a.16xlarge*. All instance types must have the same processor architecture (x86_64 or arm64) and number of vCPUs. If the instances have GPUs, all instance types must have the same number of GPUs.

```
aws pcs create-compute-node-group --region region \
    --cluster-identifier my-cluster \
    --compute-node-group-name my-node-group \
    --subnet-ids subnet-ExampleID1 \
    --custom-launch-template id=lt-ExampleID1,version='launch-template-version' \
    --iam-instance-profile arn=arn:InstanceProfile \
    --scaling-config minInstanceCount=min-instances,maxInstanceCount=max-instance \
    --instance-configs instanceType=t3.large
```

There are several optional configuration settings you can add to the `create-compute-node-group` command.

- You can specify `--amiId` if your custom launch template doesn't include a reference to an AMI, or if you wish to override that value. Note that the AMI used for the node group must be compatible with AWS PCS. You can also select a sample AMI provided by AWS. For more information on this topic, see Amazon Machine Images (AMIs) for AWS PCS.
- You can select between on-demand (`ONDEMAND`) and Spot (`SPOT`) instances using `--purchase-option`. On-demand is the default. If you choose Spot instances, you can also use `--allocation-strategy` to define how AWS PCS chooses Spot capacity pools when it launches instances in the node group. For more information, see Allocation strategies for Spot Instances in the *Amazon Elastic Compute Cloud User Guide*.
- It is possible to provide Slurm configuration options for the nodes in the node group using `--slurm-configuration`. You can set the weight (scheduling priority) and real memory. Nodes with lower weights have higher priority, and the units are arbitrary. For more information, see Weight in the Slurm documentation. Real memory sets the size (in GB) of real memory on nodes in the node group. It is meant to be used in conjunction with the `CR_CPU_Memory` option for the cluster in AWS PCS in your Slurm configuration. For more information, see RealMemory in the Slurm documentation.

> **Important**
>
> It can take several minutes to create the compute node group.

You can query the status of your node group with the following command. You won't be able to associate the node group with a queue until its status reaches `ACTIVE`.

```
aws pcs get-compute-node-group --region region \
    --cluster-identifier my-cluster \
    --compute-node-group-identifier my-node-group
```

Updating an AWS PCS compute node group

This topic provides an overview of available options and describes what to consider when you update an AWS PCS compute node group.

Options for updating an AWS PCS compute node group

Updating an AWS PCS compute node group enables you to change the properties of instances launched by AWS PCS, as well as the rules for how those instances are launched. For example, you can replace the AMI for node group instances with another one with different software installed on it. Or, you can update security groups to change inbound or outbound network connectivity. You can also change the scaling configuration or even change the preferred purchase option to or from Spot instances.

The following node group settings cannot be altered after creation:

- Name
- Instances

Considerations when updating an AWS PCS compute node group

Compute node groups define EC2 instances that are used to process jobs, provide interactive shell access, and other tasks. They are often associated with one or more AWS PCS queues. As you update your compute node group to change its behavior (or that of its nodes), consider the following:

- Changes to compute node group properties become effective when the compute node group status changes from **Updating** to **Active**. New instances launch with the updated properties.
- Updates that don't impact the configuration of specific nodes don't affect running nodes. For example, adding a subnet and changing the allocation strategy.
- If you update the launch template for a compute node group, you must update the compute node group to use the new version.
- To add or remove a security group from nodes in a compute node group, edit its launch template and update the compute node group. New instances launch with the updated set of security groups.
- If you directly edit a security group used by a compute node group, it takes immediate effect on running and future instances.
- If you add or remove permissions from the IAM instance profile used by a compute node group, it takes immediate effect on running and future instances.
- To change the AMI used by a compute node group's instances, update the compute node group (or its launch template) to use the new AMI and wait for AWS PCS to replace the instances.

- AWS PCS replaces existing instances in the node group after a node group update operation. If there are jobs running on a node, those jobs are allowed to complete before AWS PCS replaces the node. Interactive user processes (such as on login node instances) are terminated. Node group status returns to `Active` when AWS PCS marks the instances for replacement, but the actual replacement occurs when the instances are idle.
- If you decrease the maximum number of instances allowed in a compute node group, AWS PCS removes nodes from Slurm to meet the new maximum. AWS PCS terminates running instances associated with the removed Slurm nodes. The running jobs on the removed nodes fail and return to their queues.
- AWS PCS creates a managed launch template for each compute node group. They are named `pcs-identifier-do-not-delete`. Don't select them when you create or update a compute node group, or the node group will not function correctly.
- If you update a compute node group to use **Spot** for its purchase option, you must have the **AWSServiceRoleForEC2Spot** service-linked role in your account. For more information, see Amazon EC2 Spot role for AWS PCS.

To update an AWS PCS compute node group

You can update a node group using the AWS Management Console or the AWS CLI.

AWS Management Console

To update a compute node group

1. Open the AWS PCS console at `https://console.aws.amazon.com/pcs/home#/clusters`
2. Select the cluster where you wish to update a compute node group.
3. Navigate to **Compute node groups**, go to the node group you wish to update, then select **Edit**.
4. In the **Computing configuration**, **Additional settings**, and **Slurm customization** settings sections, update any values except:
   - **Instances** – You can't change the the instances in a compute node group.
5. Choose **Update**. The **Status** field will show *Updating* while changes are being applied.

> **Important**
>
> Compute node group updates can take several minutes.

AWS CLI

To update a compute node group

1. Update your compute node group with the command that follows. Before running the command, make the following replacements:

   a. Replace *region-code* with the AWS Region that you want to create your cluster in.

   b. Replace *my-node-group* with the name or `computeNodeGroupId` for your compute node group.

   c. Replace *my-cluster* with the name or `clusterId` of your cluster.

```
aws pcs update-compute-node-group --region region-code \
    --cluster-identifier my-cluster \
    --compute-node-group-identifier my-node-group
```

2. Update any node group parameters except for `--instance-configs`. For example, to set a new AMI ID, pass `--amiId my-custom-ami-id` where *my-custom-ami-id* is replaced by your AMI of choice.

> **Important**
>
> It can take several minutes to update the compute node group.

You can query the status of your node group with the following command.

```
aws pcs get-compute-node-group --region region-code \
    --cluster-identifier my-cluster \
    --compute-node-group-identifier my-node-group
```

Deleting a compute node group in AWS PCS

This topic provides an overview of available options and describes what to consider when you delete an compute node group in AWS PCS.

Considerations when deleting a compute node group

Compute node groups define EC2 instances that are used to process jobs, provide interactive shell access, and other tasks. They are often associated with one or more AWS PCS queues. Before you delete a compute node group, consider the following:

- Any EC2 instances launched by the compute node group will be terminated. This will cancel jobs that are running on these instances, and terminate running interactive processes.
- You must disassociate the compute node group from all queues before you can delete it. For more information, see Updating an AWS PCS queue.

Delete the compute node group

You can use the AWS Management Console or AWS CLI to delete a compute node group.

AWS Management Console

To delete a compute node group

1. Open the AWS PCS console.
2. Select the cluster of the compute node group.
3. Navigate to **Compute node groups** and select the compute node group to delete.
4. Choose **Delete**.
5. The **Status** field shows `Deleting`. It can take several minutes to complete.

> **Note**
>
> You can use commands native to your scheduler to confirm that the compute node group is deleted. For example, use `sinfo` or `squeue` for Slurm.

AWS CLI

To delete a compute node group

- Use the following command to delete a compute node group, with these replacements:
  - Replace *region-code* with the AWS Region your cluster is in.
  - Replace *my-node-group* with the name or ID of your compute node group.
  - Replace *my-cluster* with the name or ID of your cluster.

```
aws pcs delete-compute-node-group --region region-code \
       --compute-node-group-identifier my-node-group \
       --cluster-identifier my-cluster
```

  It can take several minutes to delete the compute node group.

> **Note**
>
> You can use commands native to your scheduler to confirm that the compute node group is deleted. For example, use `sinfo` or `squeue` for Slurm.

Finding compute node group instances in AWS PCS

Each AWS PCS compute node group can launch EC2 instances with shared configurations. You can use EC2 tags to find instances in a compute node group in the AWS Management Console or with the AWS CLI.

AWS Management Console

To find your compute node group instances

1. Open the AWS PCS console.
2. Select the cluster.
3. Choose **Compute node groups**.
4. Find the ID for the login node group you created.
5. Navigate to the EC2 console and choose **Instances**.

6. Search for the instances with the following tag. Replace *node-group-id* with the **ID** (not the name) of your compute node group.

```
aws:pcs:compute-node-group-id=node-group-id
```

7. (Optional) You can change the value of **Instance state** in the search field to find instances that are being configured or that were recently terminated.
8. Find the instance ID and IP address for each instance in the list of tagged instances.

AWS CLI

To find your node group instances, use the commands that follow. Before running the commands, make the following replacements:

- Replace *region-code* with the AWS Region of your cluster. Example: `us-east-1`
- Replace *node-group-id* with the **ID** (not the name) of your compute node group.
- Replace `running` with other instance states such as `pending` or `terminated` to find EC2 instances in other states.

```
aws ec2 describe-instances \
     --region region-code --filters \
     "Name=tag:aws:pcs:compute-node-group-id,Values=node-group-id" \
     "Name=instance-state-name,Values=running" \
     --query 'Reservations[*].Instances[*].
{InstanceID:InstanceId,State:State.Name,PublicIP:PublicIpAddress,PrivateIP:PrivateIpAddress}
```

The command returns output similar to the following. The value of `PublicIP` is `null` if the instance is in a private subnet.

```
[
    [
        {
            "InstanceID": "i-0123456789abcdefa",
            "State": "running",
            "PublicIP": "18.189.32.188",
            "PrivateIP": "10.0.0.1"
        }
    ]
```

```
]
```

> **Note**
>
> If you expect `describe-instances` to return a large number of instances, you must use options for multiple pages. For more information, see DescribeInstances in the *Amazon Elastic Compute Cloud API Reference*.

Using Amazon EC2 launch templates with AWS PCS

In Amazon EC2, a launch template can store a set of preferences so that you don't have to specify them individually when you launch instances. AWS PCS incorporates launch templates as a flexible way to configure compute node groups. When you create a node group, you provide a launch template. AWS PCS creates a derived launch template from it that includes transformations to help ensure it works with the service.

Understanding what the options and considerations are when writing a custom launch template can help you write one for use with AWS PCS. For more information on launch templates, see Launching an Instance from a Launch an instance from a launch template in the *Amazon EC2 User Guide*.

Topics

Overview of launch templates in AWS PCS

There are over 30 parameters available you can include in an EC2 launch template, controlling many aspects of how instances are configured. Most are fully compatible with AWS PCS, but there are some exceptions.

The following parameters of EC2 Launch template will be ignored by AWS PCS as these properties have to be directly managed by the service:

- **Instance type/Specify instance type attributes** (`InstanceRequirements`) – AWS PCS does not support attribute-based instance selection.
- **Instance type** (`InstanceType`) – Specify instance types when you create a node group.
- **Advanced details/IAM instance profile** (`IamInstanceProfile`) – You provide this when you create or update the node group.

- **Advanced details/Disable API termination** (`DisableApiTermination`) – AWS PCS must control the lifecycle of node group instances it launches.
- **Advanced details/Disable API stop** (`DisableApiStop`) – AWS PCS must control the lifecycle of node group instances it launches.
- **Advanced details/Stop – Hibernate behavior** (`HibernationOptions`) – AWS PCS does not support instance hibernation.
- **Advanced details/Elastic GPU** (`ElasticGpuSpecifications`) – Amazon Elastic Graphics reached end of life on January 8, 2024.
- **Advanced details/Elastic inference** (`ElasticInferenceAccelerators`) – Amazon Elastic Inference is no longer available to new customers.
- **AAdvanced details/Specify CPU options/Threads per core** (`ThreadsPerCore`) – AWS PCS sets the number of threads per core to 1.

These parameters have special requirements that support compatibility with AWS PCS:

- **User data**(`UserData`) – This must be multi-part encoded. See Working with Amazon EC2 user data.
- **Application and OS Images**(`ImageId`) – You can include this. However, if you specify an AMI ID when you create or update the node group, it will override the value in the launch template. The AMI you provide must be compatible with AWS PCS. For more information, see "Amazon Machine Images (AMIs) for AWS PCS.
- **Network settings/Firewall (security groups)**(`SecurityGroups`) – A list of security group names can't be set in an AWS PCS launch template. You can set a list of security group IDs (`SecurityGroupIds`), unless you define network interfaces in the launch template. Then, you must specify security group IDs for each interface. For more information, see Security groups in AWS PCS.
- **Network settings/Advanced network configuration**(`NetworkInterfaces`) – If you use EC2 instances with a single network card, and don't require any specialized networking configuration, AWS PCS can configure instance networking for you. To configure multiple network cards or to enable Elastic Fabric Adapter on your instances, use `NetworkInterfaces`. Each network interface must have a list of security group IDs under `Groups`. For more information, see Multiple network interfaces in AWS PCS.
- **Advanced details/Capacity reservation**(`CapacityReservationSpecification`) – This can be set, but cannot reference a specific `CapacityReservationId` when working with AWS PCS.

You can, however, reference a capacity reservation group, where that group contains one or more capacity reservations. For more information, see Capacity Reservations in AWS PCS.

Create a basic launch template

You can create a launch template using the AWS Management Console or the AWS CLI.

AWS Management Console

To create a launch template

1. Open the Amazon EC2 console and select **Launch templates**.
2. Choose **Create launch template**.
3. Under **Launch template name and description** enter a unique, distinctive name for **Launch template name**
4. Under Key pair (login) at Key pair name, select the SSH key pair that will be used to log into EC2 instances managed by AWS PCS. This is optional, but recommended.
5. Under **Network settings**, then **Firewall (security groups)**, choose security groups to attach to the network interface. All security groups in the launch template must be from your AWS PCS cluster VPC. At minimum, choose:
   - A security group that allows communication with the AWS PCS cluster
   - A security group that allows communication between EC2 instances launched by AWS PCS
   - (Optional) A security group that allows inbound SSH access to interactive instances
   - (Optional) A security group that allows compute nodes to make outgoing connections to the Internet
   - (Optional) Security group(s) that allow access to networked resources such as shared file systems or a database server.
6. Your new launch template ID will be accessible in the Amazon EC2 console under **Launch templates**. The launch template ID will have the form `lt-0123456789abcdef01`.

Recommended next step

- Use the new launch template to create or update an AWS PCS compute node group.

AWS CLI

To create a launch template

Create your launch template with the command that follows.

- Before running the command, make the following replacements:

  a. Replace *region-code* with the AWS Region where you are working with AWS PCS

  b. Replace *my-launch-template-name* with a name for your template. It must be unique to the AWS account and AWS Region you are using.

  c. Replace *my-ssh-key-name* with name of your preferred SSH key.

  d. Replace *sg-ExampleID1* and *sg-ExampleID2* with security group IDs that allow communication between your EC2 instances and the scheduler and communication between EC2 instances. If you only have one security group that enables all this traffic, you can remove `sg-ExampleID2` and its preceding comma character. You can also add more security group IDs. All security groups you include in the launch template must be from your AWS PCS cluster VPC.

```
aws ec2 create-launch-template --region region-code \
    --launch-template-name my-template-name \
    --launch-template-data '{"KeyName":"my-ssh-key-name","SecurityGroupIds":
 ["sg-ExampleID1","sg-ExampleID2"]}'
```

The AWS CLI will output text resembling the following. The launch template ID is found in `LaunchTemplateId`.

```
{
    "LaunchTemplate": {
        "LatestVersionNumber": 1,
        "LaunchTemplateId": "lt-0123456789abcdef01",
        "LaunchTemplateName": "my-launch-template-name",
        "DefaultVersionNumber": 1,
        "CreatedBy": "arn:aws:iam::123456789012:user/Bob",
        "CreateTime": "2019-04-30T18:16:06.000Z"
    }
}
```

Recommended next step

- Use the new launch template to create or update an AWS PCS compute node group.

Working with Amazon EC2 user data

You can supply EC2 user data in your launch template that `cloud-init` runs when your instances launch. User data blocks with the content type `cloud-config` run before the instance registers with the AWS PCS API, while user data blocks with content type `text/x-shellscript` run after registration completes, but before the Slurm daemon starts. For more information about content types, see the cloud-init documentation.

our user data can perform common configuration scenarios, including but not limited to the following:

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

User data in launch templates must be in the MIME multi-part archive format. This is because your user data is merged with other AWS PCS user data that is required to configure nodes in your node group. You can combine multiple user data blocks together into a single MIME multi-part file.

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

- The content type and part boundary declaration: `Content-Type: multipart/mixed; boundary="==BOUNDARY=="`
- The MIME version declaration: `MIME-Version: 1.0`
- One or more user data blocks that contain the following components:
  - The opening boundary that signals the beginning of a user data block: `--==BOUNDARY==`. You must keep the line before this boundary blank.
  - The content type declaration for the block: `Content-Type: text/cloud-config; charset="us-ascii"` or `Content-Type: text/x-shellscript; charset="us-ascii"`. You must keep the line after the content type declaration blank.
  - The content of the user data, such as a list of shell commands or `cloud-config` directives.

- The closing boundary that signals the end of the MIME multi-part file: `--==BOUNDARY==--`. You must keep the line before the closing boundary blank.

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


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


Examples

- Example: Install software from a package repository
- Example: Run scripts from an S3 bucket
- Example: Set global environment variables
- Using network file systems with AWS PCS
- Example: Use an EFS file system as a shared home directory

Example: Install software for AWS PCS from a package repository

Provide this script as the value of `"userData"` in your launch template. For more information, see Working with Amazon EC2 user data.

This script uses **cloud-config** to install software packages on node group instances at launch. For more information, see the User data formats in the *cloud-init documentation*. This example installs `curl` and `llvm`.

Note

Your instances must be able to connect to their configured package repositories.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
- python3-devel
- rust
- golang

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

Example: Run additional scripts for AWS PCS from an S3 bucket

Provide this script as the value of `"userData"` in your launch template. For more information, see Working with Amazon EC2 user data.

The following user data script uses **cloud-config** to import a script from an S3 bucket and run it on node group instances at launch. For more information, see the User data formats in the *cloud-init documentation*.

Replace the following values with your own details:

- *`amzn-s3-demo-bucket`* – The name of an S3 bucket your account can read from.
- *`object-key`* – The S3 object key of the script to import. This includes the name of the script and its location in the folder structure of the bucket. For example, `scripts/script.sh`. For more information, see Organizing objects in the Amazon S3 console by using folders in the *Amazon Simple Storage Service User Guide*.
- *`shell`* – The Linux shell to use to run the script, such as `bash`.

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

```
--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- aws s3 cp s3://amzn-s3-demo-bucket/object-key /tmp/script.sh
- /usr/bin/shell /tmp/script.sh

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

The IAM instance profile for the node group must have access to the bucket. The following IAM policy is an example for the bucket in the user data script above.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

Example: Set global environment variables for AWS PCS

Provide this script as the value of `"userData"` in your launch template. For more information, see Working with Amazon EC2 user data.

The following example uses `/etc/profile.d` to set global variables on node group instances.

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

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
```

```
#!/bin/bash
touch /etc/profile.d/awspcs-userdata-vars.sh
echo MY_GLOBAL_VAR1=100 >> /etc/profile.d/awspcs-userdata-vars.sh
echo MY_GLOBAL_VAR2=abc >> /etc/profile.d/awspcs-userdata-vars.sh

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

Example: Use an EFS file system as a shared home directory for AWS PCS

Provide this script as the value of `"userData"` in your launch template. For more information, see Working with Amazon EC2 user data.

This example extends the example EFS mount in Using network file systems with AWS PCS to implement a shared home directory. The contents of /home are backed up before the EFS file system is mounted. The contents are then quickly copied into place on the shared storage after the mount completes.

Replace the following values in this script with your own details:

- *`/mount-point-directory`* – The path on an instance where you want to mount the EFS file system.
- *`filesystem-id`* – The file system ID for the EFS file system.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
  - amazon-efs-utils

runcmd:
  - mkdir -p /tmp/home
  - rsync -a /home/ /tmp/home
  - echo "filesystem-id:/ /mount-point-directory efs tls,_netdev" >> /etc/fstab
  - mount -a -t efs defaults
  - rsync -a --ignore-existing /tmp/home/ /home
  - rm -rf /tmp/home/
```

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

Example: Enabling passwordless SSH

You can build on the shared home directory example to implement SSH connections between cluster instances using SSH keys. For each user using the shared home file system, run a script that resembles the following:

```
#!/bin/bash

mkdir -p $HOME/.ssh && chmod 700 $HOME/.ssh
touch $HOME/.ssh/authorized_keys
chmod 600 $HOME/.ssh/authorized_keys

if [ ! -f "$HOME/.ssh/id_rsa" ]; then
    ssh-keygen -t rsa -b 4096 -f $HOME/.ssh/id_rsa -N ""
    cat ~/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
fi
```

> **Note**
>
> The instances must use a security group that allows SSH connections between cluster nodes.

Capacity Reservations in AWS PCS

You can reserve Amazon EC2 capacity in a specific Availability Zone and for a specific duration using On-Demand Capacity Reservations or EC2 Capacity Blocks to make sure that you have the necessary compute capacity available when you need it.

