



Red Hat Enterprise Linux 10

Deploying and managing RHEL on Amazon Web Services

Obtaining Red Hat Enterprise Linux system images and creating RHEL instances on
AWS

Red Hat Enterprise Linux 10 Deploying and managing RHEL on Amazon Web Services

Obtaining Red Hat Enterprise Linux system images and creating RHEL instances on AWS

Legal Notice

Copyright © Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

To use Red Hat Enterprise Linux (RHEL) on Amazon Web Services (AWS), you can create and deploy RHEL system images. This also involves: Registering, deploying, and provisioning RHEL images on AWS Managing networking configurations for RHEL AWS EC2 instances Configuring platform security and trusted execution technologies Managing Red Hat High Availability (HA) clusters for RHEL instances

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	4
CHAPTER 1. INTRODUCING RHEL ON PUBLIC CLOUD PLATFORMS	5
1.1. BENEFITS OF USING RHEL IN A PUBLIC CLOUD	5
1.2. PUBLIC CLOUD USE CASES FOR RHEL	6
1.3. FREQUENT CONCERNS WHEN MIGRATING TO A PUBLIC CLOUD	6
1.4. OBTAINING RHEL FOR PUBLIC CLOUD DEPLOYMENTS	7
1.5. METHODS FOR CREATING RHEL CLOUD INSTANCES	8
CHAPTER 2. PREPARING AND UPLOADING AMI IMAGES TO AWS	9
2.1. PREPARING TO MANUALLY UPLOAD AWS AMI IMAGES	9
2.2. MANUALLY UPLOADING AN AMI IMAGE TO AWS BY USING THE CLI	10
2.3. CREATING AND AUTOMATICALLY UPLOADING IMAGES TO THE AWS CLOUD AMI	11
CHAPTER 3. DEPLOYING A RHEL IMAGE AS AN EC2 INSTANCE ON AWS	14
3.1. AVAILABLE RHEL IMAGE TYPES FOR PUBLIC CLOUD	14
3.2. DEPLOYING A RHEL INSTANCE BY USING A CUSTOM BASE IMAGE	15
3.3. UPLOADING A RHEL IMAGE TO AWS BY USING THE COMMAND LINE	17
3.3.1. Installing AWSCLI2	17
3.3.2. Converting and pushing an image to Amazon S3	18
3.3.3. Managing a RHEL VM on AWS by using the command line	18
3.3.4. Attaching Red Hat subscriptions	19
3.3.5. Setting up automatic registration on AWS gold images	20
3.4. UPLOADING A RHEL IMAGE TO AWS BY USING THE AWS CONSOLE	20
3.4.1. Converting and pushing an image to S3 by using the AWS console	21
3.4.2. Managing a RHEL VM on AWS by using the AWS console	21
CHAPTER 4. CONFIGURING A RED HAT HIGH AVAILABILITY CLUSTER ON AWS	23
4.1. BENEFITS OF USING HIGH-AVAILABILITY CLUSTERS ON PUBLIC CLOUD PLATFORMS	23
4.2. INSTALLING THE HIGH AVAILABILITY PACKAGES AND AGENTS	24
4.3. CREATING A HIGH AVAILABILITY CLUSTER	25
4.4. CONFIGURING FENCING IN A RED HAT HIGH AVAILABILITY CLUSTER	26
4.4.1. Displaying available fence agents and their options	27
4.4.2. Creating a fence device	28
4.4.3. General properties of fencing devices	28
4.4.4. Fencing delays	35
4.4.5. Testing a fence device	37
4.4.6. Configuring fencing levels	40
4.4.7. Removing a fence level	41
4.4.8. Clearing fence levels	41
4.4.9. Verifying nodes and devices in fence levels	41
4.4.10. Specifying nodes in fencing topology	42
4.4.11. Configuring fencing for redundant power supplies	42
4.4.12. Administering fence devices	43
4.4.12.1. Displaying configured fence devices	43
4.4.12.2. Exporting fence devices as pcs commands	43
4.4.12.3. Exporting fence level configuration	43
4.4.12.4. Modifying and deleting fence devices	44
4.4.12.5. Manually fencing a cluster node	44
4.4.12.6. Disabling a fence device	44
4.4.12.7. Preventing a node from using a fencing device	45
4.4.13. Configuring ACPI for use with integrated fence devices	45