> **Note**
>
> AWS PCS supports On-Demand Capacity Reservations (ODCR) but doesn't currently support Capacity Blocks for ML.

Using ODCRs with AWS PCS

You can choose how AWS PCS consumes your reserved instances. If you create an **open** ODCR, any matching instances launched by AWS PCS or other processes in your account count against the reservation. With a **targeted** ODCR, only instances launched with the specific reservation ID count against the reservation. For time-sensitive workloads, targeted ODCRs are more common.

You can configure an AWS PCS compute node group to use a targeted ODCR by adding it to a launch template. Here are the steps to do so:

1. Create a targeted on-demand Capacity Reservation (ODCR).
2. Add the ODCR to a Capacity Reservation group.
3. Associate the Capacity Reservation group with a launch template.
4. Create or update an AWS PCS compute node group to use the launch template.

Example: Reserve and use hpc6a.48xlarge instances with a targeted ODCR

This example command creates a targeted ODCR for 32 hpc6a.48xlarge instances. To launch the reserved instances in a placement group, add `--placement-group-arn` to the command. You can define a stop date with `--end-date` and `--end-date-type`, otherwise the reservation will continue until it is manually terminated.

```
aws ec2 create-capacity-reservation \
    --instance-type hpc6a.48xlarge \
    --instance-platform Linux/UNIX \
    --availability-zone us-east-2a \
    --instance-count 32 \
    --instance-match-criteria targeted
```

The result from this command will be an ARN for the new ODCR. To use the ODCR with AWS PCS, it must be added to a Capacity Reservation group. This is because AWS PCS does not support individual ODCRs. For more information, see Capacity Reservation groups in the *Amazon Elastic Compute Cloud User Guide*.

Here is how to add the ODCR to a Capacity Reservation group named `EXAMPLE-CR-GROUP`.

```
aws resource-groups group-resources --group EXAMPLE-CR-GROUP \
```

```
    --resource-arns arn:aws:ec2:sa-east-1:123456789012:capacity-reservation/
cr-1234567890abcdef1
```

With the ODCR created and added to a Capacity Reservation group, it can now be connected to an AWS PCS compute node group by adding it to a launch template. Here is an example launch template that references the Capacity Reservation group.

```
{
  "CapacityReservationSpecification": {
    "CapacityReservationResourceGroupArn": "arn:aws:resource-groups:us-
east-2:123456789012:group/EXAMPLE-CR-GROUP"
  }
}
```

Finally, create or update an AWS PCS compute node group to use hpc6a.48xlarge instances and use the launch template that references the ODCR in its Capacity Reservation group. For a static node group, set minimum and maximum instances to the size of the reservation (32). For a dynamic node group, set the minimum instances to 0 and the maximum up to the reservation size.

This example is a simple implementation of a single ODCR that provisioned for one compute node group. But, AWS PCS supports many other designs. For example, you can subdivide a large ODCR or Capacity Reservation group among multiple compute node groups. Or, you can use ODCRs that another AWS account has created and shared with yours. The key constraint is that ODCRs always must be contained in a Capacity Reservation group.

For more information, see On-Demand Capacity Reservations and Capacity Blocks for ML in the *Amazon Elastic Compute Cloud User Guide*.

Useful launch template parameters

This section describes some launch template parameters that may be broadly useful with AWS PCS.

Turn on detailed CloudWatch monitoring

You can enable collection of CloudWatch metrics at a shorter interval using a launch template parameter.

AWS Management Console

On the console pages for creating or editing launch templates, this option is found under the **Advanced details** section. Set **Detailed CloudWatch monitoring** to *Enable.*

YAML

```
Monitoring:
    Enabled: True
```

JSON

```
{"Monitoring": {"Enabled": "True"}}
```

For more information, see Enable or turn off detailed monitoring for your instances in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

Instance Metadata Service Version 2 (IMDS v2)

Using IMDS v2 with EC2 instances offers significant security enhancements and helps mitigate potential risks associated with accessing instance metadata in AWS environments.

AWS Management Console

On the console pages for creating or editing launch templates, this option is found under the **Advanced details** section. Set **Metadata accessible** to *Enabled*, **Metadata version** to *V2 only (token required)*, and **Metadata response hop limit** to *4*.

YAML

```
MetadataOptions:
  HttpEndpoint: enabled
  HttpTokens: required
  HttpPutResponseHopLimit: 4
```

JSON

```
{
    "MetadataOptions": {
        "HttpEndpoint": "enabled",
        "HttpPutResponseHopLimit": 4,
```

```
        "HttpTokens": "required"
    }
}
```

.

AWS PCS queues

An AWS PCS queue is a lightweight abstraction over the scheduler's native implementation of a work queue. In the case of Slurm, an AWS PCS queue is equivalent to a Slurm partition.

Users submit jobs to a queue where they reside until they can be scheduled to run on nodes provided by one or more compute node groups. An AWS PCS cluster can have multiple job queues. For example, you can create a queue that uses Amazon EC2 On-demand Instances for high priority jobs and another queue that uses Amazon EC2 Spot Instances for low-priority jobs.

Topics

Creating a queue in AWS PCS

This topic provides an overview of available options and describes what to consider when you create a queue in AWS PCS.

Prerequisites

- An AWS PCS cluster - queues can only be created in association with a specific AWS PCS cluster.
- One or more AWS PCS compute node groups - a queue must be associated with at least one AWS PCS compute node group.

To create a queue in AWS PCS

You can create a queue using the AWS Management Console or the AWS CLI.

AWS Management Console

To create a queue using the console

1. Open the AWS PCS console.
2. Select the cluster for the queue. Navigate to **Queues** and choose **Create queue**.

3. In the **Queue configuration** section, provide the following values:

   a. **Queue name** – A name for your queue. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 25 characters. The name must be unique within the cluster.

   b. **Compute node groups** – Select 1 or more compute node groups to service this queue. A compute node group can be associated with more than 1 queue.

4. (Optional) Under **Tags**, add any tags to your AWS PCS queue

5. Choose **Create queue**. The **Status** field will show **Creating** while AWS PCS creates the queue. Queue creation can take several minutes.

Recommended next step

- Submit a job to your new queue.

AWS CLI

To create a queue using AWS CLI

Use the following command to create your queue. Make the following replacements:

1. Replace *region-code* with the AWS Region of the cluster. For example, `us-east-1`.
2. Replace *my-queue* with the name for your queue. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 25 characters. The name must be unique within the cluster.
3. Replace *my-cluster* with the name or ID of your cluster.
4. Replace *compute-node-group-id* with the ID of the compute node group to service the queue. For example, `pcs_abcdef12345`.

   > **Note**
   >
   > When you create a queue, you must provide the ID of the compute node group and not its name.

```
aws pcs create-queue --region region-code \
    --queue-name my-queue \
```

```
    --cluster-identifier my-cluster \
    --compute-node-group-configurations \
    computeNodeGroupId=compute-node-group-id
```

It can take several minutes to create the queue. You can query the status of your queue with the following command. You won't be able to submit jobs to the queue until its status reaches `ACTIVE`.

```
aws pcs get-queue --region region-code \
    --cluster-identifier my-cluster \
    --queue-identifier my-queue
```

Recommended next step

- Submit a job to your new queue

Updating an AWS PCS queue

This topic provides an overview of available options and describes what to consider when you update an AWS PCS queue.

Considerations when updating an AWS PCS queue

Queue updates will not impact running jobs but the cluster may not be able to accept new jobs while the queue is being updated.

To update an AWS PCS queue

You can use the AWS Management Console or AWS CLI to update a queue.

AWS Management Console

To update a queue

1. Open the AWS PCS console at `https://console.aws.amazon.com/pcs/home#/clusters`
2. Select the cluster where you wish to update a queue.
3. Navigate to **Queues**, go to the queue wish to update, then select **Edit**.
4. In the queue configuration section, update any of the following values:

- **Node groups** – Add or remove compute node groups from association with the queue.
- **Tags** – Add or remove tags for the queue.

5. Choose **Update**. The **Status** field will show *Updating* while changes are being applied.

> **Important**
>
> Queue updates can take several minutes.

AWS CLI

To update a queue

1. Update your queue with the command that follows. Before running the command, make the following replacements:

   a. Replace *region-code* with the AWS Region that you want to create your cluster in.

   b. Replace *my-queue* with the name or `computeNodeGroupId` for your queue.

   c. Replace *my-cluster* with the name or `clusterId` of your cluster.

   d. To change compute node group associations, provide an updated list for `--compute-node-group-configurations`.

   - For example, to add a second compute node group `computeNodeGroupExampleID2`:

```
--compute-node-group-configurations
 computeNodeGroupId=computeNodeGroupExampleID1,computeNodeGroupId=computeNodeGro
```

```
aws pcs update-queue --region region-code \
    --queue-identifier my-queue \
    --cluster-identifier my-cluster \
    --compute-node-group-configurations \
    computeNodeGroupId=computeNodeGroupExampleID1
```

2. It can take several minutes to update the queue. You can query the status of your queue with the following command. You won't be able to submit jobs to the queue until its status reaches `ACTIVE`.

```
aws pcs get-queue --region region-code \
    --cluster-identifier my-cluster \
    --queue-identifier my-queue
```

Recommended next steps

- Submit a job to your updated queue.

Deleting a queue in AWS PCS

This topic provides an overview of how to delete an queue in AWS PCS.

Considerations when deleting a queue

- If there are jobs running in the queue, they will be terminated by the scheduler when the queue is deleted. Pending jobs in the queue will be canceled. Consider waiting for jobs in the queue to finish or manually stop/cancel them using the scheduler's native commands (such as `scancel` for Slurm).

Delete the queue

You can use the AWS Management Console or AWS CLI to delete a queue.

AWS Management Console

To delete a queue

1. Open the AWS PCS console.
2. Select the cluster of the queue.
3. Navigate to **Queues** and select the queue to delete.
4. Choose **Delete**.
5. The **Status** field shows `Deleting`. It can take several minutes to complete.

> **Note**
>
> You can use commands native to your scheduler to confirm that the queue is deleted. For example, use `sinfo` or `squeue` for Slurm.

AWS CLI

To delete a queue

- Use the following command to delete a queue, with these replacements:
  - Replace *`region-code`* with the AWS Region your cluster is in.
  - Replace *`my-queue`* with the name or ID of your queue.
  - Replace *`my-cluster`* with the name or ID of your cluster.

```
aws pcs delete-queue --region region-code \
       --queue-identifier my-queue \
       --cluster-identifier my-cluster
```

  It can take several minutes to delete the queue.

> **Note**
>
> You can use commands native to your scheduler to confirm that the queue is deleted. For example, use `sinfo` or `squeue` for Slurm.

AWS PCS login nodes

An AWS PCS cluster usually needs at least 1 login node to support interactive access and job management. A way to accomplish this is with a static AWS PCS compute node group configured for login node capability. You can also configure a standalone EC2 instance to act as a login node.

Topics

Using an AWS PCS compute node group to provide login nodes

This topic provides an overview of suggested configuration options and describes what to consider when you use an AWS PCS compute node group to provide persistent, interactive access to your cluster.

Creating an AWS PCS compute node group for login nodes

Operationally, this is not much different from creating a regular compute node group. However, there are some key configuration choices make:

- Set a static scaling configuration of at least one EC2 instance in the compute node group.
- Choose on-demand purchase option to avoid having your instance(s) reclaimed.
- Choose an informative name for the compute node group, such as login.
- If you want the login node instance(s) to be accessible outside your VPC, consider using a public subnet.
- If you intend to allow SSH access, the launch template will need have a security group that exposes the SSH port to your choice of IP addresses.
- The IAM instance profile should have only the AWS permissions you want your end users to have. See IAM instance profiles for AWS Parallel Computing Service for details.
- Consider allowing AWS Systems Manager Session Manager to manage your login instances.
- Consider restricting access to the instance AWS credentials to only administrative users
- Select less expensive instance types than for regular compute node groups, since the login node(s) will be running continuously.

- Use the same (or a derivative) AMI as for your other compute node groups to help ensure all instances have the same software installed. For more information about customzing AMIs, see Amazon Machine Images (AMIs) for AWS PCS
- Configure the same network file system (Amazon EFS, Amazon FSx for Lustre, etc.) mounts on your login nodes as on your compute instances. For more information, see Using network file systems with AWS PCS.

Access your login nodes

Once your new compute node group reaches ACTIVE status, you can find the EC2 instance(s) it has created and log into them. For more information, see Finding compute node group instances in AWS PCS.

Updating an AWS PCS compute node group for login nodes

You can update a login node group using UpdateComputeNodeGroup. As part of the node group update process, running instances will be replaced. Note that this will interrupt any active user sessions or processes on the instance. Running or queued Slurm jobs will be unaffected. For more information, see Updating an AWS PCS compute node group.

You can also edit the launch template used by your compute node group. You must use UpdateComputeNodeGroup to apply the updated launch template to the compute node group. New EC2 instances launched in the compute node group use the updated launch template. For more information, see Using Amazon EC2 launch templates with AWS PCS.

Deleting an AWS PCS compute node group for login nodes

You can update a login node group using the **delete compute node group** mechanism in AWS PCS. Running instances will be terminated as part of node group deletion. Please note that this will interrupt any active user sessions or processes on the instance. Running or queued Slurm jobs will be unaffected. For more information, see Deleting a compute node group in AWS PCS.

Using standalone instances as AWS PCS login nodes

You can set up independent EC2 instances to interact with an AWS PCS cluster's Slurm scheduler. This is useful for creating login nodes, workstations, or dedicated workflow management hosts that work with AWS PCS clusters but operate outside of AWS PCS management. To do this, each standalone instance must:

1. Have a compatible Slurm software version installed.
2. Be able to connect to the AWS PCS cluster's Slurmctld endpoint.
3. Have the Slurm Auth and Cred Kiosk Daemon (`sackd`) properly configured with the AWS PCS cluster's endpoint and secret. For more information, see [sackd](#) in the Slurm documentation.

This tutorial helps you configure an independent instance that connects to an AWS PCS cluster.

Contents

Step 1 – Retrieve the address and secret for the target AWS PCS cluster

Retrieve details about the target AWS PCS cluster using the AWS CLI with the command that follows. Before running the command, make the following replacements:

- Replace *`region-code`* with the AWS Region where the target cluster is running.
- Replace *`cluster-ident`* with the name or identifier for the target cluster

```
aws pcs get-cluster --region region-code --cluster-identifier cluster-ident
```

The command will return output similar to this example.

```
{
    "cluster": {
        "name": "get-started",
        "id": "pcs_123456abcd",
        "arn": "arn:aws:pcs:us-east-1:111122223333:cluster/pcs_123456abcd",
        "status": "ACTIVE",
        "createdAt": "2024-12-17T21:03:52+00:00",
```

```
        "modifiedAt": "2024-12-17T21:03:52+00:00",
        "scheduler": {
            "type": "SLURM",
            "version": "24.05"
        },
        "size": "SMALL",
        "slurmConfiguration": {
            "authKey": {
                "secretArn": "arn:aws:secretsmanager:us-east-1:111122223333:secret:pcs!
slurm-secret-pcs_123456abcd-a12ABC",
                "secretVersion": "ef232370-d3e7-434c-9a87-ec35c1987f75"
            }
        },
        "networking": {
            "subnetIds": [
                "subnet-0123456789abcdef0"
            ],
            "securityGroupIds": [
                "sg-0123456789abcdef0"
            ]
        },
        "endpoints": [
            {
                "type": "SLURMCTLD",
                "privateIpAddress": "10.3.149.220",
                "port": "6817"
            }
        ]
    }
}
```

In this sample, the cluster Slurm controller endpoint has an IP address of `10.3.149.220` and it is running on port `6817`. The `secretArn` will be used in later steps to retrieve the cluster secret. The IP address and port will be used in later steps to configure the `sackd` service.

Step 2 – Launch an EC2 instance

To launch an EC2 instance

1. Open the Amazon EC2 console.
2. In the navigation pane, choose **Instances**, and then choose **Launch Instances** to open the new launch instance wizard.

3. (Optional) In the **Name and tags** section, provide a name for the instance, such as `PCS-LoginNode`. The name is assigned to the instance as a resource tag (`Name=PCS-LoginNode`).
4. In the **Application and OS Images** section, select an AMI for one of the operating systems supported by AWS PCS. For more information, see Supported operating systems.
5. In the **Instance type** section, select a supported instance type. For more information, see Supported instance types.
6. In the **Key pair** section, select the SSH key pair to use for the instance.
7. In the **Network settings** section:
   - Choose **Edit**.
     i. Select the VPC of your AWS PCS cluster.
     ii. For **Firewall (security groups)**, choose **Select existing security group**.
        A. Select a security group that permits traffic between the instance and the target AWS PCS cluster's Slurm controller. For more information, see Security group requirements and considerations.
        B. (Optional) Select a security group that allows inbound SSH access to your instance.
8. In the **Storage** section, configure storage volumes as needed. Make sure to configure sufficient space to install applications and libraries to enable your use case.
9. Under **Advanced**, choose an IAM role that allows access to the cluster secret. For more information, see Get the Slurm cluster secret.
10. In the **Summary** pane, choose **Launch instance**.

Step 3 – Install Slurm on the instance

When the instance has launched and becomes active, connect to it using your preferred mechanism. Use the Slurm installer provided by AWS to install Slurm onto the instance. For more information, see Slurm installer.

Download the Slurm installer, uncompress it, and use the `installer.sh` script to install Slurm. For more information, see Step 3 – Install Slurm.

Step 4 – Retrieve and store the cluster secret

These instructions require the AWS CLI. For more information, see Install or update to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

Store the cluster secret with the following commands.

- Create the configuration directory for Slurm.

```
sudo mkdir -p /etc/slurm
```

- Retrieve, decode, and store the cluster secret. Before running this command, replace *region-code* with the Region where the target cluster is running, and replace *secret-arn* with the value for `secretArn` retrieved in Step 1.

```
aws secretsmanager get-secret-value \
 --region region-code \
 --secret-id 'secret-arn' \
 --version-stage AWSCURRENT \
 --query 'SecretString' \
 --output text | base64 -d | sudo tee /etc/slurm/slurm.key
```

> **Warning**
>
> In a multiuser environment, any user with access to the instance might be able to fetch the cluster secret if they can access the instance metadata service (IMDS). This, in turn, could allow them to impersonate other users. Consider restricting access to IMDS to root or administrative users only. Alternatively, consider using a different mechanism that doesn't rely on the instance profile to fetch and configure the secret.

- Set ownership and permissions on the Slurm key file.

```
sudo chmod 0600 /etc/slurm/slurm.key
sudo chown slurm:slurm /etc/slurm/slurm.key
```

> **Note**
>
> The Slurm key must be owned by the user and group that the `sackd` service runs as.

Step 5 – Configure the connection to the AWS PCS cluster

To establish a connection to the AWS PCS cluster, launch `sackd` as a system service by following these steps.

1. Set up the environment file for the `sackd` service with the command that follows. Before running the command, replace *`ip-address`* and *`port`* with the values retrieved from endpoints in Step 1.

```
sudo echo "SACKD_OPTIONS='--conf-server=ip-address:port'" > /etc/sysconfig/sackd
```

2. Create a `systemd` service file for managing the `sackd` process.

```
sudo cat << EOF > /etc/systemd/system/sackd.service
[Unit]
Description=Slurm auth and cred kiosk daemon
After=network-online.target remote-fs.target
Wants=network-online.target
ConditionPathExists=/etc/sysconfig/sackd

[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/sackd
User=slurm
Group=slurm
RuntimeDirectory=slurm
RuntimeDirectoryMode=0755
ExecStart=/opt/aws/pcs/scheduler/slurm-24.05/sbin/sackd --systemd \$SACKD_OPTIONS
ExecReload=/bin/kill -HUP \$MAINPID
KillMode=process
LimitNOFILE=131072
LimitMEMLOCK=infinity
LimitSTACK=infinity

[Install]
WantedBy=multi-user.target
EOF
```

3. Set ownership of the `sackd` service file.

```
sudo chown root:root /etc/systemd/system/sackd.service && \
    sudo chmod 0644 /etc/systemd/system/sackd.service
```

4. Enable the `sackd` service.

```
sudo systemctl daemon-reload && sudo systemctl enable sackd
```

5. Start the `sackd` service.

```
sudo systemctl start sackd
```

Step 6 – (Optional) Test the connection

Confirm that the `sackd` service is running. Sample output follows. If there are errors, they will commonly show up here.

```
[root@ip-10-3-27-112 ~]# systemctl status sackd
[x] sackd.service - Slurm auth and cred kiosk daemon
   Loaded: loaded (/etc/systemd/system/sackd.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2024-12-17 16:34:55 UTC; 8s ago
 Main PID: 9985 (sackd)
   CGroup: /system.slice/sackd.service
           ##9985 /opt/aws/pcs/scheduler/slurm-24.05/sbin/sackd --systemd --conf-
server=10.3.149.220:6817

Dec 17 16:34:55 ip-10-3-27-112.ec2.internal systemd[1]: Starting Slurm auth and cred
 kiosk daemon...
Dec 17 16:34:55 ip-10-3-27-112.ec2.internal systemd[1]: Started Slurm auth and cred
 kiosk daemon.
Dec 17 16:34:55 ip-10-3-27-112.ec2.internal sackd[9985]: sackd: running
```

Confirm connections to the cluster are working using Slurm client commands such as `sinfo` and `squeue`. Here is example output from `sinfo`.

```
[root@ip-10-3-27-112 ~]# /opt/aws/pcs/scheduler/slurm-24.05/bin/sinfo
PARTITION AVAIL TIMELIMIT NODES STATE NODELIST
all up infinite 4 idle~ compute-[1-4]
```

You should also be able to submit jobs. For example, a command similar to this example would launch an interactive job on 1 node in the cluster.

```
/opt/aws/pcs/scheduler/slurm-24.05/bin/srun --nodes=1 -p all --pty bash -i
```

AWS PCS Networking

Your AWS PCS cluster is created in an Amazon VPC. This chapter includes the following topics about networking for your cluster's scheduler and nodes.

Except for choosing a subnet to launch instances in, you must use EC2 launch templates to configure networking for AWS PCS compute node groups. For more information about launch templates, see Using Amazon EC2 launch templates with AWS PCS.

Topics

AWS PCS VPC and subnet requirements and considerations

When you create an AWS PCS cluster, you specify a VPC a subnet in that VPC. This topic provides an overview of AWS PCS specific requirements and considerations for the VPC and subnet(s) that you use with your cluster. If you don't have a VPC to use with AWS PCS, you can create one using an AWS-provided AWS CloudFormation template. For more information about VPCs, see Virtual private clouds (VPC) in the *Amazon VPC User Guide*.

VPC requirements and considerations

When you create a cluster, the VPC that you specify must meet the following requirements and considerations:

- The VPC must have a sufficient number of IP addresses available for the cluster, any nodes, and other cluster resources that you want to create. For more information, see IP addressing for your VPCs and subnets in the *Amazon VPC User Guide*.
- The VPC must have a DNS hostname and DNS resolution support. Otherwise, nodes can't register the customer cluster. For more information, see DNS attributes for your VPC in the *Amazon VPC User Guide*.

- The VPC might require VPC endpoints using AWS PrivateLink to be able to contact the AWS PCS API. For more information, see Connect your VPC to services using AWS PrivateLink in the *Amazon VPC User Guide*.

> **Important**
>
> AWS PCS doesn't support a VPC with dedicated instance tenancy. The VPC you use for AWS PCS must use `default` instance tenancy. You can change the instance tenancy for an existing VPC. For more information, see Change the instance tenancy of a VPC in the *Amazon Elastic Compute Cloud User Guide*.

Subnet requirements and considerations

When you create a Slurm cluster, AWS PCS creates an Elastic Network Interface(ENI) in the subnet you specified. This network interface enables communication between the scheduler controller and the customer VPC. The network interface also enables Slurm to communicate with the components deployed in the customer account. You can only specify the subnet for a cluster at creation time.

Subnet requirements for clusters

The subnet that you specify when you create a cluster must meet the following requirements:

- The subnet must have at least 1 IP address for use by AWS PCS.
- The subnet can't reside in AWS Outposts, AWS Wavelength, or an AWS Local Zone.
- The subnet can be a public or private. We recommend that you specify a private subnet, if possible. A public subnet is a subnet with a route table that includes a route to an internet gateway; a private subnet is a subnet with a route table that doesn't include a route to an internet gateway.

Subnet requirements for nodes

You can deploy nodes and other cluster resources to the subnet you specify when you create your AWS PCS cluster, and to other subnets in the same VPC.

Any subnet that you deploy nodes and cluster resources to must meet the following requirements:

- You must ensure that the subnet has enough available IP addresses to deploy all the nodes and cluster resources.
- If you plan to deploy nodes to a public subnet, that subnet must auto-assign IPv4 public addresses.
- If the subnet where you deploy nodes to is a private subnet and its route table doesn't include a route to a network address translation (NAT) device (IPv4), add VPC endpoints using AWS PrivateLink to the customer VPC. VPC endpoints are needed for all the AWS services that the nodes contact. The only required endpoint is for AWS PCS to allow the node to call the `RegisterComputeNodeGroupInstance` API action. For more information, see RegisterComputeNodeGroupInstance in the *AWS PCS API Reference*.
- Public or private subnet status doesn't impact AWS PCS; the required endpoints must be reachable.

Creating a VPC for your AWS PCS cluster

You can create an Amazon Virtual Private Cloud (Amazon VPC) for your clusters within AWS Parallel Computing Service (AWS PCS).

Use Amazon VPC to launch VPC resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you might operate in your own data center. However, it comes with the benefits of using the scalable infrastructure of Amazon Web Services. We recommend that you have a thorough understanding of the Amazon VPC service before deploying production VPC clusters. For more information, see What is Amazon VPC? in the author visual mode.*Amazon VPC User Guide*.

An PCS cluster, nodes, and supporting resources (such as file systems and directory services) are deployed within your Amazon VPC. If you want to use an existing Amazon VPC with PCS, it must meet the requirements described in AWS PCS VPC and subnet requirements and considerations . This topic describes how to create a VPC that meets PCS requirements using an AWS–provided AWS CloudFormation template. Once you've deployed a template, you can view the resources created by the template to know exactly what resources it created, and the configuration of those resources.

Prerequisites

To create an Amazon VPC for PCS, you must have the necessary IAM permissions to create Amazon VPC resources. These resources are VPCs, subnets, security groups, route tables and routes, and internet and NAT gateways. For more information, see [Create a VPC with a public subnet] in the *Amazon VPC User Guide*. To review the full list for Amazon EC2, see [Actions, resources, and condition keys for Amazon EC2] in the *Service Authorization Reference*.

Create an Amazon VPC

Create a VPC by copy and pasting the appropriate URL for the AWS Region where you will use PCS. You may also download the AWS CloudFormation template and upload it yourself to the AWS CloudFormation console.

- **US East (N. Virginia) (us-east-1)**

```
https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/
create/review?stackName=hpc-networking&templateURL=https://aws-hpc-recipes.s3.us-
east-1.amazonaws.com/main/recipes/net/hpc_large_scale/assets/main.yaml
```

- **US East (Ohio) (us-east-2)**

```
https://console.aws.amazon.com/cloudformation/home?region=us-east-2#/stacks/
create/review?stackName=hpc-networking&templateURL=https://aws-hpc-recipes.s3.us-
east-1.amazonaws.com/main/recipes/net/hpc_large_scale/assets/main.yaml
```

- **US West (Oregon) (us-west-2)**

```
https://console.aws.amazon.com/cloudformation/home?region=us-west-2#/stacks/
create/review?stackName=hpc-networking&templateURL=https://aws-hpc-recipes.s3.us-
east-1.amazonaws.com/main/recipes/net/hpc_large_scale/assets/main.yaml
```

- **Template only**

```
https://aws-hpc-recipes.s3.us-east-1.amazonaws.com/main/recipes/net/hpc_large_scale/
assets/main.yaml
```

To create an Amazon VPC for PCS

1. Open the template in the AWS CloudFormation console.

> **Note**
>
> These are pre-populated in the template so that you can simply leave them as the default values.

2. Under **Provide a stack name**, then **Stack name**, enter `hpc-networking`.
3. Under **parameters**, enter the following details:

    a. Under **VPC**, then **CidrBlock**, enter `10.3.0.0/16`

    b. Under **Subnets A**:

        i. Then **CidrPublicSubnetA**, enter `10.3.0.0/20`

        ii. Then **CidrPrivateSubnetA**, enter `10.3.128.0/20`

    c. Under **Subnets B**:

        i. Then **CidrPublicSubnetB**, enter `10.3.16.0/20`

        ii. Then **CidrPrivateSubnetA**, enter `10.3.144.0/20`

    d. Under **Subnets C**:

        i. For **ProvisionSubnetsC**, select `True`.

        > **Note**
        >
        > If you are creating a VPC in a Region that has less than three Availability Zones, this option will be ignored if set to `True`.

        ii. Then **CidrPublicSubnetB**, enter `10.3.32.0/20`

        iii. Then **CidrPrivateSubnetA**, enter `10.3.160.0/20`

4. Under **Capabilities**, check the box for **I acknowledge that AWS CloudFormation might create IAM resources**.

Monitor the status of the AWS CloudFormation stack. When it reaches `CREATE_COMPLETE`, the VPC resource are ready for you to use.

> **Note**
>
> To see all the resources the AWS CloudFormation template created, open the AWS CloudFormation console. Choose the `hpc-networking` stack and then choose the **Resources** tab.

Security groups in AWS PCS

Security groups in Amazon EC2 act as virtual firewalls to control inbound and outbound traffic to instances. Use a launch template for an AWS PCS compute node group to add or remove security groups to its instances. If your launch template doesn't contain any network interfaces, use `SecurityGroupIds` to provide a list of security groups. If your launch template defines network interfaces, you must use the `Groups` parameter to assign security groups to each network interface. For more information about launch templates, see Using Amazon EC2 launch templates with AWS PCS.

> **Note**
>
> Changes to the security group configuration in the launch template only affects new instances launched after the compute node group is updated.

Security group requirements and considerations

AWS PCS creates a cross-account Elastic Network Interface (ENI) in the subnet you specify when creating a cluster. This provides the HPC scheduler, which is running in an account managed by AWS, a path to communicate with EC2 instances launched by AWS PCS. You must provide a security group for that ENI that allows 2-way communication between the scheduler ENI and your cluster EC2 instances.

A straightforward way to accomplish this is to create a permissive self-referencing security group that permits TCP/IP traffic on all ports between all members of the group. You can attach this to both the cluster and to node group EC2 instances.

Example permissive security group configuration

Rule type	Protocols	Ports	Source	Destination
Inbound	All	All	Self	
Outbound	All	All		0.0.0.0/0
Outbound	All	All		Self

These rules allow all traffic to flow freely between the Slurm controller and nodes, allows all outbound traffic to any destination, and enables EFA traffic.

Example restrictive security group configuration

You can also limit the open ports between the cluster and its compute nodes. For the Slurm scheduler, the security group attached to your cluster must allow the following ports:

- 6817 – enable inbound connections to `slurmctld` from EC2 instances
- 6818 – enable outbound connections from `slurmctld` to `slurmd` running on EC2 instances

The security group attached to your compute nodes must allow the following ports:

- 6817 – enable outbound connections to `slurmctld` from EC2 instances.
- 6818 – enable inbound and outbound connections to `slurmd` from `slurmctld` and from `slurmd` on node group instances
- 60001–63000 – inbound and outbound connections between node group instances to support `srun`
- EFA traffic between node group instances. For more information, see Prepare an EFA-enabled security group in the *User Guide for Linux Instances*
- Any other inter-node traffic required by your workload

Multiple network interfaces in AWS PCS

Some EC2 instances have multiple network cards. This allows them to provide higher network performance, including bandwidth capabilities above 100 Gbps and improved packet handling. For more information about instances with multiple network cards, see [Elastic network interfaces] in the *Amazon Elastic Compute Cloud User Guide*.

Configure additional network cards for instances in an AWS PCS compute node group by adding network interfaces to its EC2 launch template. Below is an example launch template that enables two network cards, such as can be found on an `hpc7a.96xlarge` instance. Note the following details:

- The subnet for each network interface must be the same as you choose when configuring the AWS PCS compute node group that will use the launch template.
- The primary network device, where routine network communication such as SSH and HTTPS traffic will occur, is established by setting a `DeviceIndex` of `0`. Other network interfaces have a `DeviceIndex` of `1`. There can only be one primary network interface—all other interfaces are secondary.
- All network interfaces must have a unique `NetworkCardIndex`. A recommended practice is to number them sequentially as they are defined in the launch template.
- Security groups for each network interface are set using `Groups`. In this example, an inbound SSH security group (`sg-SshSecurityGroupId`) is added to the primary network interface, as well as the security group enabling within-cluster communications (`sg-ClusterSecurityGroupId`). Finally, a security group allowing outbound connections to the internet (`sg-InternetOutboundSecurityGroupId`) is added to both primary and secondary interfaces.

```
{
    "NetworkInterfaces": [
        {
            "DeviceIndex": 0,
            "NetworkCardIndex": 0,
            "SubnetId": "subnet-SubnetId",
            "Groups": [
               "sg-SshSecurityGroupId",
               "sg-ClusterSecurityGroupId",
               "sg-InternetOutboundSecurityGroupId"
            ]
        },
        {
            "DeviceIndex": 1,
            "NetworkCardIndex": 1,
```

```
            "SubnetId": "subnet-SubnetId",
            "Groups": ["sg-InternetOutboundSecurityGroupId"]
        }
    ]
}
```

Placement groups for EC2 instances in AWS PCS

You can use a **placement group** to influence the placement of EC2 instances to suit the needs of the workload that runs on them.

Placement group types

- **Cluster** – Packs instances close together in an Availability Zone to optimize for low-latency communication.
- **Partition** – Spreads instances across logical partitions to help maximize resilience.
- **Spread** – Strictly enforces that a small number of instances launch on distinct hardware, which can also help with resiliency.

For more information, see Placement groups for your Amazon EC2 instances in the *Amazon Elastic Compute Cloud User Guide*.

We recommended you include a **cluster** placement group when you configure an AWS PCS compute node group to use Elastic Fabric Adapter (EFA).

To create a cluster placement group that works with EFA

1. Create a placement group with the type **cluster** for the compute node group.
   - Use the following AWS CLI command:

     ```
     aws ec2 create-placement-group --strategy cluster --group-name PLACEMENT-GROUP-NAME
     ```

   - You can also use a CloudFormation template to create a placement group. For more information, see Working with CloudFormation templates in the *AWS CloudFormation User Guide*. Download the template from the following URL and upload it into the CloudFormation console.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/pcs/enable_efa/assets/efa-placement-group.yaml
```

2. Include the placement group in the EC2 launch template for the AWS PCS compute node group.

Using Elastic Fabric Adapter (EFA) with AWS PCS

Elastic Fabric Adapter (EFA) is a high performance advanced networking interconnect from AWS that you can attach to your EC2 instance to accelerate High Performance Computing (HPC) and machine learning applications. Enabling your applications running on an AWS PCS cluster with EFA involves configuring the AWS PCS compute node group instances to use EFA as follows.

> **Note**
>
> **Install EFA on an AWS PCS-compatible AMI** – The AMI used in the AWS PCS compute node group must have the EFA driver installed and loaded. For information on how to build a custom AMI with EFA software installed, see Custom Amazon Machine Images (AMIs) for AWS PCS.

Contents

Identify EFA-enabled EC2 instances

To use EFA, all instance types that are allowed for an AWS PCS compute group must support EFA, and must have the same number of vCPUs (and GPUs if appropriate). For a list of EFA-enabled instances, see Elastic Fabric Adapter for HPC and ML workloads on Amazon EC2 in the *Amazon*

Elastic Compute Cloud User Guide. You can also use the AWS CLI to view a list of instance types that support EFA. Replace *region-code* with the AWS Region where you use AWS PCS, such as `us-east-1`.

```
aws ec2 describe-instance-types \
   --region region-code \
   --filters Name=network-info.efa-supported,Values=true \
   --query "InstanceTypes[*].[InstanceType]" \
   --output text | sort
```

> **Note**
>
> **Determine how many network interfaces are available** – Some EC2 instances have multiple network cards. This allows them to have multiple EFAs. For more information, see Multiple network interfaces in AWS PCS.

Create a security group to support EFA communications

AWS CLI

You can use the following AWS CLI command to create a security group that supports EFA. The command outputs a security group ID. Make the following replacements:

- *region-code* – Specify the AWS Region where you use AWS PCS, such as `us-east-1`.
- *vpc-id* – Specify the ID of the VPC that you use for AWS PCS.
- *efa-group-name* – Provide your chosen name for the security group.

```
aws ec2 create-security-group \
    --group-name efa-group-name \
    --description "Security group to enable EFA traffic" \
    --vpc-id vpc-id \
    --region region-code
```

Use the following commands to attach inbound and outbound security group rules. Make the following replacement:

- *efa-secgroup-id* – Provide the ID of the EFA security group you just created.

```
aws ec2 authorize-security-group-ingress \
    --group-id efa-secgroup-id \
    --protocol -1 \
    --source-group efa-secgroup-id

aws ec2 authorize-security-group-egress \
    --group-id efa-secgroup-id \
    --protocol -1 \
    --source-group efa-secgroup-id
```

CloudFormation template

You can use a CloudFormation template to create a security group that supports EFA. Download the template from the following URL, then upload it into the AWS CloudFormation console.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/pcs/enable_efa/assets/efa-
sg.yaml
```

With the template open in the AWS CloudFormation console, enter the following options.

- Under **Provide a stack name**
  - Under **Stack name**, enter a name such as `efa-sg-stack`.
- Under **Parameters**
  - Under **SecurityGroupName**, enter a name such as `efa-sg`.
  - Under **VPC**, select the VPC where you will use AWS PCS.

Finish creating the CloudFormation stack and monitor its status. When it reaches `CREATE_COMPLETE` the EFA security group is ready for use.

(Optional) Create a placement group

We recommended you launch all instances that use EFA in a cluster placement group to minimize the physical distance between them. Create a placement group for each compute node group where you plan to use EFA. See Placement groups for EC2 instances in AWS PCS to create a placement group for your compute node group.

Create or update an EC2 launch template

EFA network interfaces are set up in the EC2 launch template for an AWS PCS compute node group. If there are multiple network cards, multiple EFAs can be configured. The EFA security group and the optional placement group are included in the launch template as well.

Here is an example launch template for instances with two network cards, such as **hpc7a.96xlarge**. The instances will be launched in `subnet-SubnetID1` in cluster placement group `pg-PlacementGroupId1`.

Security groups must be added specifically to each EFA interface. Every EFA needs the security group that enables EFA traffic (`sg-EfaSecGroupId`). Other security groups, especially ones that handle regular traffic like SSH or HTTPS, only need to be attached to the primary network interface (designated by a `DeviceIndex` of `0`). Launch templates where network interfaces are defined do not support setting security groups using the `SecurityGroupIds` parameter—you must set a value for `Groups` in each network interface that you configure.

```
{
    "Placement": {
        "GroupId": "pg-PlacementGroupId1"
    },
    "NetworkInterfaces": [
        {
            "DeviceIndex": 0,
            "InterfaceType": "efa",
            "NetworkCardIndex": 0,
            "SubnetId": "subnet-SubnetId1",
            "Groups": [
                "sg-SecurityGroupId1",
                "sg-EfaSecGroupId"
            ]
        },
        {
            "DeviceIndex": 1,
            "InterfaceType": "efa",
            "NetworkCardIndex": 1,
            "SubnetId": "subnet-SubnetId1"
            "Groups": ["sg-EfaSecGroupId"]
        }
    ]
}
```

Create or update compute node groups for EFA

Your AWS PCS compute node groups must contain instances that have the same number of vCPUs, processor architecture, and EFA support. Configure the compute node group to use the AMI with the EFA software installed on it, and to use the launch template that configures EFA-enabled network interfaces.

(Optional) Test EFA

You can demonstrate EFA-enabled communication between two nodes in a compute node group by running the `fi_pingpong` program, which is included in the EFA software installation. If this test is successful, it is likely that EFA is configured properly.