4.4.13.1. Disabling ACPI Soft-Off with the BIOS	46
4.4.13.2. Disabling ACPI Soft-Off in the logind.conf file	47
4.4.13.3. Disabling ACPI completely in the GRUB 2 file	47
4.5. SETTING UP IP ADDRESS RESOURCES ON AWS	47
4.5.1. Creating an IP address resource to manage an IP address exposed to the internet	48
4.5.2. Creating an IP address resource to manage a private IP address limited to a single AWS Availability Zone	49
4.5.3. Creating an IP address resource to manage an IP address that can move across multiple AWS Availability Zones	51
4.5.4. Additional resources	53
4.6. CONFIGURING SHARED BLOCK STORAGE	53
CHAPTER 5. CONFIGURING RHEL ON AWS WITH SECURE BOOT	55
5.1. TYPES OF RHEL AMIS ON AWS	55
5.2. UNDERSTANDING SECURE BOOT FOR RHEL ON CLOUD	55
5.2.1. Components of Secure Boot	55
5.2.2. Stages of Secure Boot for RHEL on Cloud	56
5.3. CONFIGURING A RHEL INSTANCE ON THE AWS MARKETPLACE WITH SECURE BOOT	57
5.4. CONFIGURING A RHEL INSTANCE FROM A CUSTOM RHEL IMAGE WITH SECURE BOOT	60
CHAPTER 6. CONFIGURING RHEL ON PUBLIC CLOUD PLATFORMS WITH AMD SEV SNP	64
6.1. PROPERTIES OF SEV-SNP	64
6.2. UNDERSTANDING AMD SEV SNP SECURE BOOT PROCESS	64
6.3. CONFIGURING A RHEL INSTANCE ON AMAZON WEB SERVICES WITH AMD SEV SNP	65
CHAPTER 7. CONFIGURING RHEL ON PUBLIC CLOUD PLATFORMS WITH UKI	68
7.1. INTRODUCTION TO UNIFIED KERNEL IMAGE	68
7.2. UNDERSTANDING THE UKI SECURE BOOT PROCESS	69

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the [Jira](#) website.
2. Click **Create** in the top navigation bar.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. INTRODUCING RHEL ON PUBLIC CLOUD PLATFORMS

Public cloud platforms offer computing resources as a service. Instead of using on-premise hardware, you can run your IT workloads, including Red Hat Enterprise Linux (RHEL) systems, as public cloud instances.

1.1. BENEFITS OF USING RHEL IN A PUBLIC CLOUD

Running Red Hat Enterprise Linux (RHEL) on public cloud platforms provides flexible resource allocation, cost efficiency, and software-controlled configurations to optimize your infrastructure without managing physical hardware.

RHEL as a cloud instance located on a public cloud platform has the following benefits over RHEL on-premise physical systems or virtual machines (VMs):

Flexible and fine-grained allocation of resources

A cloud instance of RHEL runs as a VM on a cloud platform, which means a cluster of remote servers maintained by the cloud service provider. Therefore, on the software level, allocating hardware resources to the instance is easily customizable, such as a specific type of CPU or storage.

In comparison to a local RHEL system, you are also not limited by the capabilities of physical host. Instead, you can select from a variety of features, based on selections offered by the cloud provider.

Space and cost efficiency

You do not need to own any on-premise servers to host cloud workloads. This avoids the space, power, and maintenance requirements associated with physical hardware.

Instead, on public cloud platforms, you pay the cloud provider directly for using a cloud instance. The cost is typically based on the hardware allocated to the instance and the time to use it. Therefore, you can optimize your costs based on the requirements.

Software-controlled configurations

You save the entire configuration of a cloud instance as data on the cloud platform and control it with software. Therefore, you can easily create, remove, clone, or migrate the instance. You also operate a cloud instance remotely in a cloud provider console, and it connects to remote storage by default.

In addition, you can back up the current state of a cloud instance as a snapshot at any time. Afterwards, you can load the snapshot to restore the instance to the saved state.

Separation from the host and software compatibility

Similarly to a local VM, the RHEL guest operating system on a cloud instance runs on a Kernel-based Virtual Machine (KVM). This kernel is separate from the host operating system and from the *client* system that you use to connect to the instance.

Therefore, you can install any operating system on the cloud instance. This means that on a RHEL public cloud instance, you can run RHEL-specific applications not usable on your local operating system.

In addition, even if the operating system of the instance becomes unstable or compromised, it does not affect your client system.

Additional resources

- [What is public cloud?](#)

- [What is a hyperscaler?](#)
- [Types of cloud computing](#)

1.2. PUBLIC CLOUD USE CASES FOR RHEL

Deploying on a public cloud provides many benefits, but might not be the most efficient solution in every scenario. If you are evaluating whether to migrate your Red Hat Enterprise Linux (RHEL) deployments to the public cloud, consider whether your use case will benefit from the advantages of the public cloud.

Beneficial use cases

- Deploying public cloud instances is very effective for flexibly increasing and decreasing the active computing power of your deployments. This is also known as *scaling up* and *scaling down*. You can use RHEL on public cloud in the following scenarios:
 - Clusters with high peak workloads and low general performance requirements. Scaling up and down based on your demands can be highly efficient in terms of resource costs.
 - Quickly setting up or expanding your clusters. This avoids high upfront costs of setting up local servers.
- What happens in your local environment does not affect cloud instances. Therefore, you can use them for backup and disaster recovery.

Potentially problematic use cases

- You are running an existing environment that you cannot adjust. Customizing a cloud instance to fit the specific needs of an existing deployment might not be economically efficient in comparison with your current host platform.
- You are operating with a hard limit on your budget. Maintaining your deployment in a local data center typically provides less flexibility but more control over the maximum resource costs than the public cloud does.

For details on how to obtain RHEL for public cloud deployments, see [Obtaining RHEL for public cloud deployments](#).

Additional resources

- [Should I migrate my application to the cloud? Here's how to decide.](#)

1.3. FREQUENT CONCERNS WHEN MIGRATING TO A PUBLIC CLOUD

Moving your RHEL workloads from a local environment to a public cloud platform might raise concerns about the changes involved. The following are the most commonly asked questions:

Will my RHEL work differently as a cloud instance than as a local virtual machine?

In most respects, RHEL instances on a public cloud platform work the same as RHEL virtual machines on a local host, such as an on-premise server. Notable exceptions include:

- Instead of private orchestration interfaces, public cloud instances use provider-specific console interfaces for managing your cloud resources.

- Certain features, such as nested virtualization, might not work correctly. If a specific feature is critical for your deployment, check the feature's compatibility in advance with your chosen public cloud provider.

Will my data stay safe in a public cloud as opposed to a local server?

The data in your RHEL cloud instances is in your ownership, and your public cloud provider does not have any access to it. In addition, major cloud providers support data encryption in transit, which improves the security of data when migrating your virtual machines to the public cloud.

In terms of security of RHEL public cloud instances, the following applies:

- Your public cloud provider is responsible for the security of the cloud hypervisor
- Red Hat provides the security features of the RHEL guest operating systems in your instances
- You manage the specific security settings and practices in your cloud infrastructure

What effect does my geographic region have on the functionality of RHEL public cloud instances?

You can use RHEL instances on a public cloud platform regardless of your geographical location. Therefore, you can run your instances in the same region as your on-premises server. However, hosting your instances in a physically distant region might cause high latency when operating them. In addition, depending on the public cloud provider, certain regions might offer additional features or be more cost-efficient. Before creating your RHEL instances, review the properties of the hosting regions available for your chosen cloud provider.

1.4. OBTAINING RHEL FOR PUBLIC CLOUD DEPLOYMENTS

To deploy Red Hat Enterprise Linux (RHEL) in a public cloud environment, you must select a certified cloud service provider and create a RHEL cloud instance.

- Based on your requirements and the current offerings in the market, select the optimal cloud service provider for your use case. The certified cloud service providers for running RHEL instances are:
 - [Amazon Web Services \(AWS\)](#)
 - [Google Cloud](#)
 - [Microsoft Azure](#)



NOTE

This document specifically addresses the process of deploying RHEL on AWS.

- Create a RHEL cloud instance on your cloud platform. For details, see [Methods for creating RHEL cloud instances](#).
- To keep your RHEL deployment up-to-date, use [Red Hat Update Infrastructure \(RHUI\)](#).