To start, you need two running instances in the compute node group. If your compute node group uses static capacity, there should be already be instances available. For a compute node group that uses dynamic capacity, you can launch two nodes using the `salloc` command. Here is an example from a cluster with a dynamic node group named `hpc7g` associated with a queue named `all`.

```
% salloc --nodes 2 -p all
salloc: Granted job allocation 6
salloc: Waiting for resource configuration
... a few minutes pass ...
salloc: Nodes hpc7g-[1-2] are ready for job
```

Find out the IP address for the two allocated nodes using `scontrol`. In the example that follows, the addresses are `10.3.140.69` for `hpc7g-1` and `10.3.132.211` for `hpc7g-2`.

```
% scontrol show nodes hpc7g-[1-2]
NodeName=hpc7g-1 Arch=aarch64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=64 CPUTot=64 CPULoad=0.00
   AvailableFeatures=hpc7g
   ActiveFeatures=hpc7g
   Gres=(null)
   NodeAddr=10.3.140.69 NodeHostName=ip-10-3-140-69 Version=23.11.8
   OS=Linux 5.10.218-208.862.amzn2.aarch64 #1 SMP Tue Jun 4 16:52:10 UTC 2024
   RealMemory=124518 AllocMem=0 FreeMem=110763 Sockets=64 Boards=1
   State=IDLE+CLOUD ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=efa
   BootTime=2024-07-02T19:00:09 SlurmdStartTime=2024-07-08T19:33:25
   LastBusyTime=2024-07-08T19:33:25 ResumeAfterTime=None
   CfgTRES=cpu=64,mem=124518M,billing=64
```

```
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/a ExtSensorsWatts=0 ExtSensorsTemp=n/a
   Reason=Maintain Minimum Number Of Instances [root@2024-07-02T18:59:00]
   InstanceId=i-04927897a9ce3c143 InstanceType=hpc7g.16xlarge

NodeName=hpc7g-2 Arch=aarch64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=64 CPUTot=64 CPULoad=0.00
   AvailableFeatures=hpc7g
   ActiveFeatures=hpc7g
   Gres=(null)
   NodeAddr=10.3.132.211 NodeHostName=ip-10-3-132-211 Version=23.11.8
   OS=Linux 5.10.218-208.862.amzn2.aarch64 #1 SMP Tue Jun 4 16:52:10 UTC 2024
   RealMemory=124518 AllocMem=0 FreeMem=110759 Sockets=64 Boards=1
   State=IDLE+CLOUD ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=efa
   BootTime=2024-07-02T19:00:09 SlurmdStartTime=2024-07-08T19:33:25
   LastBusyTime=2024-07-08T19:33:25 ResumeAfterTime=None
   CfgTRES=cpu=64,mem=124518M,billing=64
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/a ExtSensorsWatts=0 ExtSensorsTemp=n/a
   Reason=Maintain Minimum Number Of Instances [root@2024-07-02T18:59:00]
   InstanceId=i-0a2c82623cb1393a7 InstanceType=hpc7g.16xlarge
```

Connect to one of the nodes (in this example case, `hpc7g-1`) using SSH (or SSM). Note that this is an internal IP address, so you may need to connect from one of your login nodes if you use SSH. Also be aware that the instance needs to be configured with an SSH key by way of the compute node group launch template.

```
% ssh ec2-user@10.3.140.69
```

Now, launch `fi_pingpong` in server mode.

```
/opt/amazon/efa/bin/fi_pingpong -p efa
```

Connect to the second instance (`hpc7g-2`).

```
% ssh ec2-user@10.3.132.211
```

Run `fi_pingpong` in client mode, connecting to the server on `hpc7g-1`. You should see output that resembles the example below.

```
% /opt/amazon/efa/bin/fi_pingpong -p efa 10.3.140.69

bytes   #sent   #ack     total       time     MB/sec    usec/xfer   Mxfers/sec
64      10      =10      1.2k        0.00s      3.08      20.75        0.05
256     10      =10      5k          0.00s     21.24      12.05        0.08
1k      10      =10      20k         0.00s     82.91      12.35        0.08
4k      10      =10      80k         0.00s    311.48      13.15        0.08
[error] util/pingpong.c:1876: fi_close (-22) fid 0
```

(Optional) Use a CloudFormation template to create an EFA-enabled launch template

Because there are several dependencies to setting up EFA, a CloudFormation template has been provided that you can use to configure a compute node group. It supports instances with up to four network cards. To learn more about instances with multiple network cards, see Elastic network interfaces in the *Amazon Elastic Compute Cloud User Guide*.

Download the CloudFormation template from the following URL, then upload it to the CloudFormation console in the AWS Region where you use AWS PCS.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/pcs/enable_efa/assets/pcs-lt-
efa.yaml
```

With the template open in the AWS CloudFormation console, enter the following values. Note that the template will provide some default parameter values—you can leave them as their default values.

- Under **Provide a stack name**
  - Under **Stack name**, enter a descriptive name. We recommend incorporating the name you will choose for your AWS PCS compute node group, such as *NODEGROUPNAME*`-efa-lt`.
- Under **Parameters**
  - Under **NumberOfNetworkCards**, choose the number of network cards in the instances that will be in your node group.
  - Under **VpcId**, choose the VPC where your AWS PCS cluster is deployed.

  - Under **NodeGroupSubnetId**, choose the subnet in your cluster VPC where EFA-enabled instances will be launched.
  - Under **PlacementGroupName**, leave the field blank to create a new cluster placement group for the node group. If you have an existing placement group you want to use, enter its name here.
  - Under **ClusterSecurityGroupId**, choose the security group you are using to allow access to other instances in the cluster and to the AWS PCS API. Many customers choose the default security group from their cluster VPC.
  - Under **SshSecurityGroupId**, provide the ID for a security group you are using to allow inbound SSH access to nodes in your cluster.
  - For **SshKeyName**, select the SSH keypair for access to nodes in your cluster.
  - For **LaunchTemplateName**, enter a descriptive name for the launch template such as `*NODEGROUPNAME*-efa-lt`. The name must be unique to your AWS account in the AWS Region where you will use AWS PCS.
- Under **Capabilities**
  - Check the box for **I acknowledge that AWS CloudFormation might create IAM resources**.

Monitor the status of the CloudFormation stack. When it reaches `CREATE_COMPLETE` the launch template is ready to be used. Use it with an AWS PCS compute node group, as described above in Create or update compute node groups for EFA.

Using network file systems with AWS PCS

You can attach network file systems to nodes launched in an AWS Parallel Computing Service (AWS PCS) compute node group to provide a persistent location where data and files can be written and accessed. You can use file systems provided by AWS services, including Amazon Elastic File System (Amazon EFS), Amazon FSx for OpenZFS, Amazon FSx for Lustre, and Amazon File Cache. You can also use self-managed file systems, such as NFS servers.

This topic covers considerations for and examples of using network file systems with AWS PCS.

Considerations for using network file systems

The implementation details for various file systems are different, but there are some common considerations.

- The relevant file system software must be installed on the instance. For example, to use Amazon FSx for Lustre, the appropriate Lustre package should be present. This can be accomplished by including it in the compute node group AMI or using a script that runs at instance boot.
- There must be a network route between the shared network file system and the compute node group instances.
- The security group rules for both the shared network file system and the compute node group instances must allow connections to the relevant ports.
- You must maintain a consistent POSIX user and group namespace across resources that access the file systems. Otherwise, jobs and interactive processes that run on your PCS cluster may encounter permissions errors.
- File system mounts are done using EC2 launch templates. Errors or timeouts in mounting a network file system may prevent instances from becoming available to run jobs. This, in turn, may lead to unexpected costs. For more information about debugging launch templates, see Using Amazon EC2 launch templates with AWS PCS.

Example network mounts

You can create file systems using Amazon EFS, Amazon FSx for Lustre, Amazon FSx for OpenZFS, and Amazon File Cache. Expand the relevant section below to see an example of each network mount.

Amazon EFS

File system setup

Create an Amazon EFS file system. Make sure it has a mount target in each Availability Zone where you will launch PCS compute node group instances. Also ensure each mount target is associated with a security group that allows inbound and outbound access from the PCS compute node group instances. For more information, see [Mount targets and security groups]() in the *Amazon Elastic File System User Guide*.

Launch template

Add the security group(s) from your file system setup to the launch template you will use for the compute node group.

Include user data that uses `cloud-config` mechanism to mount the Amazon EFS file system. Replace the following values in this script with your own details:

- *`mount-point-directory`* – The path on a each instance where you will mount Amazon EFS
- *`filesystem-id`* – The file system ID for the EFS file system

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
  - amazon-efs-utils

runcmd:
  - mkdir -p /mount-point-directory
  - echo "filesystem-id:/ /mount-point-directory efs tls,_netdev" >> /etc/fstab
  - mount -a -t efs defaults

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

Amazon FSx for Lustre

File system setup

Create an FSx for Lustre file system in the VPC where you will use AWS PCS. To minimize inter-zone transfers, deploy in a subnet in the same Availability Zone where you will launch the majority of your PCS compute node group instances. Ensure the file system is associated with a security group that allows inbound and outbound access from the PCS compute node group instances. For more information on security groups, see [File system access control with Amazon VPC](#) in the *Amazon FSx for Lustre User Guide*.

Launch template

Include user data that uses `cloud-config` to mount the FSx for Lustre file system. Replace the following values in this script with your own details:

- *`mount-point-directory`* – The path on an instance where you want to mount FSx for Lustre
- *`filesystem-id`* – The file system ID for the FSx for Lustre file system
- *`mount-name`* – The mount name for the FSx for Lustre file system
- *`region-code`* – The AWS Region where the FSx for Lustre file system is deployed (must be the same as your AWS PCS system)
- (Optional) *`latest`* – Any version of Lustre supported by FSx for Lustre

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- amazon-linux-extras install -y lustre=latest
- mkdir -p /mount-point-directory
- mount -t lustre filesystem-id.fsx.region-code.amazonaws.com@tcp:/mount-name /mount-point-directory

--==MYBOUNDARY==
```

Amazon FSx for OpenZFS

File system setup

Create an FSx for OpenZFS file system in the VPC where you will use AWS PCS. To minimize inter-zone transfers, deploy in a subnet in the same Availability Zone where you will launch the majority of your AWS PCS compute node group instances. Make sure the file system is associated with a security group that allows inbound and outbound access from the AWS PCS compute node group instances. For more information on security groups, see Managing file system access with Amazon VPC in the *FSx for OpenZFS User Guide*.

Launch template

Include user data that uses `cloud-config` to mount the root volume for an FSx for OpenZFS file system. Replace the following values in this script with your own details:

- *`mount-point-directory`* – The path on an instance where you want to mount your FSx for OpenZFS share
- *`filesystem-id`* – The file system ID for the FSx for OpenZFS file system
- *`region-code`* – The AWS Region where the FSx for OpenZFS file system is deployed (must be the same as your AWS PCS system)

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- mkdir -p /mount-point-directory
- mount -t nfs -o noatime,nfsvers=4.2,sync,rsize=1048576,wsize=1048576 filesystem-id.fsx.region-code.amazonaws.com:/fsx/ /mount-point-directory

--==MYBOUNDARY==
```

Amazon File Cache

File system setup

Create an Amazon File Cache in the VPC where you will use AWS PCS. To minimize inter-zone transfers, choose a subnet in the same Availability Zone where you will launch the majority of your PCS compute node group instances. Ensure the File Cache is associated with a security group that allows inbound and outbound traffic on port 988 between your PCS instances and the File Cache.

For more information on security groups, see [Cache access control with Amazon VPC](#) in the *Amazon File Cache User Guide*.

Launch template

Add the security group(s) from your file system setup to the launch template you will use for the compute node group.

Include user data that uses `cloud-config` to mount the Amazon File Cache. Replace the following values in this script with your own details:

- *mount-point-directory* – The path on an instance where you want to mount FSx for Lustre
- *cache-dns-name* – The Domain Name System (DNS) name for the File Cache
- *mount-name* – The mount name for the File Cache

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- amazon-linux-extras install -y lustre=2.12
- mkdir -p /mount-point-directory
- mount -t lustre -o relatime,flock cache-dns-name@tcp:/mount-name /mount-point-
directory

--==MYBOUNDARY==
```

Amazon Machine Images (AMIs) for AWS PCS

AWS PCS works with AMIs that you provide, affording great flexibility in the software and configuration found on nodes in your cluster. If you are trying out AWS PCS, you can use a sample AMI provided by and maintained by AWS. If you are using AWS PCS in production, we recommend you build your own AMIs. This topic covers how to discover and use the sample AMIs, as well as how to build and use your own customized AMIs.

Topics

Using sample Amazon Machine Images (AMIs) with AWS PCS

AWS provides sample AMIs that you can use as a starting point for working with AWS PCS.

> **Important**
>
> Sample AMIs are for demonstration purposes and are not recommended for production workloads.

Find current AWS PCS sample AMIs

AWS Management Console

AWS PCS sample AMIs have the following naming convention:

```
aws-pcs-sample_ami-OS-architecture-scheduler-scheduler-major-version
```

Accepted values

- *OS* – amzn2
- *architecture* – x86_64 or arm64

- *scheduler* – slurm
- *scheduler-major-version* – 24.05

To find AWS PCS sample AMIs

1. Open the Amazon EC2 console.
2. Navigate to **AMIs**.
3. Choose **Public images**.
4. In **Find AMI by attribute or tag**, search for an AMI using the templated name.

   Examples

   - Sample AMI for Slurm 24.05 on Arm64 instances

     ```
     aws-pcs-sample_ami-amzn2-arm64-slurm-24.05
     ```

   - Sample AMI for Slurm 24.05 on x86 instances

     ```
     aws-pcs-sample_ami-amzn2-x86_64-slurm-24.05
     ```

   Note

   If there are multiple AMIs, use the AMI with the most recent time stamp.

5. Use the AMI ID when you create or update a compute node group.

AWS CLI

You can find the latest AWS PCS sample AMI with the commands that follow. Replace *region-code* with the AWS Region where you use AWS PCS, such as us-east-1.

- **x86_64**

```
aws ec2 describe-images --region region-code --owners amazon \
--filters 'Name=name,Values=aws-pcs-sample_ami-amzn2-x86_64-slurm-24.05*' \
            'Name=state,Values=available' \
--query 'sort_by(Images, &CreationDate)[-1].[Name,ImageId]' --output text
```

- **Arm64**

```
aws ec2 describe-images --region region-code --owners amazon \
--filters 'Name=name,Values=aws-pcs-sample_ami-amzn2-arm64-slurm-24.05*' \
            'Name=state,Values=available' \
--query 'sort_by(Images, &CreationDate)[-1].[Name,ImageId]' --output text
```

Use the AMI ID when you create or update a compute node group.

Learn more about AWS PCS sample AMIs

To view the contents, configuration details for current and previous releases of the AWS PCS sample AMIs, see Release notes for AWS PCS sample AMIs.

Build your own AMIs compatible with AWS PCS

To learn how to build your own AMIs that work with AWS PCS, see Custom Amazon Machine Images (AMIs) for AWS PCS.

Custom Amazon Machine Images (AMIs) for AWS PCS

AWS PCS is designed to work with Amazon Machine Images (AMI) that you bring to the service. These AMIs can have arbitrary software and configurations installed on them, so long as they have the AWS PCS agent and a compatible version of Slurm installed and configured correctly. You must use AWS-provided installers to install the AWS PCS software on your custom AMI. We recommend you use AWS-provided installers to install Slurm on your custom AMI but you can install Slurm on your own if you prefer (not recommended).

> **Note**
>
> If you want to try AWS PCS without building a custom AMI, you can use a sample AMI provided by AWS. For more information, see Using sample Amazon Machine Images (AMIs) with AWS PCS.

This tutorial helps you create an AMI that can be used with PCS compute node groups to power your HPC and AI/ML workloads.

Topics

Step 1 – Launch a temporary instance

Launch a temporary instance that you can use to install and configure the AWS PCS software and Slurm scheduler. You use this instance to create an AMI compatible with AWS PCS.

To launch a temporary instance

1. Open the Amazon EC2 console.
2. In the navigation pane, choose **Instances**, then choose **Launch instances** to open the new launch instance wizard.
3. (Optional) In the **Name and tags** section, provide a name for the instance, such as `PCS-AMI-instance`. The name is assigned to the instance as a resource tag (`Name=PCS-AMI-instance`).
4. In the **Application and OS Images** section, select an AMI for one of the supported operating systems.
5. In the **Instance type** section, select a supported instance type.
6. In the **Key pair** section, select the key pair to use for the instance.
7. In the **Network settings** section:
   - For **Firewall (security groups)**, choose **Select existing security group**, then select a security group that allows inbound SSH access to your instance.
8. In the **Storage** section, configure the volumes as needed. Make sure to configure sufficient space to install your own applications and libraries.
9. In the **Summary** panel, choose **Launch instance**.

Step 2 – Install the AWS PCS agent

Install the agent that configures the instances launched by AWS PCS for use with Slurm.

To install the AWS PCS agent

1. Connect to the instance you launched. For more information, see Connect to your Linux instance.
2. (Optional) To ensure that all of your software packages are up to date, perform a quick software update on your instance. This process may take a few minutes.
   - **Amazon Linux 2, RHEL 9, Rocky Linux 9**

     ```
     sudo yum update -y
     ```
   - **Ubuntu 22.04**

     ```
     sudo apt-get update && sudo apt-get upgrade -y
     ```
3. Reboot the instance and reconnect to it.
4. Download the AWS PCS agent installation files. The installation files are packaged into a compressed tarball (`.tar.gz`) file. To download the latest *stable* version, use the following command. Substitute *`region`* with the AWS Region where you launched your temporary instance, such as `us-east-1`.

   ```
   curl https://aws-pcs-repo-region.s3.amazonaws.com/aws-pcs-agent/aws-pcs-agent-v1.1.1-1.tar.gz -o aws-pcs-agent-v1.1.1-1.tar.gz
   ```

   You can also get the latest version by replacing the version number with `latest` in the preceding command (for example: `aws-pcs-agent-v1-latest.tar.gz`).

   > **Note**
   >
   > This might change in future releases of the AWS PCS agent software.
5. (Optional) Verify the authenticity and integrity of the AWS PCS software tarball. We recommend that you do this to verify the identity of the software publisher and to check that the file has not been altered or corrupted since it was published.

a. Download the public GPG key for AWS PCS and import it into your keyring. Substitute *region* with the AWS Region where you launched your temporary instance. The command should return a key value. Record the key value; you use it in the next step.

```
wget https://aws-pcs-repo-public-keys-region.s3.amazonaws.com/aws-pcs-public-key.pub && \
    gpg --import aws-pcs-public-key.pub
```

b. Run the following command to verify the GPG key's fingerprint.

```
gpg --fingerprint 7EEF030EDDF5C21C
```

The command should return a fingerprint that is identical to the following:

```
1C24 32C1 862F 64D1 F90A  239A 7EEF 030E DDF5 C21C
```

Important

Don't run the AWS PCS agent installation script if the fingerprint doesn't match. Contact AWS Support.

c. Download the signature file and verify the signature of the AWS PCS software tarball file. Replace *region* with the AWS Region where you launched your temporary instance, such as `us-east-1`.

```
wget https://aws-pcs-repo-region.s3.amazonaws.com/aws-pcs-agent/aws-pcs-agent-v1.1.1-1.tar.gz.sig && \
    gpg --verify ./aws-pcs-agent-v1.1.1-1.tar.gz.sig
```

The output should be similar to the following:

```
gpg: assuming signed data in './aws-pcs-agent-v1.1.1-1.tar.gz'
gpg: Signature made Fri Dec 13 18:50:19 2024 CEST
gpg:                using RSA key 4BAA531875430EB0739E6D961BA7F0AF6E34C496
gpg: Good signature from "AWS PCS Packages (AWS PCS Packages)" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 1C24 32C1 862F 64D1 F90A  239A 7EEF 030E DDF5 C21C
```

```
Subkey fingerprint: 4BAA 5318 7543 0EB0 739E  6D96 1BA7 F0AF 6E34 C496
```

If the result includes `Good signature` and the fingerprint matches the fingerprint returned in the previous step, proceed to the next step.

> **Important**
>
> Don't run the AWS PCS software installation script if the fingerprint doesn't match. Contact AWS Support.

6. Extract the files from the compressed `.tar.gz` file and navigate to the extracted directory.

```
tar -xf aws-pcs-agent-v1.1.1-1.tar.gz && \
    cd aws-pcs-agent
```

7. Install the AWS PCS software.

```
sudo ./installer.sh
```

8. Check the AWS PCS software version file to confirm a successful installation.

```
cat /opt/aws/pcs/version
```

The output should be similar to the following:

```
AGENT_INSTALL_DATE='Fri Dec 13 12:28:43 UTC 2024'
AGENT_VERSION='1.1.1'
AGENT_RELEASE='1'
```

Step 3 – Install Slurm

Install a version of Slurm that is the compatible with AWS PCS.

> **Note**
>
> If you have an AMI with a previous version of the Slurm software installed on it, you must perform the following steps to install the new version of Slurm. The AWS PCS agent

enables the correct version of the Slurm binaries at runtime, according to the Slurm version configured at cluster creation time.

To install Slurm

1. Connect to the same temporary instance where you installed the AWS PCS software.
2. Download the Slurm installer software. The Slurm installer is packaged into a compressed tarball (`.tar.gz`) file. To download the latest *stable* version, use the following command. Substitute *`region`* with the AWS Region of your temporary instance, such as `us-east-1`.

```
curl https://aws-pcs-repo-region.s3.amazonaws.com/aws-pcs-slurm/aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz \
    -o aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz
```

You can also get the latest version by replacing the version number with `latest` in the preceding command (for example: `aws-pcs-slurm-24.05-installer-latest.tar.gz`).

Note

This might change in future releases of the Slurm installer software.

3. (Optional) Verify the authenticity and integrity of the Slurm installer tarball. We recommend that you do this to verify the identity of the software publisher and to check that the file has not been altered or corrupted since it was published.

   a. Download the public GPG key for AWS PCS and import it into your keyring. Substitute *`region`* with the AWS Region where you launched your temporary instance. The command should return a key value. Record the key value; you use it in the next step.

```
wget https://aws-pcs-repo-public-keys-region.s3.amazonaws.com/aws-pcs-public-key.pub && \
    gpg --import aws-pcs-public-key.pub
```

   b. Run the following command to verify the GPG key's fingerprint.

```
gpg --fingerprint 7EEF030EDDF5C21C
```

   The command should return a fingerprint that is identical to the following:

```
1C24 32C1 862F 64D1 F90A  239A 7EEF 030E DDF5 C21C
```

> **Important**
>
> Don't run the Slurm installation script if the fingerprint doesn't match. Contact [AWS Support](https://aws.amazon.com/contact-us/).

c. Download the signature file and verify the signature of the Slurm installer tarball file. Replace *region* with the AWS Region where you launched your temporary instance, such as `us-east-1`.

```
wget https://aws-pcs-repo-region.s3.amazonaws.com/aws-pcs-slurm/aws-pcs-
slurm-24.05-installer-24.05.5-2.tar.gz.sig && \
    gpg --verify ./aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz.sig
```

The output should be similar to the following:

```
gpg: assuming signed data in './aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz'
gpg: Signature made Wed Dec 18 14:23:38 2024 CEST
gpg:                using RSA key 4BAA531875430EB0739E6D961BA7F0AF6E34C496
gpg: Good signature from "AWS PCS Packages (AWS PCS Packages)" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 1C24 32C1 862F 64D1 F90A  239A 7EEF 030E DDF5 C21C
   Subkey fingerprint: 4BAA 5318 7543 0EB0 739E  6D96 1BA7 F0AF 6E34 C496
```

If the result includes `Good signature` and the fingerprint matches the fingerprint returned in the previous step, proceed to the next step.