Additional resources

- [RHUI documentation](#)

- [Red Hat Open Hybrid Cloud](#)
- [Red Hat Ecosystem Catalog](#)

1.5. METHODS FOR CREATING RHEL CLOUD INSTANCES

You can create a Red Hat Enterprise Linux (RHEL) system image and import it to the cloud platform by using either RHEL image builder or purchasing a RHEL image directly from the cloud service provider marketplace. You can then deploy the RHEL image as a cloud instance.

To deploy a RHEL instance on a public cloud platform, you can use either of the following methods:

Create a RHEL system image and import it to the cloud platform

- To create the system image, you can use the [RHEL image builder](#) or build the image manually.
- This method uses your existing RHEL subscription. This is also referred to as *bring your own subscription* (BYOS).
- You pre-pay a yearly subscription, and you can use your Red Hat customer discount.
- Red Hat provides customer service.
- For creating many images effectively, you can use the **cloud-init** utility.

Purchase a RHEL instance directly from the cloud provider marketplace

- You post-pay an hourly rate for using the service. This method is also referred to as *pay as you go* (PAYG).
- The cloud service provider provides customer service.

Additional resources

- [What is a golden image?](#)

CHAPTER 2. PREPARING AND UPLOADING AMI IMAGES TO AWS

You can create custom images and update them, either manually or automatically, on the AWS cloud with RHEL image builder.

2.1. PREPARING TO MANUALLY UPLOAD AWS AMI IMAGES

Before uploading an AWS AMI image, you must configure a system for uploading the images.

Prerequisites

- You must have an Access Key ID configured in the [AWS IAM account manager](#).
- You must have a writable [S3 bucket](#) prepared.

Procedure

1. Install Python 3 and the **pip** tool:

```
# dnf install python3 python3-pip
```

2. Install the [AWS command-line tools](#) with **pip**:

```
# pip3 install awscli
```

3. Set your profile. The terminal prompts you to provide your credentials, region, and output format:

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

4. Define a name for your bucket and create a bucket:

```
$ BUCKET=bucketname
$ aws s3 mb s3://$BUCKET
```

Replace ***bucketname*** with the actual bucket name. It must be a globally unique name. As a result, your bucket is created.

5. To grant permission to access the S3 bucket, create a **vmimport** S3 Role in the AWS Identity and Access Management (IAM), if you have not already done so in the past:
 - a. Create a **trust-policy.json** file with the trust policy configuration in JSON format. For example:

```
{
  "Version": "2022-10-17",
  "Statement": [{
    "Effect": "Allow",
```

```

    "Principal": {
      "Service": "vmie.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "sts:Externalid": "vmimport"
      }
    }
  }
}

```

- b. Create a **role-policy.json** file with the role policy configuration, in the JSON format. For example:

```

{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["s3:GetBucketLocation", "s3:GetObject", "s3:ListBucket"],
    "Resource": ["arn:aws:s3:::%s", "arn:aws:s3:::%s/*"], { "Effect": "Allow", "Action":
["ec2:ModifySnapshotAttribute", "ec2:CopySnapshot", "ec2:RegisterImage",
"ec2:Describe"],
    "Resource": "*"
  }
  ]
}
$BUCKET $BUCKET

```

- c. Create a role for your Amazon Web Services account, by using the **trust-policy.json** file:

```

$ aws iam create-role --role-name vmimport --assume-role-policy-document file://trust-policy.json

```

- d. Embed an inline policy document by using the **role-policy.json** file:

```

$ aws iam put-role-policy --role-name vmimport --policy-name vmimport --policy-document file://role-policy.json

```

Additional resources

- [Using high-level \(s3\) commands with the AWS CLI](#)

2.2. MANUALLY UPLOADING AN AMI IMAGE TO AWS BY USING THE CLI

You can use RHEL image builder to build **ami** images and manually upload them directly to the Amazon AWS Cloud service provider by using the CLI.

Prerequisites

- You have an **Access Key ID** configured in the [AWS IAM](#) account manager.
- You have a writable [S3 bucket](#) prepared.

- You have a defined blueprint.