> **Important**
>
> Don't run the Slurm installation script if the fingerprint doesn't match. Contact [AWS Support](https://aws.amazon.com/contact-us/).

4. Extract the files from the compressed `.tar.gz` file and navigate into the extracted directory.

```
tar -xf aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz && \
```

```
cd aws-pcs-slurm-24.05-installer
```

5. Install Slurm. The installer downloads, compiles, and installs Slurm and its dependencies. It takes several minutes, depending on the specifications of the temporary instance you selected.

```
sudo ./installer.sh -y
```

6. Check the scheduler version file to confirm the installation.

```
cat /opt/aws/pcs/scheduler/slurm-24.05/version
```

The output should be similar to the following:

```
SLURM_INSTALL_DATE='Wed Dec 18 12:38:56 UTC 2024'
SLURM_VERSION='24.05.5'
PCS_SLURM_RELEASE='2'
```

Step 4 – (Optional) Install additional drivers, libraries, and application software

Install additional drivers, libraries, and application software on the temporary instance. The installation procedures will vary depending on the specific applications and libraries. If you have not built a custom AMI for AWS PCS before, we recommend you first build and test an AMI with just the AWS PCS software and Slurm installed, then incrementally add your own software and configurations once you have confirmed initial success.

Examples

- Elastic Fabric Adapter (EFA) software. For more information, see Get started with EFA and MPI for HPC workloads on Amazon EC2 in the *Amazon Elastic Compute Cloud User Guide*.
- Amazon Elastic File System (Amazon EFS) client. For more information, see Manually installing the Amazon EFS client in the *Amazon Elastic File System User Guide*.
- Lustre client, to use Amazon FSx for Lustre and Amazon File Cache. For more information, see Installing the Lustre client in the *FSx for Lustre User Guide*.
- Amazon CloudWatch agent, to use CloudWatch Logs and Metrics. For more information, see Install the CloudWatch agent in the *Amazon CloudWatch User Guide*.

- AWS Neuron, to use **trn*** and **inf*** instance types. For more information, see the AWS Neuron documentation.
- NVIDIA Driver, CUDA, and DCGM, to use **p*** or **g*** instance types.

Step 5 – Create an AMI compatible with AWS PCS

After you have installed the required software components, you create an AMI that you can reuse to launch instances in AWS PCS compute node groups.

To create an AMI from your temporary instance

1. Open the Amazon EC2 console.
2. In the navigation pane, choose **Instances**.
3. Select the temporary instance that you created. Choose **Actions**, **Image**, **Create image**.
4. For **Create image**, do the following:
   a. For **Image name**, enter a descriptive name for the AMI.
   b. (Optional) For **Image description**, enter a brief description of the purpose of the AMI.
   c. Choose **Create image**.
5. In the navigation pane, choose **AMIs**.
6. Locate the AMI tht you created in the list. Wait for its status to change from **Pending** to **Available**, then use it with a AWS PCS compute node group.

Step 6 – Use the custom AMI with an AWS PCS compute node group

You can use your custom AMI with a new or existing AWS PCS compute node group.

New compute node group

To use the custom AMI

1. Open the AWS PCS console.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you will use the custom AMI, then select **Compute node groups**.

4. Create a new compute node group. For more information, see [Creating a compute node group in AWS PCS](). Under **AMI ID**, search for the name or ID of the custom AMI you want to use. Finish configuring the compute node group, then choose **Create compute node group**.
5. (Optional) Confirm the AMI supports instance launches. Launch an instance in the compute node group. You can do this by configuring the compute node group to have a single static instance, or you can submit a job to a queue that uses the compute node group.

    a. Check the Amazon EC2 console until an instance appears tagged with the new compute node group ID. For more information on this, see [Finding compute node group instances in AWS PCS]()..

    b. When you see an instance launch and complete its bootstrap process, confirm it is using the expected AMI. To do this, select the instance, then inspect **AMI ID** under **Details**. It should match the AMI you configured in the compute node group settings.

    c. (Optional) Update the compute node group scaling configuration to your preferred values.

Existing compute node group

To use the custom AMI

1. Open the [AWS PCS console]().
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you will use the custom AMI, then select **Compute node groups**.
4. Select the node group you wish to configure and choose **Edit**. Under **AMI ID**, search for the name or ID of the custom AMI you want to use. Finish configuring the compute node group, then choose **Update**. New instances launched in the compute node group will use the updated AMI ID. Existing instances will continue to use the old AMI until AWS PCS replaces them. For more information, see [Updating an AWS PCS compute node group]().
5. (Optional) Confirm the AMI supports instance launches. Launch an instance in the compute node group. You can do this by configuring the compute node group to have a single static instance, or you can submit a job to a queue that uses the compute node group.

    a. Check the Amazon EC2 console until an instance appears tagged with the new compute node group ID. For more information on this, see [Finding compute node group instances in AWS PCS]()..

b. When you see an instance launch and complete its bootstrap process, confirm it is using the expected AMI. To do this, select the instance, then inspect **AMI ID** under **Details**. It should match the AMI you configured in the compute node group settings.

c. (Optional) Update the compute node group scaling configuration to your preferred values.

Step 7 – Terminate the temporary instance

After you have confirmed that your AMI works as intended with AWS PCS, you can terminate the temporary instance to stop incurring charges for it.

To terminate the temporary instance

1. Open the Amazon EC2 console.
2. In the navigation pane, choose **Instances**.
3. Select the temporary instance that you created and choose **Actions**, **Instance state**, **Terminate instance**.
4. When prompted to confirm, choose **Terminate**.

Software installers to build custom AMIs for AWS PCS

AWS provides a downloadable file that can install the AWS PCS software on an instance. AWS also provides software that can download, compile, and install relevant versions of Slurm and its dependencies. You can use these instructions to build custom AMIs for use with AWS PCS or you can use your own methods.

Contents

AWS PCS software installer

The AWS PCS software installer configures an instance to work with AWS PCS during the instance bootstrap process. You must use AWS-provided installers to install the AWS PCS software on your custom AMI.

Slurm installer

The Slurm installer downloads, compiles, and installs relevant versions of Slurm and its dependencies. You can use the Slurm installer to build custom AMIs for AWS PCS. You can also use your own mechanisms if they are consistent with the software configuration that the Slurm installer provides.

The AWS-provided software installs the following:

- Slurm at the requested major and maintenance version (currently version 24.05.x) - License GPL 2
  - Slurm is built with `--sysconfdir` set to `/etc/slurm`
  - Slurm is built with the option `--enable-pam` and `--without-munge`
  - Slurm is built with the option `--sharedstatedir=/run/slurm/`
  - Slurm is built with PMIX and JWT support
  - Slurm is installed at `/opt/aws/pcs/schedulers/slurm-24.05`
- OpenPMIX (version 4.2.6) – License
  - OpenPMIX is installed as a subdirectory of `/opt/aws/pcs/scheduler/`
- libjwt (version 1.17.0) – License MPL-2.0
  - libjwt is installed as a subdirectory of `/opt/aws/pcs/scheduler/`

The AWS-provided software changes the system configuration as follows:

- The Slurm `systemd` file created by the build is copied to `/etc/systemd/system/` with file name `slurmd-24.05.service`.
- If they don't exist, a Slurm user and group (`slurm:slurm`) are created with UID/GID of `401`.
- On Amazon Linux 2 and Rocky Linux 9 the installation adds the EPEL repository to install the required software to build Slurm or its dependencies.

- On RHEL9 the installation will enable `codeready-builder-for-rhel-9-rhui-rpms` and `epel-release-latest-9` from `fedoraproject` to install the required software to build Slurm or its dependencies.

Supported operating systems

The AWS PCS software and Slurm installers suppport the following operating systems:

- Amazon Linux 2
- RedHat Enterprise Linux 9
- Rocky Linux 9
- Ubuntu 22.04

For more information, see Supported operating systems in AWS PCS.

> **Note**
>
> AWS Deep Learning AMIs (DLAMI) versions based on Amazon Linux 2 and Ubuntu 22.04 should be compatible with the AWS PCS software and Slurm installers. For more information, see Choosing Your DLAMI in the *AWS Deep Learning AMIs Developer Guide*.

Supported instance types

AWS PCS software and Slurm installers support any x86_64 or arm64 instance type than can run one of the supported operating systems.

Supported Slurm versions

The following major versions of Slurm are supported:

- Slurm 24.05
- Slurm 23.11

Verify installers using a checksum

You can use SHA256 checksums to verify the installer tarball (.tar.gz) files. We recommend that you do this to verify the identity of the software publisher and to check that the application has not been altered or corrupted since it was published.

To verify a tarball

Use the **sha256sum** utility for the SHA256 checksum and specify the tarball filename. You must run the command from the directory where you saved the tarball file.

- SHA256

  ```
  $ sha256sum tarball_filename.tar.gz
  ```

The command should return a checksum value in the following format.

```
checksum_value tarball_filename.tar.gz
```

Compare the checksum value returned by the command with the checksum value provided in the following table. If the checksums match, then it's safe to run the installation script.

> **Important**
>
> If the checksums don't match, don't run the installation script. Contact Support.

For example, the following command generates the SHA256 checksum for the Slurm 24.05.5-2 tarball.

```
$ sha256sum aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz
```

Example output:

```
7cc8d8294f2fbff95fe0602cf9e21e02003b5d96c0730e0a18c6aa04c7a4967b aws-pcs-slurm-24.05-installer-24.05.5-2.tar.gz
```

The following tables list the checksums for recent versions of the installers. Replace *us-east-1* with the AWS Region where you use AWS PCS.

AWS PCS agent

Installer	Download URL	SHA256 checksum
AWS PCS agent 1.1.1-1	`https://aws-pcs-repo- us-east-1 .s3.amazonaws.com/aws-pcs-agent/aws-pcs-agent-v1.1.1-1.tar.gz`	`bef078bf60a6d8ecde2e6c49cd34d088703f02550279e3bf483d57a235334dc6`
AWS PCS agent 1.1.0-1	`https://aws-pcs-repo- us-east-1 .s3.amazonaws.com/aws-pcs-agent/aws-pcs-agent-v1.1.0-1.tar.gz`	`594c32194c71bccc5d66e5213213ae38dd2c6d2f9a950bb01accea0bbab0873a`
AWS PCS agent 1.0.1-1	`https://aws-pcs-repo- us-east-1 .s3.amazonaws.com/aws-pcs-agent/aws-pcs-agent-v1.0.1-1.tar.gz`	`04e22264019837e3f42d8346daf5886eaacecd21571742eb505ea8911786bcb2`
AWS PCS agent 1.0.0-1	`https://aws-pcs-repo- us-east-1 .s3.amazonaws.com/aws-pcs-agent/aws-pcs-agent-v1.0.0-1.tar.gz`	`d2d3d68d00c685435c38af471d7e2492dde5ce9eb222d7b6ef0042144b134ce0`

Slurm installer

Installer	Download URL	SHA256 checksum
Slurm 24.05.5-2	`https://aws-pcs-repo- us-east-1 .s3.amazonaws.com/aws-pcs-slurm/aws-pcs-slurm`	`7cc8d8294f2fbff95fe0602cf9e21e02003b5d96c0730e0a18c6aa04c7a4967b`

Installer	Download URL	SHA256 checksum
	-24.05-installer-24.05.5-2.tar.gz	
Slurm 23.11.10-3	https://aws-pcs-repo- *us-east-1* .s3.amazonaws.com/aws-pcs-slurm/aws-pcs-slurm-23.11-installer-23.11.10-3.tar.gz	488a10ee0fbd57ec0e0ff7ea708a9e3038fafdc025c6bb391c75c2e2a7852a00
Slurm 23.11.10-2	https://aws-pcs-repo- *us-east-1* .s3.amazonaws.com/aws-pcs-slurm/aws-pcs-slurm-23.11-installer-23.11.10-2.tar.gz	0bbe85423305c05987931168caf98da08a34c25f9eec0690e8e74de0b7bc8752
Slurm 23.11.10-1	https://aws-pcs-repo- *us-east-1* .s3.amazonaws.com/aws-pcs-slurm/aws-pcs-slurm-23.11-installer-23.11.10-1.tar.gz	27e8faa9980e92cdfd8cfdc71f937777f0934552ce61e33dac4ecf5a20321e44
Slurm 23.11.9-1	https://aws-pcs-repo- *us-east-1* .s3.amazonaws.com/aws-pcs-slurm/aws-pcs-slurm-23.11-installer-23.11.9-1.tar.gz	1de7d919c8632fe8e2806611bed4fde1005a4fadc795412456e935c7bba2a9b8

Release notes for AWS PCS sample AMIs

AMIs for the latest supported major versions of the scheduler receive security updates and critical bug fixes. These incremental security patches aren't included in official release notes.

> **Important**
>
> Sample AMIs related to old scheduler versions aren't supported and don't receive updates.

> **Important**
>
> Sample AMIs are for demonstration purposes and are not recommended for production workloads.

Contents

AWS PCS sample AMIs for x86_64 (Amazon Linux 2)

Slurm 24.05

AMI name

- `aws-pcs-sample_ami-amzn2-x86_64-slurm-24.05`

Supported EC2 instances

- All instances with an 64-bit x86 processor. To find compatible instances, navigate to the Amazon EC2 console. Choose **Instance Types**, then search for `Architectures=x86_64`.

AMI contents

- Supported AWS Service: AWS PCS
- Operating System: Amazon Linux 2
- Compute Architecture: x86_64
- EBS volume type: gp2
- EFA Installer: 1.33.0

- GDRCopy: 2.4
- NVIDIA Driver: 550.127.08
- NVIDIA CUDA: 12.4.1_550.54.15

Slurm 23.11

AMI name

- `aws-pcs-sample_ami-amzn2-x86_64-slurm-23.11`

Supported EC2 instances

- All instances with an 64-bit x86 processor. To find compatible instances, navigate to the Amazon EC2 console. Choose **Instance Types**, then search for `Architectures=x86_64`.

AMI contents

- Supported AWS Service: AWS PCS
- Operating System: Amazon Linux 2
- Compute Architecture: x86_64
- EBS volume type: gp2
- EFA Installer: 1.33.0
- GDRCopy: 2.4
- NVIDIA Driver: 550.127.08
- NVIDIA CUDA: 12.4.1_550.54.15

AWS PCS sample AMIs for Arm64 (Amazon Linux 2)

Slurm 24.05

AMI name

- `aws-pcs-sample_ami-amzn2-arm64-slurm-24.05`

Supported EC2 instances

- All instances with an 64-bit Arm processor. To find compatible instances, navigate to the [Amazon EC2 console](). Choose **Instance Types**, then search for `Architectures=arm64`.

AMI contents

- Supported AWS Service: AWS PCS
- Operating System: Amazon Linux 2
- Compute Architecture: arm64
- EBS volume type: gp2
- EFA Installer: 1.33.0
- GDRCopy: 2.4
- NVIDIA Driver: 550.127.08
- NVIDIA CUDA: 12.4.1_550.54.15

Slurm 23.11

AMI name

- `aws-pcs-sample_ami-amzn2-arm64-slurm-23.11`

Supported EC2 instances

- All instances with an 64-bit Arm processor. To find compatible instances, navigate to the [Amazon EC2 console](). Choose **Instance Types**, then search for `Architectures=arm64`.

AMI contents

- Supported AWS Service: AWS PCS
- Operating System: Amazon Linux 2
- Compute Architecture: arm64
- EBS volume type: gp2
- EFA Installer: 1.33.0
- GDRCopy: 2.4

- NVIDIA Driver: 550.127.08
- NVIDIA CUDA: 12.4.1_550.54.15

Supported operating systems in AWS PCS

AWS PCS uses the Amazon Machine Image (AMI) configured for a compute node group to launch EC2 instances in that compute node group. The AMI determines the operating system that the EC2 instances use. You can't change the operating system in AWS PCS sample AMIs. You must create a custom AMI if you want to use a different operating system. For more information, see Amazon Machine Images (AMIs) for AWS PCS.

Supported operating systems

- **Amazon Linux 2**

  This is the operating system in the AWS PCS sample AMIs.

  > **Important**
  >
  > Sample AMIs are for demonstration purposes and are not recommended for production workloads. You should create and use a custom AMI for production workloads, even if you intend to use Amazon Linux 2.

- **RedHat Enterprise Linux 9 (RHEL 9)**

  The on-demand cost for RHEL any instance type is higher than for other supported operation systems. For more information about pricing, see On-Demand Pricing and How is Red Hat Enterprise Linux on Amazon Elastic Compute Cloud offered and priced?.

- **Rocky Linux 9**

  You can use the official Rocky Linux 9 AMIs as a base for a custom AMI. Your custom AMI build might fail if the base AMI doesn't have the latest kernel.

  To upgrade the kernel

  1. Launch an instance using a rocky9 AMI id from here: https://rockylinux.org/cloud-images/
  2. ssh into the instance and run the following command:

     ```
     sudo yum -y update
     ```

  3. Create an image from the instance. You specify this image as the `ParentImage` for your custom AMI.

- **Ubuntu 22.04**

  Ubuntu 22.04 requires more secure keys for SSH and doesn't support RSA keys by default. We recommend you generate and use an ED25519 key instead.

  > **Note**
  >
  > You can't update Ubuntu 22.04 to the latest kernel because there isn't an FSx client for that kernel.

Slurm versions in AWS PCS

SchedMD continually enhances Slurm with new capabilities, optimizations, and security patches. SchedMD releases a new major version at regular intervals and plans to support up to 3 versions at any given time. AWS PCS initially supports Slurm 23.11. AWS PCS is designed to automatically update the Slurm controller with patch versions.

When SchedMD ends support for a particular major version, AWS PCS also ends support for that major version. AWS PCS sends advance notice if a Slurm major version is close to its end of life, to help customers know when to upgrade their clusters to a newer supported version.

We recommend you use the latest supported Slurm version to deploy your cluster, to access the most recent advancements and improvements.

Frequently asked questions about Slurm versions

How long does AWS PCS support a Slurm version?

AWS PCS follows the SchedMD support cycles for major versions. AWS PCS supports up to 3 major versions at any given time. After SchedMD releases a new major version, AWS PCS retires the oldest supported version. AWS PCS releases a new major version of Slurm as soon as possible, but there might be a delay between the SchedMD release and its availability in AWS PCS.

When does AWS PCS notify me about the End of Support Life (EOSL) for Slurm versions?

AWS PCS notifies you multiple times, in a pre-determined cadence, before the EOSL date.

What do I have to do when a Slurm version approaches EOSL?

You must update your Slurm versions before EOSL to help maintain a secure and supported environment.

How can I update my clusters to use a new major version of Slurm?

To update the Slurm version, you must create a new cluster. You must also upgrade to the equivalent AWS PCS software in your Amazon Machine Image (AMI) and use it to create the compute node groups for your new cluster.

How will my clusters get new Slurm patch version releases?

AWS PCS is designed to automatically apply patches to address Slurm Common Vulnerabilities and Exposures (CVEs). AWS PCS applies the patches to cluster controllers that run in internal service-owned accounts. To install patches on EC2 instances in your AWS account, update the AMI for your compute node groups and update the compute node groups to use the updated AMI. For more information, see Custom Amazon Machine Images (AMIs) for AWS PCS.

> **Note**
>
> Slurm controllers are unavailable while we update them. Running jobs aren't affected. Jobs submitted when the cluster's controller is unavailable are held until the controller is available.

What if I don't update Slurm by the EOSL date?

AWS PCS is designed to stop clusters that have an unsupported Slurm version. You must update the Slurm major version of the cluster controller and the AWS PCS software installed on the compute node groups.

How many Slurm versions does AWS PCS support?

AWS PCS supports up to 3 major Slurm versions at any given time, including the current and 2 previous major versions.

What Slurm version updates should I apply?

We strongly recommend you use the same major version across all components in your cluster and install the latest patches as soon as they are released. The AMIs for your compute node groups must use a version of Slurm software compatible with the Slurm version of the cluster controller. The Slurm major version in your AMIs must be within 2 versions of the Slurm major version on the cluster controller. The Slurm version installed in the AMI and on the running EC2 instances in the cluster can't be newer than the Slurm version on the cluster controller. To maintain support for your cluster, your AMIs must use a supported AWS PCS software version.

What if I update the Slurm major version but use older Slurm software in my AMI for compute node groups?

You must update the AWS PCS software to the same version to use new Slurm functionality. For full AWS PCS support, all Slurm components must use supported versions. In summary:

- We are able to provide full support when the cluster controller and all components (AWS PCS packages) in your AWS account both use the supported versions.
- AWS PCS is designed to stop a cluster if the Slurm version of its controller reaches EOSL.
- If the Slurm version of components in your AWS account reach EOSL, your cluster won't be supported.

In what order should I update components in my Cluster?

You must update the Slurm version of your cluster controller before you use an AMI with a newer Slurm version. You update a compute node group to use the AMI. AWS PCS uses the AMI to launch new EC2 instances in the compute node group. AWS PCS doesn't update existing EC2 instances that have running jobs; AWS PCS is designed to terminate those instances after their jobs complete.

Does AWS PCS offer extended support for Slurm versions?

No. We will communicate detailed information about extended support options, including any additional costs and the specific support coverage provided.

Security in AWS Parallel Computing Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Parallel Computing Service, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS PCS. The following topics show you how to configure AWS PCS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS PCS resources.

Topics

Data protection in AWS Parallel Computing Service

The AWS shared responsibility model applies to data protection in AWS Parallel Computing Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS PCS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

Encryption is enabled by default for data at rest when you create an AWS Parallel Computing Service (AWS PCS) cluster with the AWS Management Console, AWS CLI, AWS PCS API, or AWS SDKs. AWS PCS uses an **AWS owned KMS key** to encrypt data at rest. For more information, see Customer keys and AWS keys in the *AWS KMS Developer Guide*. You can also use a customer managed key. For more information, see Required KMS key policy for use with encrypted EBS volumes in AWS PCS.

The **cluster secret** is stored in AWS Secrets Manager and is encrypted with the Secrets Manager managed KMS key. For more information, see Working with cluster secrets in AWS PCS.

In an AWS PCS cluster, the following data is *at rest*:

- **Scheduler state** – It includes data on running jobs and provisioned nodes in the cluster. This is the data that Slurm persists in the `StateSaveLocation` defined in your `slurm.conf`. For more information, see the description of StateSaveLocation in the Slurm documentation. AWS PCS deletes job data after a job completes.
- **Scheduler auth secret** – AWS PCS uses it to authenticate all scheduler communications in the cluster.

For scheduler state information, AWS PCS automatically encrypts data and metadata before it writes them to the file system. The encrypted file system uses industry-standard AES-256 encryption algorithm for data at rest.

Encryption in transit

Your connections to the AWS PCS API use TLS encryption with the Signature Version 4 signing process, regardless of whether you use the AWS Command Line Interface (AWS CLI) or AWS SDKs. For more information, see Signing AWS API requests in the *AWS Identity and Access Management User Guide*. AWS manages access control through the API with the IAM policies for the security credentials you use to connect.

AWS PCS uses TLS to connect to other AWS services.

Within a Slurm cluster, the scheduler is configured with the `auth/slurm` authentication plug-in that provides authentication for all scheduler communications. Slurm doesn't provide encryption at the application level for its communications, all data flowing across cluster instances stays local to the EC2 VPC and therefore is subject to VPC encryption if those instances support encryption in transit. For more information, see Encryption in transit in the *Amazon Elastic Compute Cloud User Guide*. Communication is encrypted between the controller (provisioned in a service account) the cluster nodes in your account.

Key management

AWS PCS uses an **AWS owned KMS key** to encrypt data. For more information, see Customer keys and AWS keys in the *AWS KMS Developer Guide*. You can also use a customer managed key. For more information, see Required KMS key policy for use with encrypted EBS volumes in AWS PCS.

The **cluster secret** is stored in AWS Secrets Manager and is encrypted with the Secrets Manager managed KMS key. For more information, see Working with cluster secrets in AWS PCS.

Inter-network traffic privacy

AWS PCS compute resources for a cluster reside within 1 VPC in the customer's account. Therefore, all internal AWS PCS service traffic within a cluster stays within the AWS network and doesn't travel across the internet. Communication between the user and AWS PCS nodes can travel across the internet and we recommend using SSH or Systems Manager to connect to the nodes. For more information, see What is AWS Systems Manager? in the *AWS Systems Manager User Guide*.

You can also use the following offerings to connect your on-premises network to AWS:

- AWS Site-to-Site VPN. For more information, see What is AWS Site-to-Site VPN? in the *AWS Site-to-Site VPN User Guide*.
- An AWS Direct Connect. For more information, see What is AWS Direct Connect? in the *AWS Direct Connect User Guide*.

You access the AWS PCS API to perform administrative tasks for the service. You and your users access the Slurm endpoint ports to interact with the scheduler directly.

Encrypting API traffic

To access the AWS PCS API, clients must support Transport Layer Security (TLS) 1.2 or later. We require TLS 1.2 and recommend TLS 1.3. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes. Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. You can also use AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Encrypting data traffic

Encryption of data in transit is enabled from supported EC2 instances accessing the scheduler endpoint and between ComputeNodeGroup instances from within the AWS Cloud. For more information, see Encryption in transit.

Required KMS key policy for use with encrypted EBS volumes in AWS PCS

AWS PCS uses service-linked roles to delegate permissions to other AWS services. The AWS PCS service-linked role is predefined and includes permissions that AWS PCS requires to call other AWS services on your behalf. The predefined permissions also include access to your AWS managed keys but not to your customer managed keys.

This topic describes how to set up the key policy required to launch instances when you specify a customer managed key for Amazon EBS encryption.

> **Note**
>
> AWS PCS doesn't require additional authorization to use the default AWS managed key to protect the encrypted volumes in your account.

Contents

Overview

You can use the following AWS KMS keys for Amazon EBS encryption when AWS PCS launches instances:

- AWS managed key – An encryption key in your account that Amazon EBS creates, owns, and manages. This is the default encryption key for a new account. Amazon EBS uses the AWS managed key for encryption unless you specify a customer managed key.
- Customer managed key – A custom encryption key that you create, own, and manage. For more information, see Create a KMS key in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> The key must be symmetric. Amazon EBS doesn't support asymmetric customer managed keys.

You configure customer managed keys when you create encrypted snapshots or a launch template that specifies encrypted volumes, or when you choose to enable encryption by default.

Configure key policies

Your KMS keys must have a key policy that allows AWS PCS to launch instances with Amazon EBS volumes encrypted with a customer managed key.

Use the examples on this page to configure a key policy to give AWS PCS access to your customer managed key. You can modify the customer managed key's key policy when you create the key or at a later time.

The key policy must have the following statements:

- A statement that allows the IAM identity specified in the `Principal` element to use the customer managed key directly. It includes permissions to perform the AWS KMS `Encrypt`, `Decrypt`, `ReEncrypt*`, `GenerateDataKey*`, and `DescribeKey` operations on the key.
- A statement that allows the IAM identity specified in the `Principal` element to use the `CreateGrant` operation to generate grants that delegate a subset of its own permissions to AWS services that are integrated with AWS KMS or another principal. This allows them to use the key to create encrypted resources on your behalf.

Don't change any existing statements in the policy when you add the new policy statements to your key policy.

For more information, see:

- create-key in the *AWS CLI Command Reference*
- put-key-policy in the *AWS CLI Command Reference*
- Find the key ID and key ARN in the *AWS Key Management Service Developer Guide*
- Service-linked roles for AWS PCS
- Amazon EBS encryption in the *Amazon EBS User Guide*
- AWS Key Management Service in the *AWS Key Management Service Developer Guide*

Example 1: Key policy sections that allow access to the customer managed key

Add the following policy statements to the key policy of the customer managed key. Replace the example ARN with the ARN of the your `AWSServiceRoleForPCS` service-linked role. This example policy gives the AWS PCS service-linked role (`AWSServiceRoleForPCS`) permissions to use the customer managed key.

```
{
   "Sid": "Allow service-linked role use of the customer managed key",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
           "arn:aws:iam::account-id:role/aws-service-role/pcs.amazonaws.com/
AWSServiceRoleForPCS"
       ]
   },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
       "kms:DescribeKey"
   ],
   "Resource": "*"
}
```

```
{
   "Sid": "Allow attachment of persistent resources",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
```

```
            "arn:aws:iam::account-id:role/aws-service-role/pcs.amazonaws.com/
AWSServiceRoleForPCS"
        ]
    },
    "Action": [
        "kms:CreateGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": true
        }
     }
}
```

Example 2: Key policy sections that allow cross-account access to the customer managed key

If you create a customer managed key in a different account than your AWS PCS cluster, you must use a **grant** in combination with the key policy to allow cross-account access to the key.

To grant access to the key

1. Add the following policy statements to the customer managed key's key policy. Replace the example ARN with the ARN of the other account. Replace *111122223333* with the actual account ID of the AWS account that you want to create the AWS PCS cluster in. This allows you to give an IAM user or role in the specified account permission to create a grant for the key using the CLI command that follows. By default, users don't have access to the key.

```
{.
   "Sid": "Allow external account 111122223333 use of the customer managed key",
   "Effect": "Allow",
   "Principal": {
       "AWS": [
           "arn:aws:iam::111122223333:root"
       ]
   },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
```

```
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

```
{
    "Sid": "Allow attachment of persistent resources in external
 account 111122223333",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::111122223333:root"
        ]
    },
    "Action": [
        "kms:CreateGrant"
    ],
    "Resource": "*"
}
```

2. From the account that you want to create the AWS PCS cluster in, create a grant that delegates the relevant permissions to the AWS PCS service-linked role. The value of `grantee-principal` is the ARN of the service-linked role. The value of `key-id` is the ARN of the key.

   The following example [create-grant](https://docs.aws.amazon.com/cli/latest/reference/kms/create-grant.html) CLI command gives the service-linked role named `AWSServiceRoleForPCS` in account *111122223333* permissions to use the customer managed key in account *444455556666*.

```
aws kms create-grant \
  --region us-west-2 \
  --key-id arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d \
  --grantee-principal arn:aws:iam::111122223333:role/aws-service-role/
pcs.amazonaws.com/AWSServiceRoleForPCS \
  --operations "Encrypt" "Decrypt" "ReEncryptFrom" "ReEncryptTo" "GenerateDataKey"
 "GenerateDataKeyWithoutPlaintext" "DescribeKey" "CreateGrant"
```

> **Note**
>
> The user making the request must have permissions to use the `kms:CreateGrant` action.

The following example IAM policy allows an IAM identity (user or role) in account *111122223333* to create a grant for the customer managed key in account *444455556666*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreationOfGrantForTheKMSKeyinExternalAccount444455556666",
      "Effect": "Allow",
      "Action": "kms:CreateGrant",
      "Resource": "arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    }
  ]
}
```

For more information about creating a grant for a KMS key in a different AWS account, see Grants in AWS KMS in the *AWS Key Management Service Developer Guide*.

> **Important**
>
> The service-linked role name specified as the grantee principal must be the name of an existing role. After creating the grant, to ensure that the grant allows AWS PCS to use the specified KMS key, do not delete and recreate the service-linked role.

Edit key policies in the AWS KMS console

The examples in the previous sections show only how to add statements to a key policy, which is just one way of changing a key policy. The easiest way to change a key policy is to use the AWS KMS console's default view for key policies and make an IAM identity (user or role) one of the *key users* for the appropriate key policy. For more information, see [Using the AWS Management Console default view] in the *AWS Key Management Service Developer Guide*.

> **Warning**
>
> The console's default view policy statements include permissions to perform AWS KMS `Revoke` operations on the customer managed key. If you revoke a grant that gave an AWS account access to a customer managed key in your account, users in that AWS account lose access to the encrypted data and the key.

Access AWS Parallel Computing Service using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and AWS Parallel Computing Service (AWS PCS). You can access AWS PCS as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access AWS PCS.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for AWS PCS.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for AWS PCS

Before you set up an interface endpoint for AWS PCS, review Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

AWS PCS supports making calls to all of its API actions through the interface endpoint.

If your VPC doesn't have direct internet access, you must configure a VPC endpoint to enable your compute node group instances to call the AWS PCS `RegisterComputeNodeGroupInstance` API action.

Create an interface endpoint for AWS PCS

You can create an interface endpoint for AWS PCS using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for AWS PCS using the following service name:

```
com.amazonaws.region.pcs
```

Replace *region* with the ID of the AWS Region to create the endpoint in, such as `us-east-1`.

If you enable private DNS for the interface endpoint, you can make API requests to AWS PCS using its default Regional DNS name. For example, `pcs.us-east-1.amazonaws.com`.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to AWS PCS through the interface endpoint. To control the access allowed to AWS PCS from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for AWS PCS actions

The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed AWS PCS actions for all principals to the cluster with the specified *cluster-id*. Replace *region* with the ID of the AWS Region of the cluster, such as `us-east-1`. Replace *account-id* with the AWS account number of the cluster.

```
{
    "Statement": [
```

```
            {
                "Action": [
                "pcs:CreateCluster",
                "pcs:ListClusters",
                "pcs:DeleteCluster",
                "pcs:GetCluster",
                ],
                "Effect": "Allow",
                "Principal": "*",
                "Resource": [
                    "arn:aws:pcs:region:account-id:cluster/cluster-id*"
                ]
            }
        ]
}
```

Identity and Access Management for AWS Parallel Computing Service

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS PCS resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS PCS.

Service user – If you use the AWS PCS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS PCS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS PCS, see Troubleshooting AWS Parallel Computing Service identity and access.

Service administrator – If you're in charge of AWS PCS resources at your company, you probably have full access to AWS PCS. It's your job to determine which AWS PCS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS PCS, see How AWS Parallel Computing Service works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS PCS. To view example AWS PCS identity-based policies that you can use in IAM, see Identity-based policy examples for AWS Parallel Computing Service.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating

IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource

(instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries –** A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs) –** RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about

Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.

- **Session policies –** Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Parallel Computing Service works with IAM

Before you use IAM to manage access to AWS PCS, learn what IAM features are available to use with AWS PCS.

IAM features you can use with AWS Parallel Computing Service

IAM feature	AWS PCS support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes

IAM feature	AWS PCS support
Service roles	No
Service-linked roles	Yes

To get a high-level view of how AWS PCS and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS PCS

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS PCS

To view examples of AWS PCS identity-based policies, see Identity-based policy examples for AWS Parallel Computing Service.

Resource-based policies within AWS PCS

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS PCS

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS PCS actions, see Actions Defined by AWS Parallel Computing Service in the *Service Authorization Reference*.

Policy actions in AWS PCS use the following prefix before the action:

```
pcs
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "pcs:action1",
      "pcs:action2"
         ]
```

Policy resources for AWS PCS

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS PCS resource types and their ARNs, see Resources Defined by AWS Parallel Computing Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS Parallel Computing Service.

To view examples of AWS PCS identity-based policies, see Identity-based policy examples for AWS Parallel Computing Service.

Policy condition keys for AWS PCS

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS PCS condition keys, see Condition Keys for AWS Parallel Computing Service in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS Parallel Computing Service .

To view examples of AWS PCS identity-based policies, see Identity-based policy examples for AWS Parallel Computing Service.

ACLs in AWS PCS

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AWS PCS

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AWS PCS

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AWS PCS

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS PCS

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS PCS functionality. Edit service roles only when AWS PCS provides guidance to do so.

Service-linked roles for AWS PCS

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing AWS PCS service-linked roles, see Service-linked roles for AWS PCS.

Identity-based policy examples for AWS Parallel Computing Service

By default, users and roles don't have permission to create or modify AWS PCS resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS PCS, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for AWS Parallel Computing Service in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS PCS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS PCS console

To access the AWS Parallel Computing Service console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS PCS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

For more information about minimum permissions required to use the AWS PCS console, see Minimum permissions for AWS PCS.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
```

```
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for AWS Parallel Computing Service

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AWSPCSServiceRolePolicy

You can't attach AWSPCSServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows AWS PCS to perform actions on your behalf. For more information, see Service-linked roles for AWS PCS.

Permissions details

This policy includes the following permissions.

- `ec2` – Allows AWS PCS to create and manage Amazon EC2 resources.
- `iam` – Allows AWS PCS to create a service-linked role for the Amazon EC2 fleet and to pass the role to Amazon EC2.
- `cloudwatch` – Allows AWS PCS to publish service metrics to Amazon CloudWatch.
- `secretsmanager` – Allows AWS PCS to manage secrets for AWS PCS cluster resources.

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

```
      "Sid" : "PermissionsToCreatePCSNetworkInterfaces",
      "Effect" : "Allow",
      "Action" : [
        "ec2:CreateNetworkInterface"
      ],
      "Resource" : "arn:aws:ec2:*:*:network-interface/*",
      "Condition" : {
        "Null" : {
          "aws:RequestTag/AWSPCSManaged" : "false"
        }
      }
    },
    {
      "Sid" : "PermissionsToCreatePCSNetworkInterfacesInSubnet",
      "Effect" : "Allow",
      "Action" : [
        "ec2:CreateNetworkInterface"
      ],
      "Resource" : [
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
    },
    {
      "Sid" : "PermissionsToManagePCSNetworkInterfaces",
      "Effect" : "Allow",
      "Action" : [
        "ec2:DeleteNetworkInterface",
        "ec2:CreateNetworkInterfacePermission"
      ],
      "Resource" : "arn:aws:ec2:*:*:network-interface/*",
      "Condition" : {
        "Null" : {
          "aws:ResourceTag/AWSPCSManaged" : "false"
        }
      }
    },
    {
      "Sid" : "PermissionsToDescribePCSResources",
      "Effect" : "Allow",
      "Action" : [
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeNetworkInterfaces",
```

```
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceTypes",
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstanceAttribute",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeImages",
        "ec2:DescribeImageAttribute"
      ],
      "Resource" : "*"
    },
    {
      "Sid" : "PermissionsToCreatePCSLaunchTemplates",
      "Effect" : "Allow",
      "Action" : [
        "ec2:CreateLaunchTemplate"
      ],
      "Resource" : "arn:aws:ec2:*:*:launch-template/*",
      "Condition" : {
        "Null" : {
          "aws:RequestTag/AWSPCSManaged" : "false"
        }
      }
    },
    {
      "Sid" : "PermissionsToManagePCSLaunchTemplates",
      "Effect" : "Allow",
      "Action" : [
        "ec2:DeleteLaunchTemplate",
        "ec2:DeleteLaunchTemplateVersions",
        "ec2:CreateLaunchTemplateVersion"
      ],
      "Resource" : "arn:aws:ec2:*:*:launch-template/*",
      "Condition" : {
        "Null" : {
          "aws:ResourceTag/AWSPCSManaged" : "false"
        }
      }
    },
    {
      "Sid" : "PermissionsToTerminatePCSManagedInstances",
      "Effect" : "Allow",
```

```
      "Action" : [
        "ec2:TerminateInstances"
      ],
      "Resource" : "arn:aws:ec2:*:*:instance/*",
      "Condition" : {
        "Null" : {
          "aws:ResourceTag/AWSPCSManaged" : "false"
        }
      }
    },
    {
      "Sid" : "PermissionsToPassRoleToEC2",
      "Effect" : "Allow",
      "Action" : "iam:PassRole",
      "Resource" : [
        "arn:aws:iam::*:role/*/AWSPCS*",
        "arn:aws:iam::*:role/AWSPCS*",
        "arn:aws:iam::*:role/aws-pcs/*",
        "arn:aws:iam::*:role/*/aws-pcs/*"
      ],
      "Condition" : {
        "StringEquals" : {
          "iam:PassedToService" : [
            "ec2.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid" : "PermissionsToControlClusterInstanceAttributes",
      "Effect" : "Allow",
      "Action" : [
        "ec2:RunInstances",
        "ec2:CreateFleet"
      ],
      "Resource" : [
        "arn:aws:ec2:*::image/*",
        "arn:aws:ec2:*::snapshot/*",
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:security-group/*",
        "arn:aws:ec2:*:*:volume/*",
        "arn:aws:ec2:*:*:key-pair/*",
        "arn:aws:ec2:*:*:launch-template/*",
```

```
        "arn:aws:ec2:*:*:placement-group/*",
        "arn:aws:ec2:*:*:capacity-reservation/*",
        "arn:aws:resource-groups:*:*:group/*",
        "arn:aws:ec2:*:*:fleet/*",
        "arn:aws:ec2:*:*:spot-instances-request/*"
      ]
    },
    {
      "Sid" : "PermissionsToProvisionClusterInstances",
      "Effect" : "Allow",
      "Action" : [
        "ec2:RunInstances",
        "ec2:CreateFleet"
      ],
      "Resource" : [
        "arn:aws:ec2:*:*:instance/*"
      ],
      "Condition" : {
        "Null" : {
          "aws:RequestTag/AWSPCSManaged" : "false"
        }
      }
    },
    {
      "Sid" : "PermissionsToTagPCSResources",
      "Effect" : "Allow",
      "Action" : [
        "ec2:CreateTags"
      ],
      "Resource" : [
        "*"
      ],
      "Condition" : {
        "StringEquals" : {
          "ec2:CreateAction" : [
            "RunInstances",
            "CreateLaunchTemplate",
            "CreateFleet",
            "CreateNetworkInterface"
          ]
        }
      }
    },
    {
```

```
      "Sid" : "PermissionsToPublishMetrics",
      "Effect" : "Allow",
      "Action" : "cloudwatch:PutMetricData",
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudwatch:namespace" : "AWS/PCS"
        }
      }
    },
    {
      "Sid" : "PermissionsToManageSecret",
      "Effect" : "Allow",
      "Action" : [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage",
        "secretsmanager:DeleteSecret"
      ],
      "Resource" : "arn:aws:secretsmanager:*:*:secret:pcs!*",
      "Condition" : {
        "StringEquals" : {
          "secretsmanager:ResourceTag/aws:secretsmanager:owningService" : "pcs",
          "aws:ResourceAccount" : "${aws:PrincipalAccount}"
        }
      }
    }
  ]
}
```

AWS PCS updates to AWS managed policies

View details about updates to AWS managed policies for AWS PCS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS PCS Document history page.

Change	Description	Date
Updated the JSON in this document	Corrected the JSON in this document to include `"arn:aws:ec2:*:*:s`	September 5, 2024

Change	Description	Date
	pot-instances-requ est/*" .	
AWS PCS started tracking changes	AWS PCS started tracking changes for its AWS managed policies.	August 28, 2024

Service-linked roles for AWS PCS

AWS Parallel Computing Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS PCS. Service-linked roles are predefined by AWS PCS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS PCS easier because you don't have to manually add the necessary permissions. AWS PCS defines the permissions of its service-linked roles, and unless defined otherwise, only AWS PCS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting its related resources. This protects your AWS PCS resources because you can't accidentally remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS PCS

AWS PCS uses the service-linked role named **AWSServiceRoleForPCS** – Allow AWS PCS to manage Amazon EC2 resources.

The AWSServiceRoleForPCS service-linked role trusts the following services to assume the role:

- `pcs.amazonaws.com`

The role permissions policy named AWSPCSServiceRolePolicy allows AWS PCS to complete actions on specific resources.

You must configure permissions to allow your users, groups, or roles to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for AWS PCS

You don't need to manually create a service-linked role. AWS PCS creates a service-linked role for you when you create a cluster.

Editing a service-linked role for AWS PCS

AWS PCS does not allow you to edit the AWSServiceRoleForPCS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS PCS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the AWS PCS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To remove AWS PCS resources used by the AWSServiceRoleForPCS

You must delete all your clusters to delete the AWSServiceRoleForPCS service-linked role. For more information, see Delete a cluster.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForPCS service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for AWS PCS service-linked roles

AWS PCS supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS Regions and endpoints.

Amazon EC2 Spot role for AWS PCS

If you want to create an AWS PCS compute node group that uses **Spot** as its purchase option, you must also have the **AWSServiceRoleForEC2Spot** service-linked role in your AWS account. You can use the following AWS CLI command to create the role. For more information, see Create a service-linked role and Create a role to delegate permissions to an AWS service in the *AWS Identity and Access Management User Guide*.

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

> **Note**
>
> You receive the following error if your AWS account already has an `AWSServiceRoleForEC2Spot` IAM role.
>
> ```
>  An error occurred (InvalidInput) when calling the CreateServiceLinkedRole
> operation: Service role name AWSServiceRoleForEC2Spot has been taken in this
> account, please try a different suffix.
> ```

Minimum permissions for AWS PCS

This section describes the minimum IAM permissions required for an IAM identity (user, group, or role) to use the service.

Contents

Minimum permissions to use API actions

API action	Minimum permissions	Additional permissions for the console
CreateCluster	`ec2:CreateNetworkInterface, ec2:DescribeVpcs, ec2:DescribeSubnets, ec2:DescribeSecurityGroups, ec2:GetSecurityGroupsForVpc, iam:CreateServiceLinkedRole, secretsmanager:CreateSecret, secretsmanager:TagResource, pcs:CreateCluster`	
ListClusters	`pcs:ListClusters`	
GetCluster	`pcs:GetCluster`	`ec2:DescribeSubnets`
DeleteCluster	`pcs:DeleteCluster`	
CreateComputeNodeGroup	`ec2:DescribeVpcs, ec2:DescribeSubnets, ec2:DescribeSecurityGroups, ec2:DescribeLaunchTemplates, ec2:DescribeLaunchTemplateVersions, ec2:DescribeInstanceTypes, ec2:DescribeInstanceTypeOfferings,`	`iam:ListInstanceProfiles, ec2:DescribeImages, pcs:GetCluster`

API action	Minimum permissions	Additional permissions for the console
	ec2:RunInstances, ec2:CreateFleet, ec2:CreateTags, iam:PassRole, iam:GetInstanceProfile, pcs:CreateComputeNodeGroup	
ListComputerNodeGroups	pcs:ListComputeNodeGroups	pcs:GetCluster
GetComputeNodeGroup	pcs:GetComputeNodeGroup	ec2:DescribeSubnets
UpdateComputeNodeGroup	ec2:DescribeVpcs, ec2:DescribeSubnets, ec2:DescribeSecurityGroups, ec2:DescribeLaunchTemplates, ec2:DescribeLaunchTemplateVersions, ec2:DescribeInstanceTypes, ec2:DescribeInstanceTypeOfferings, ec2:RunInstances, ec2:CreateFleet, ec2:CreateTags, iam:PassRole, iam:GetInstanceProfile, pcs:UpdateComputeNodeGroup	pcs:GetComputeNodeGroup, iam:ListInstanceProfiles, ec2:DescribeImages, pcs:GetCluster

API action	Minimum permissions	Additional permissions for the console
DeleteComputeNodeGroup	`pcs:DeleteComputeNodeGroup`	
CreateQueue	`pcs:CreateQueue`	`pcs:ListComputeNodeGroups, pcs:GetCluster`
ListQueues	`pcs:ListQueues`	`pcs:GetCluster`
GetQueue	`pcs:GetQueue`	
UpdateQueue	`pcs:UpdateQueue`	`pcs:ListComputeNodeGroups, pcs:GetQueue`
DeleteQueue	`pcs:DeleteQueue`	

Minimum permissions to use tags

The following permissions are required to use tags with your resources in AWS PCS.

```
pcs:ListTagsForResource,
pcs:TagResource,
pcs:UntagResource
```

Minimum permissions to support logs

AWS PCS sends log data to Amazon CloudWatch Logs (CloudWatch Logs). You must make sure that your identiy has the minimum permissions to use CloudWatch Logs. For more information, see Overview of managing access permissions to your CloudWatch Logs resources in the *Amazon CloudWatch Logs User Guide*.

For information about permissions required for a service to send logs to CloudWatch Logs, see Enabling logging from AWS services in the *Amazon CloudWatch Logs User Guide*.

Minimum permissions for a service administrator

The following IAM policy specifies the minimum permissions required for an IAM identity (user, group, or role) to configure and manage the AWS PCS service.

> **Note**
>
> Users who don't configure and manage the service don't require these permissions. Users who only run jobs use secure shell (SSH) to connect to the cluster. AWS Identity and Access Management (IAM) doesn't handle authentication or authorization for SSH.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PCSAccess",
      "Effect": "Allow",
      "Action": [
        "pcs:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EC2Access",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeImages",
        "ec2:GetSecurityGroupsForVpc",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:DescribeInstanceTypes",
        "ec2:DescribeInstanceTypeOfferings",
        "ec2:RunInstances",
        "ec2:CreateFleet",
```

```
        "ec2:CreateTags"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IamInstanceProfile",
      "Effect": "Allow",
      "Action": [
        "iam:GetInstanceProfile"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IamPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/*/AWSPCS*",
        "arn:aws:iam::*:role/AWSPCS*",
        "arn:aws:iam::*:role/aws-pcs/*",
        "arn:aws:iam::*:role/*/aws-pcs/*"
      ],
      "Condition": {
        "StringEquals": {
           "iam:PassedToService": [
             "ec2.amazonaws.com"
           ]
        }
      }
    },
    {
      "Sid": "SLRAccess",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/pcs.amazonaws.com/AWSServiceRoleFor*",
        "arn:aws:iam::*:role/aws-service-role/spot.amazonaws.com/AWSServiceRoleFor*"
      ],
      "Condition": {
        "StringLike": {
```

```
          "iam:AWSServiceName": [
            "pcs.amazonaws.com",
            "spot.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "AccessKMSKey",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:GenerateDataKey",
        "kms:CreateGrant",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SecretManagementAccess",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:TagResource",
        "secretsmanager:UpdateSecret"
      ],
      "Resource": "*"
    },
    {
       "Sid": "ServiceLogsDelivery",
       "Effect": "Allow",
       "Action": [
         "pcs:AllowVendedLogDeliveryForResource",
         "logs:PutDeliverySource",
         "logs:PutDeliveryDestination",
         "logs:CreateDelivery"
       ],
       "Resource": "*"
    }
  ]
}
```

IAM instance profiles for AWS Parallel Computing Service

Applications that run on an EC2 instance must include AWS credentials in any AWS API requests they make. We recommended you use an IAM role to manage temporary credentials on the EC2 instance. You can define an instance profile to do this, and attach it to your instances. For more information, see IAM roles for Amazon EC2 in the *Amazon Elastic Compute Cloud User Guide*.

Note

When you use the AWS Management Console to create an IAM role for Amazon EC2, the console creates an instance profile automatically and gives it the same name as the IAM role. If you use the AWS CLI, AWS API actions, or an AWS SDK to create the IAM role, you create the instance profile as a separate action. For more information, see Instance profiles in the *Amazon Elastic Compute Cloud User Guide*.

You must specify the Amazon Resource Name (ARN) of an instance profile when you create a compute node groups. You can choose different instance profiles for some or all compute node groups.

Instance Profile Requirements

Instance Profile ARN

The IAM role name part of the ARN must either begin with `AWSPCS` or contain `/aws-pcs/` in its path:

- `arn:aws:iam::*:instance-profile/AWSPCS-example-role-1` and
- `arn:aws:iam::*:instance-profile/aws-pcs/example-role-2`.

Note

If you use the AWS CLI, provide a `--path` value to `iam create-instance-profile` to include `/aws-pcs/` in the ARN path. For example:

```
aws iam create-instance-profile --path /aws-pcs/ --instance-profile-name
 example-role-2
```

Permissions

At minimum, the instance profile for AWS PCS must include the following policy. It allows compute nodes to notify the AWS PCS service when they become operational.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "pcs:RegisterComputeNodeGroupInstance"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Additional policies

You may consider adding managed policies to the instance profile. For example:

- AmazonS3ReadOnlyAccess provides read-only access to all S3 buckets.
- AmazonSSMManagedInstanceCore enables AWS Systems Manager service core functionality, such as remote access directly from the Amazon Management Console.
- CloudWatchAgentServerPolicy contains permissions required to use AmazonCloudWatchAgent on servers.

You can also include your own IAM policies that support your specific use case.

Creating an instance profile

You can create an instance profile directly from the Amazon EC2 console. For more information, see Using instance profiles in the *AWS Identity and Access Management User Guide*.

Troubleshooting AWS Parallel Computing Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS PCS and IAM.

Topics

I am not authorized to perform an action in AWS PCS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `pcs:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 pcs:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `pcs:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS PCS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS PCS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS PCS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS PCS supports these features, see How AWS Parallel Computing Service works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS Parallel Computing Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS Parallel Computing Service

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure Security in AWS Parallel Computing Service

As a managed service, AWS Parallel Computing Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access AWS PCS through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

When AWS PCS creates a cluster, the service launches the Slurm controller in a service-owned account, separate from the compute nodes in your account. To bridge communication between the controller and the compute nodes, AWS PCS creates a cross-account Elastic Network Interface (ENI) in your VPC. The Slurm controller uses the ENI to manage and communicate with the compute nodes across different AWS accounts, maintaining the security and isolation of resources while facilitating efficient HPC and AI/ML operations.

Vulnerability analysis and management in AWS Parallel Computing Service

Configuration and IT controls are a shared responsibility between AWS and you. For more information, see the AWS shared responsibility model. AWS handles basic security tasks for the underlying infrastructure in the service account, such as patching the operating system on controller instances, firewall configuration, and AWS infrastructure disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see Best Practices for Security, Identity, and Compliance.

> **Note**
>
> Slurm controllers are unavailable while we update them. Running jobs aren't affected. Jobs submitted when the cluster's controller is unavailable are held until the controller is available.

You are responsible for the security of the underlying infrastructure in your AWS account:

- Maintain your code, including updates and security patches.
- Patch and update the operating system in the Amazon Machine Image (AMI) for your compute node groups and update your compute node groups to use the updated AMI.
- Update the scheduler to keep it within supported versions. Update the AMI for your compute node groups and update your compute node group to use the updated AMI.
- Authenticate and encrypt communication between user clients and the nodes they connect to.

For more information about updating the AMI for your compute node groups, see Amazon Machine Images (AMIs) for AWS PCS.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Parallel Computing Service (AWS PCS) gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:servicename:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

The value of `aws:SourceArn` must be a cluster ARN.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in AWS PCS to prevent the confused deputy problem.

```
{
"Version": "2012-10-17",
  "Statement": {
"Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "pcs.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": [
          "arn:aws:pcs:us-east-1:123456789012:cluster/*"
        ]
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

IAM role for Amazon EC2 instances provisioned as part of a compute node group

AWS PCS automatically orchestrates Amazon EC2 capacity for each of the configured compute node groups in a cluster. When creating a compute node group, users must provide an IAM instance profile through the `iamInstanceProfileArn` field. The instance profile specifies the permissions associated with the provisioned EC2 instances. AWS PCS accepts any role that has `AWSPCS` as role name prefix or `/aws-pcs/` as part of the role path. The `iam:PassRole` permission is required on the IAM identity (user or role) that creates or updates a compute node group. When a user calls the `CreateComputeNodeGroup` or `UpdateComputeNodeGroup` API actions, AWS PCS checks to see if the user is allowed to perform the `iam:PassRole` action.

The following example policy grants permissions to pass only IAM roles whose name begins with `AWSPCS`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/AWSPCS*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Security best practices for AWS Parallel Computing Service

This section describes security best practices that are specific to AWS Parallel Computing Service (AWS PCS). To learn more about security best practices in AWS, see [Best Practices for Security, Identity, and Compliance](#).

AMI-related security

- Don’t use AWS PCS sample AMIs for production workloads. The sample AMIs are unsupported and only intended for testing.
- Regularly update the operating system and software in the AMI for your compute node groups to mitigate vulnerabilities.
- Only use authenticated official AWS PCS packages downloaded from official AWS sources.
- Regularly update AWS PCS packages in the AMI for compute node groups and update the compute nodes to use the updated AMI. Consider automating this process to minimize vulnerabilities.

For more information, see Custom Amazon Machine Images (AMIs) for AWS PCS.

Slurm Workload Manager security

- Implement access controls and network restrictions to secure Slurm control and compute nodes. Only allow trusted users and systems to submit jobs and access Slurm management commands.
- Use Slurm's built-in security features, such as Slurm authentication, to ensure that job submissions and communications are authenticated.
- Update Slurm versions to maintain smooth operations and cluster support.

> **Important**
>
> Any cluster that uses a version of Slurm that has reached *end of support life* (EOSL) is stopped immediately. Use the link at the top of the user guide pages to subscribe to the AWS PCS documentation RSS feed to receive notification when a Slurm version approaches EOSL.

For more information, see Slurm versions in AWS PCS.

Monitoring and logging

- Use Amazon CloudWatch Logs and AWS CloudTrail to monitor and record actions in your clusters and AWS account. Use the data for troubleshooting and auditing.

Network security

- Deploy your AWS PCS clusters in a separate VPC to isolate your HPC environment from other network traffic.
- Use security groups and network access control lists (ACLs) to control inbound and outbound traffic to AWS PCS instances and subnets.
- Use AWS PrivateLink or VPC endpoints to keep network traffic to between your clusters and other AWS services inside the AWS network. For more information, see Access AWS Parallel Computing Service using an interface endpoint (AWS PrivateLink).

Logging and monitoring for AWS PCS

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS PCS and your other AWS resources. AWS provides the following monitoring tools to watch AWS PCS, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

AWS PCS scheduler logs

You can configure AWS PCS to send detailed logging data from your cluster scheduler to Amazon CloudWatch Logs, Amazon Simple Storage Service (Amazon S3), and Amazon Data Firehose. This can assist with monitoring and troubleshooting. You can set up AWS PCS scheduler logs using the AWS PCS console, as well as programmatically using the AWS CLI or SDK.

Contents

    - Connect the cluster delivery source to the delivery destination
- Scheduler log stream paths and names
- Example AWS PCS scheduler log record

Prerequisites

The IAM principal used to manage the AWS PCS cluster must allow `pcs:AllowVendedLogDeliveryForResource`. Here is a sample AWS IAM policy that enables it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PcsAllowVendedLogsDelivery",
            "Effect": "Allow",
            "Action": ["pcs:AllowVendedLogDeliveryForResource"],
            "Resource": [
                "arn:aws:pcs:::cluster/*"
            ]
        }
    ]
}
```

Setting up scheduler logs using the AWS PCS console

To set up AWS PCS scheduler logs in the console, follow these steps:

1. Open the AWS PCS console.
2. Choose **Clusters** and navigate to the detail page for the AWS PCS cluster where you will enable logging.
3. Choose **Logs**.
4. Under **log deliveries – Scheduler Logs** – *optional*
    a. Add up to three log delivery destinations. Choices include CloudWatch Logs, Amazon S3, or Firehose.
    b. Choose **Update log deliveries**.

You can reconfigure, add, or remove log deliveries by revisiting this page.

Setting up scheduler logs using the AWS CLI

To accomplish this, you need at least one delivery destination, one delivery source (the PCS cluster), and one delivery, which is a relationship that connects a source to a destination.

Create a delivery destination

You need at least one delivery destination to receive scheduler logs from an AWS PCS cluster. You can learn more about this topic in the PutDeliveryDestination section of the CloudWatch API User Guide.

To create a delivery destination using the AWS CLI

- Create a destination with the command that follows. Before running the command, make the following replacements:
  - Replace *region-code* with the AWS Region where you will create your destination. This will generally be the same region as where the AWS PCS cluster is deployed.
  - Replace *pcs-logs-destination* with your preferred name. It must be unique for all delivery destinations in your account.
  - Replace *resource-arn* with the ARN for an existing log group in CloudWatch Logs, an S3 bucket, or a delivery stream in Firehose. Examples include:
    - **CloudWatch Logs group**

      ```
      arn:aws:logs:region-code:account-id:log-group:/log-group-name:*
      ```

    - **S3 bucket**

      ```
      arn:aws:s3:::bucket-name
      ```

    - **Firehose delivery stream**

      ```
      arn:aws:firehose:region-code:account-id:deliverystream/stream-name
      ```

```
aws logs put-delivery-destination --region region-code \
    --name pcs-logs-destination \
    --delivery-destination-configuration destinationResourceArn=resource-arn
```

Take note of the ARN for the new delivery destination, since you will need it to configure deliveries.

Enable the AWS PCS cluster as a delivery source

To collect scheduler logs from AWS PCS, configure the cluster as a delivery source. For more information, see PutDeliverySource in the *Amazon CloudWatch Logs API Reference*.

To configure a cluster as a delivery source using the AWS CLI

- Enable logs delivery from your cluster with the command that follows. Before running the command, make the following replacements:
  - Replace *region-code* with the AWS Region where your cluster is deployed.
  - Replace *cluster-logs-source-name* with a name for this source. It must be unique for all delivery sources in your AWS account. Consider incorporating the name or ID of the AWS PCS cluster.
  - Replace *cluster-arn* with the ARN for your AWS PCS cluster

```
aws logs put-delivery-source \
    --region region-code \
    --name cluster-logs-source-name \
    --resource-arn cluster-arn \
    --log-type PCS_SCHEDULER_LOGS
```

Connect the cluster delivery source to the delivery destination

For scheduler log data to flow from the cluster to the destination, you must configure a delivery that connects them. For more information, see CreateDelivery in the *Amazon CloudWatch Logs API Reference*.

To create a delivery using the AWS CLI

- Create a delivery using the command that follows. Before running the command, make the following replacements:
  - Replace *region-code* with the AWS Region where your source and destination exist.
  - Replace *cluster-logs-source-name* with the name of your delivery source from above.
  - Replace *destination-arn* with the ARN from a delivery destination where you want logs to be delivered.

```
aws logs create-delivery \
```

```
    --region region-code \
    --delivery-source-name cluster-logs-source \
    --delivery-destination-arn destination-arn
```

Scheduler log stream paths and names

The path and name for AWS PCS scheduler logs depend on the destination type.

- **CloudWatch Logs**
  - A CloudWatch Logs stream follows this naming convention.

    ```
    AWSLogs/PCS/${cluster_id}/${log_name}_${scheduler_major_version}.log
    ```

    Example

    ```
    AWSLogs/PCS/abcdef0123/slurmctld_24.05.log
    ```
- **S3 bucket**
  - An S3 bucket output path follows this naming convention:

    ```
    AWSLogs/${account-id}/PCS/${region}/${cluster_id}/${log_name}/
    ${scheduler_major_version}/yyyy/MM/dd/HH/
    ```

    Example

    ```
    AWSLogs/111111111111/PCS/us-east-2/abcdef0123/slurmctld/24.05/2024/09/01/00.
    ```
  - An S3 object name follows this convention:

    ```
    PCS_${log_name}_${scheduler_major_version}_#{expr date 'event_timestamp', format:
     "yyyy-MM-dd-HH"}_${cluster_id}_${hash}.log
    ```

    Example

    ```
    PCS_slurmctld_24.05_2024-09-01-00_abcdef0123_0123abcdef.log
    ```

Example AWS PCS scheduler log record

AWS PCS scheduler logs are structured. They include fields such as the cluster identifier, scheduler type, major and patch versions, in addition to the log message emitted from the Slurm controller process. Here is an example.

```
{
    "resource_id": "s3431v9rx2",
    "resource_type": "PCS_CLUSTER",
    "event_timestamp": 1721230979,
    "log_level": "info",
    "log_name": "slurmctld",
    "scheduler_type": "slurm",
    "scheduler_major_version": "23.11",
    "scheduler_patch_version": "8",
    "node_type": "controller_primary",
    "message": "[2024-07-17T15:42:58.614+00:00] Running as primary controller\n"
}
```

Monitoring AWS Parallel Computing Service with Amazon CloudWatch

Amazon CloudWatch provides monitoring of your AWS Parallel Computing Service (AWS PCS) cluster health and performance by collecting metrics from the cluster at intervals. These metrics are retained, allowing you to access historical data and gain insights into your cluster's performance over time.

CloudWatch also enables you to monitor the EC2 instances launched by AWS PCS to meet your scaling requirements. While you can inspect logs on running instances, CloudWatch metrics and logging data are typically deleted once instances are terminated. However, you can configure the CloudWatch agent on instances using an EC2 launch template to persist metrics and logs even after instance termination, enabling long-term monitoring and analysis.

Explore the topics in this section to learn more about monitoring AWS PCS using CloudWatch.

Topics

Monitoring AWS PCS metrics using CloudWatch

You can monitor AWS PCS cluster health using Amazon CloudWatch, which collects data from your cluster and turns it into near real-time metrics. These statistics are retained for a period of 15 months, so that you can access historical information and gain a better perspective on how your cluster is performing. Cluster metrics are sent to CloudWatch at 1-minute periods. For more information about CloudWatch, see [What Is Amazon CloudWatch?](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html) in the *Amazon CloudWatch User Guide*.

AWS PCS publishes the following metrics into the **AWS/PCS** namespace in CloudWatch. They have a single dimension, `ClusterId`.

Name	Description	Units
ActualCapacity	IdleCapacity + UtilizedCapacity	Count
CapacityUtilization	UtilizedCapacity / ActualCapacity	Count
DesiredCapacity	ActualCapacity + PendingCapacity	Count
IdleCapacity	Count of instances that are running but not allocated to jobs	Count
UtilizedCapacity	Count of instances that are running and allocated to jobs	Count

Monitoring AWS PCS instances using Amazon CloudWatch

AWS PCS launches Amazon EC2 instances as needed to meet the scaling requirements defined in your PCS compute node groups. You can monitor these instances while they are running using Amazon CloudWatch. You can inspect the logs of running instances by logging into them and using interactive command line tools. However, by default, CloudWatch metrics data is only retained for a limited period once an instance is terminated, and instance logs are usually deleted along with the EBS volumes that back the instance. To retain metrics or logging data from the instances launched by PCS after they are terminated, you can configure the CloudWatch agent on your instances with an EC2 launch template. This topic provides an overview of monitoring running instances and provides examples of how to configure persistent instance metrics and logs.

Monitoring running instances

Finding AWS PCS instances

To monitor instances launched by PCS, find the running instances associated with a cluster or compute node group. Then, in the EC2 console for a given instance, inspect the **Status and alarms** and **Monitoring** sections. If login access is configured for those instances, you can connect to them and inspect various log files on the instances. For more information on identifying which instances are managed by PCS, see Finding compute node group instances in AWS PCS.

Enabling detailed metrics

By default, instance metrics are collected at 5-minute intervals. To collect metrics at one minute intervals, enable detailed CloudWatch monitoring in your compute node group launch template. For more information, see Turn on detailed CloudWatch monitoring.

Configuring persistent instance metrics and logs

You can retain the metrics and logs from your instances by installing and configuring the Amazon CloudWatch agent on them. This consists of three main steps:

1. Create a CloudWatch agent configuration.
2. Store the configuration where it can be retrieved by PCS instances.
3. Write an EC2 launch template that installs the CloudWatch agent software, fetches your configuration, and starts the CloudWatch agent using the configuration.

For more information, see Collect metrics, logs, and traces with the CloudWatch agent in the *Amazon CloudWatch User Guide*, and Using Amazon EC2 launch templates with AWS PCS.

Create a CloudWatch Agent configuration

Before deploying the CloudWatch agent on your instances, you must generate a JSON configuration file that specifies the metrics, logs, and traces to collect. Configuration files can be created using a wizard or manually, using a text editor. The configuration file will be created manually for this demonstration.

On a computer where you have the AWS CLI installed, create a CloudWatch configuration file named **config.json** with the contents that follow. You can also use the following URL to download a copy of the file.

```
https://aws-hpc-recipes.s3.amazonaws.com/main/recipes/pcs/cloudwatch/assets/config.json
```

Notes

- The log paths in the sample file are for Amazon Linux 2. If your instances will use a different base operating system, change the paths as appropriate.
- To capture other logs, add additional entries under `collect_list`.
- Values in `{brackets}` are templated variables. For the complete list of supported variables, see Manually create or edit the CloudWatch agent configuration file in the *Amazon CloudWatch User Guide*.
- You can choose to omit `logs` or `metrics` if you don't want to collect these information types.

```
{
    "agent": {
        "metrics_collection_interval": 60
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/var/log/cloud-init.log",
                        "log_group_class": "STANDARD",
                        "log_group_name": "/PCSLogs/instances",
                        "log_stream_name": "{instance_id}.cloud-init.log",
                        "retention_in_days": 30
                    },
                    {
                        "file_path": "/var/log/cloud-init-output.log",
                        "log_group_class": "STANDARD",
                        "log_stream_name": "{instance_id}.cloud-init-output.log",
                        "log_group_name": "/PCSLogs/instances",
                        "retention_in_days": 30
                    },
                    {
                        "file_path": "/var/log/amazon/pcs/bootstrap.log",
```

```
                        "log_group_class": "STANDARD",
                        "log_stream_name": "{instance_id}.bootstrap.log",
                        "log_group_name": "/PCSLogs/instances",
                        "retention_in_days": 30
                    },
                    {
                        "file_path": "/var/log/slurmd.log",
                        "log_group_class": "STANDARD",
                        "log_stream_name": "{instance_id}.slurmd.log",
                        "log_group_name": "/PCSLogs/instances",
                        "retention_in_days": 30
                    },
                    {
                        "file_path": "/var/log/messages",
                        "log_group_class": "STANDARD",
                        "log_stream_name": "{instance_id}.messages",
                        "log_group_name": "/PCSLogs/instances",
                        "retention_in_days": 30
                    },
                    {
                        "file_path": "/var/log/secure",
                        "log_group_class": "STANDARD",
                        "log_stream_name": "{instance_id}.secure",
                        "log_group_name": "/PCSLogs/instances",
                        "retention_in_days": 30
                    }
                ]
            }
        }
    },
    "metrics": {
        "aggregation_dimensions": [
            [
                "InstanceId"
            ]
        ],
        "append_dimensions": {
            "AutoScalingGroupName": "${aws:AutoScalingGroupName}",
            "ImageId": "${aws:ImageId}",
            "InstanceId": "${aws:InstanceId}",
            "InstanceType": "${aws:InstanceType}"
        },
        "metrics_collected": {
            "cpu": {
```

```
                "measurement": [
                    "cpu_usage_idle",
                    "cpu_usage_iowait",
                    "cpu_usage_user",
                    "cpu_usage_system"
                ],
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ],
                "totalcpu": false
            },
            "disk": {
                "measurement": [
                    "used_percent",
                    "inodes_free"
                ],
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ]
            },
            "diskio": {
                "measurement": [
                    "io_time"
                ],
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ]
            },
            "mem": {
                "measurement": [
                    "mem_used_percent"
                ],
                "metrics_collection_interval": 60
            },
            "swap": {
                "measurement": [
                    "swap_used_percent"
                ],
                "metrics_collection_interval": 60
            }
        }
```

```
    }
}
```

This file instructs the CloudWatch agent to monitor several files that can be helpful in diagnosing errors in instance bootstrapping, authentication and login, and other troubleshooting domains. These include:

- `/var/log/cloud-init.log` – Output from the initial stage of instance configuration
- `/var/log/cloud-init-output.log` – Output from commands that run during instance configuration
- `/var/log/amazon/pcs/bootstrap.log` – Output from PCS-specific operations that run during instance configuration
- `/var/log/slurmd.log` – Output from the Slurm workload manager's daemon slurmd
- `/var/log/messages` – System messages from the kernel, system services, and applications
- `/var/log/secure` – Logs related to authentication attempts, such as SSH, sudo, and other security events

The log files are sent to a CloudWatch log group named `/PCSLogs/instances`. The log streams are a combination of the instance ID and the base name of the log file. The log group has a retention time of 30 days.

In addition, the file instructs CloudWatch agent to collect several common metrics, aggregating them by instance ID.

Store the configuration

The CloudWatch agent configuration file has to be stored where it can be accessed by PCS compute node instances. There are two common ways to do this. You can upload it to an Amazon S3 bucket that your compute node group instances will have access to via their instance profile, Alternatively, you can store it as an SSM parameter in Amazon Systems Manager Parameter Store.

Upload to an S3 bucket

To store your file in S3, use the AWS CLI commands that follow. Before running the command, make these replacements:

- Replace *`amzn-s3-demo-bucket`* with your own S3 bucket name

First, (this is optional if you have an existing bucket), create a bucket to hold your configuration file(s).

```
aws s3 mb s3://amzn-s3-demo-bucket
```

Next, upload the file to the bucket.

```
aws s3 cp ./config.json s3://amzn-s3-demo-bucket/
```

Store as an SSM parameter

To store your file as an SSM parameter, use the command that follows. Before running the command, make these replacements:

- Replace *region-code* with the AWS Region where you are working with AWS PCS.
- (Optional) Replace *AmazonCloudWatch-PCS* with your own name for the parameter. Note that if you change the prefix of the name from `AmazonCloudWatch-` you will need to specifically add read access to the SSM parameter in your node group instance profile.

```
aws ssm put-parameter \
   --region region-code \
   --name "AmazonCloudWatch-PCS" \
   --type String \
   --value file://config.json
```

Write an EC2 launch template

The specific details for the launch template depend on whether your configuration file is stored in S3 or SSM.

Use a configuration stored in S3

This script installs CloudWatch agent, imports a configuration file from an S3 bucket, and launches the CloudWatch agent with it. Replace the following values in this script with your own details:

- *amzn-s3-demo-bucket* – The name of an S3 bucket your account can read from
- */config.json* – Path relative to the S3 bucket root where the configuration is stored

```
MIME-Version: 1.0
```

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
- amazon-cloudwatch-agent

runcmd:
- aws s3 cp s3://amzn-s3-demo-bucket/config.json /etc/s3-cw-config.json
- /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m
 ec2 -s -c file://etc/s3-cw-config.json

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

The IAM instance profile for the node group must have access to the bucket. Here is an example IAM policy for the bucket in the user data script above.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

Also note that the instances must allow outbound traffic to the S3 and CloudWatch endpoints. This can be accomplished using security groups or VPC endpoints, depending on your cluster architecture.

Use a configuration stored in SSM

This script installs CloudWatch agent, imports a configuration file from an SSM parameter, and launches the CloudWatch agent with it. Replace the following values in this script with your own details:

- (Optional) Replace *AmazonCloudWatch-PCS* with your own name for the parameter.

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

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
- amazon-cloudwatch-agent

runcmd:
- /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m
 ec2 -s -c ssm:AmazonCloudWatch-PCS

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

The IAM instance policy for the node group must have the **CloudWatchAgentServerPolicy** attached to it.

If your parameter name does not start with `AmazonCloudWatch-` you will need to specifically add read access to the SSM parameter in your node group instance profile. Here is an example IAM policy that illustrates this for prefix *DOC-EXAMPLE-PREFIX*.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Sid" : "CustomCwSsmMParamReadOnly",
      "Effect" : "Allow",
      "Action" : [
        "ssm:GetParameter"
      ],
      "Resource" : "arn:aws:ssm:*:*:parameter/DOC-EXAMPLE-PREFIX*"
    }
```

```
    ]
}
```

Also note that the instances must allow outbound traffic to the SSM and CloudWatch endpoints. This can be accomplished using security groups or VPC endpoints, depending on your cluster architecture.

Logging AWS Parallel Computing Service API calls using AWS CloudTrail

AWS PCS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS PCS. CloudTrail captures all API calls for AWS PCS as events. The calls captured include calls from the AWS PCS console and code calls to the AWS PCS API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS PCS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS PCS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS PCS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS PCS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for AWS PCS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail

- [Receiving CloudTrail log files from multiple regions] and [Receiving CloudTrail log files from multiple accounts]

All AWS PCS actions are logged by CloudTrail and are documented in the AWS Parallel Computing Service API Reference. For example, calls to the `CreateComputeNodeGroup`, `UpdateQueue`, and `DeleteCluster` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding CloudTrail log file entries from AWS PCS

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry for a `CreateQueue` action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
        "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
        "accountId": "012345678910",
        "accessKeyId": "ASIAY36PTPIEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAY36PTPIEEXAMPLE",
```

```
                "arn": "arn:aws:iam::012345678910:role/Admin",
                "accountId": "012345678910",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2024-07-16T17:05:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-16T17:13:09Z",
    "eventSource": "pcs.amazonaws.com",
    "eventName": "CreateQueue",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36",
    "requestParameters": {
        "clientToken": "c13b7baf-2894-42e8-acec-example",
        "clusterIdentifier": "abcdef0123",
        "computeNodeGroupConfigurations": [
            {
                "computeNodeGroupId": "abcdef0123"
            }
        ],
        "queueName": "all"
    },
    "responseElements": {
        "queue": {
            "arn": "arn:aws:pcs:us-east-1:609783872011:cluster/abcdef0123/queue/
abcdef0123",
            "clusterId": "abcdef0123",
            "computeNodeGroupConfigurations": [
                {
                    "computeNodeGroupId": "abcdef0123"
                }
            ],
            "createdAt": "2024-07-16T17:13:09.276069393Z",
            "id": "abcdef0123",
            "modifiedAt": "2024-07-16T17:13:09.276069393Z",
            "name": "all",
            "status": "CREATING"
        }
    },
```

```
    "requestID": "a9df46d7-3f6d-43a0-9e3f-example",
    "eventID": "7ab18f88-0040-47f5-8388-example",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "012345678910",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "pcs.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

Endpoints and service quotas for AWS PCS

The following sections describe the endpoints and service quotas for AWS Parallel Computing Service (AWS PCS). Service quotas, formerly referred to as *limits*, are the maximum number of service resources or operations for your AWS account.

Your AWS account has default quotas for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

For more information, see AWS service quotas in the *AWS General Reference*.

Contents

Service endpoints

Region name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	pcs.us-east-1.amazonaws.com	HTTPS
US East (Ohio)	us-east-2	pcs.us-east-2.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	pcs.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	pcs.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	pcs.ap-southeast-2.amazonaws.com	HTTPS

Region name	Region	Endpoint	Protocol
Asia Pacific (Tokyo)	ap-northeast-1	pcs.ap-northeast-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	pcs.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	pcs.eu-west-1.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	pcs.eu-north-1.amazonaws.com	HTTPS

Service quotas

Name	Default	Adjustable	Description
Clusters	5	Yes	The maximum number of clusters per AWS Region.

Note

The default values are the initial quotas set by AWS. These default values are separate from the actual applied quota values and maximum possible service quotas. For more information, see Terminology in Service Quotas in the *Service Quotas User Guide*.

These service quotas are listed under **AWS Parallel Computing Service (PCS)** in the AWS Management Console. To request a quota increase for values that are shown as adjustable, see Requesting a Quota Increase in the *Service Quotas User Guide*.

Important

Remember to check the current AWS Region setting in the AWS Management Console.

Internal quotas

The following quotas are internal and non-adjustable.

Name	Default	Adjustable	Description
Concurrent cluster creation	1	No	The maximum number of clusters in the `Creating` state per AWS Region.

Relevant quotas for other AWS services

AWS PCS uses other AWS services. Your service quotas for those services impact your use of AWS PCS.

Amazon EC2 service quotas that impact AWS PCS

- Spot instance requests
- Running on-demand instances
- Launch templates
- Launch template versions
- Amazon EC2 API requests

For more information, see Amazon EC2 service quotas in the *Amazon Elastic Compute Cloud User Guide*.

Troubleshooting problems in AWS Parallel Computing Service

The following topics provide guidance to troubleshoot some problems you might encounter in AWS PCS.

Topics

An EC2 instance in AWS PCS is terminated and replaced after reboot

Problem overview

After an EC2 instance in a compute node group is rebooted, AWS PCS automatically terminates and replaces the instance.

Why this happens

AWS PCS doesn't support instance reboots. If an EC2 instance is rebooted, AWS PCS considers the instance unhealthy and replaces it. If AWS PCS continuously terminates and replaces your instances, it might be because something reboots your instances after they launch. Some examples include reboots by automation on the EC2 instance (such as an automatic reboot after patching), automation external to the EC2 instance (such as a network management application), another AWS service (such as AWS Systems Manager), or a manual reboot by a person.

What to do

You can check your `slurmctld` or `slurmd` logs to see if your instance was rebooted. For more information, see AWS PCS scheduler logs and Monitoring AWS PCS instances using Amazon CloudWatch. The following example `slurmctld` log entry indicates that the instance rebooted:

Example

```
[2024-09-12T06:42:50.393+00:00] validate_node_specs: Node Login-1 unexpectedly rebooted
 boot_time=1726123354 last response=1726123285
```

Rebooting because of patching

A reboot is often required after you apply patches. Don't apply patches directly to an EC2 instance that is part of a AWS PCS compute node group. If you must patch your EC2 instances, you should apply your patches to an updated Amazon Machine Image (AMI) and update your compute node groups to use the updated AMI. New EC2 instances that AWS PCS launches for those compute node groups will use the updated (patched) AMI. For more information, see Custom Amazon Machine Images (AMIs) for AWS PCS.

Document history for the AWS PCS User Guide

The following table describes the important changes to the documentation for AWS PCS.

Date	Change	Documentation updates	API versions updated
December 18, 2024	Updated for Slurm 24.05	Updated the user guide for Slurm 24.05 support. For more information, see Software installers to build custom AMIs for AWS PCS and Release notes for AWS PCS sample AMIs.	N/A
December 18, 2024	Updated NVIDIA versions for Slurm 23.11 sample AMIs	Updated NVIDIA driver and CUDA versions in the Slurm 23.11 sample AMIs. For more information, see Release notes for AWS PCS sample AMIs.	N/A
December 17, 2024	Updated Slurm installer	Updated the AMI topic for Slurm installer 23.11.10-3. For more information, see Software installers to build custom AMIs for AWS PCS.	N/A
December 13, 2024	Updated PCS agent	Updated the AMI topic for AWS PCS agent 1.1.1-1. For more information, see Software installers to build custom AMIs for AWS PCS.	N/A

Date	Change	Documentation updates	API versions updated
December 6, 2024	Updated PCS agent and Slurm installer	Updated the AMI topic for AWS PCS agent 1.1.0-1 and Slurm installer 23.11.10-2. For more information, see Software installers to build custom AMIs for AWS PCS.	N/A
December 6, 2024	Added a topic about OS support	For more information, see Supported operating systems in AWS PCS.	N/A
November 8, 2024	Reorganized user guide	We reorganized the user guide to bring topics to the top level, moved some topics to their own pages, and grouped similar topics together.	N/A
November 7, 2024	Updated AMI topics	Updated the AMI topic for Slurm 23.11.10 and libjwt 17.0. For more information, see Software installers to build custom AMIs for AWS PCS and Step 3 – Install Slurm. Simplified and corrected the release notes for AMIs. For more informati on, see Release notes for AWS PCS sample AMIs.	N/A

Date	Change	Documentation updates	API versions updated
November 7, 2024	Added a new topic about using encrypted EBS volumes with AWS PCS	Added a topic that describes the KMS key policy required for encrypted EBS volumes in AWS PCS. For more information, see Required KMS key policy for use with encrypted EBS volumes in AWS PCS.	N/A
October 18, 2024	AWS PCS agent 1.0.1-1 released	Updated AMI-related documentation to refer to AWS PCS agent version 1.0.1-1. For more information, see Software installers to build custom AMIs for AWS PCS and Step 2 – Install the AWS PCS agent.	N/A
October 10, 2024	Added a troubleshooting chapter	Added a troubleshooting chapter with a topic about EC2 instances being automatically replaced after a reboot. For more information, see Troubleshooting problems in AWS Parallel Computing Service.	N/A

Date	Change	Documentation updates	API versions updated
September 23, 2024	Updated the minimum permissions to use API actions and for a service administrator	The `ec2:DescribeInstanceTypeOfferings` permission is now required for the `CreateComputeNodeGroup` and `UpdateComputeNodeGroup` API actions. For more information, see Minimum permissions for AWS PCS.	N/A
September 5, 2024	Updated the example IAM policy for the minimum permissions for a service administrator	For more information, see Minimum permissions for a service administrator.	N/A
September 5, 2024	Added a missing permission to the JSON in the managed policies page	This was a correction to the documentation only. The actual managed policy wasn't changed. For more information, see AWS managed policies for AWS Parallel Computing Service.	N/A
August 28, 2024	Managed policies page added	For more information, see AWS managed policies for AWS Parallel Computing Service.	N/A
August 28, 2024	AWS PCS release	Initial release of the AWS PCS user guide.	AWS SDK: 2024-08-28

AWS Glossary

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