Procedure

1. Build the image:

```
# image-builder build ami \
--blueprint blueprint-name \
--aws-region us-east-1 \
--aws-bucket <example-bucket> \
--aws-ami-name <image-name> \
```

2. Upload the image to AWS:

```
# image-builder upload <image-name> \
```

Verification

1. Confirm that the image upload was successful by accessing [EC2](#) on the menu and selecting the correct region in the AWS console. The image must have the **available** status to indicate that it was successfully uploaded.
2. On the dashboard, select your image and click **Launch**.

Additional resources

- [Required service role to import a VM](#)

2.3. CREATING AND AUTOMATICALLY UPLOADING IMAGES TO THE AWS CLOUD AMI

You can create a **.raw** image by using RHEL image builder, and choose to check the **Upload to AWS** checkbox to automatically push the output image that you create directly to the **Amazon AWS Cloud AMI** service provider.

Prerequisites

- You must have **root** or **wheel** group user access to the system.
- You have opened the RHEL image builder interface of the RHEL web console in a browser.
- You have created a blueprint. See [Creating a blueprint in the web console interface](#) .
- You must have an Access Key ID configured in the [AWS IAM](#) account manager.
- You must have a writable [S3 bucket](#) prepared.

Procedure

1. In the RHEL image builder dashboard, click the **blueprint name** that you previously created.
2. Select the tab **Images**.

3. Click **Create Image** to create your customized image.
The **Create Image** window opens.
 - a. From the **Type** drop-down menu list, select **Amazon Machine Image Disk (.raw)**.
 - b. Check the **Upload to AWS** checkbox to upload your image to the AWS Cloud and click **Next**.
 - c. To authenticate your access to AWS, type your **AWS access key ID** and **AWS secret access key** in the corresponding fields. Click **Next**.

**NOTE**

You can view your AWS secret access key only when you create a new Access Key ID. If you do not know your Secret Key, generate a new Access Key ID.

- d. Type the name of the image in the **Image name** field, type the Amazon bucket name in the **Amazon S3 bucket name** field, and type the **AWS region** field for the bucket you are going to add your customized image to. Click **Next**.
 - e. Review the information and click **Finish**.
Optionally, click **Back** to modify any incorrect details.

**NOTE**

You must have the correct IAM settings for the bucket you are going to send your customized image to. This procedure uses the IAM Import and Export, so you have to set up a **policy** for your bucket before you are able to upload images to it. For more information, see [Required Permissions for IAM Users](#).

4. A pop-up on the upper right informs you of the saving progress. It also informs that the image creation has been initiated, the progress of this image creation, and the subsequent upload to the AWS Cloud.
After the process is complete, you can see the **Image build complete** status.
5. In a browser, access [Service→EC2](#).
 - a. On the AWS console dashboard menu, choose the [correct region](#). The image must have the **Available** status to indicate that it is uploaded.
 - b. On the AWS dashboard, select your image and click **Launch**.
6. A new window opens. Choose an instance type according to the resources you need to start your image. Click **Review and Launch**.
7. Review your instance start details. You can edit each section if you need to make any changes. Click **Launch**.
8. Before you start the instance, select a public key to access it.
You can either use the key pair you already have or you can create a new key pair.

Follow the next steps to create a new key pair in EC2 and attach it to the new instance.

- a. From the drop-down menu list, select **Create a new key pair**.

- b. Enter the name of the new key pair. It generates a new key pair.
 - c. Click **Download Key Pair** to save the new key pair on your local system.
9. Then, you can click **Launch Instance** to start your instance.
You can check the status of the instance, which displays as **Initializing**.
10. After the instance status is **running**, the **Connect** button becomes available.
11. Click **Connect**. A window is displayed with instructions on how to connect by using SSH.
 - a. Select **A standalone SSH client** as the preferred connection method to open a terminal.
 - b. In the location where you store your private key, ensure that your key is publicly viewable for SSH to work. To do so, run the command:

```
$ chmod 400 <your_instance_name>.pem
```
 - c. Connect to your instance by using its Public DNS:

```
$ ssh -i <your-instance_name>.pem ec2-user@<your-instance-IP-address>
```
 - d. Type **yes** to confirm that you want to continue connecting.
As a result, you are connected to your instance over SSH.

Verification

- Check if you are able to perform any action while connected to your instance by using SSH.

Additional resources

- [Open a case on Red Hat Customer Portal](#)
- [Connecting to your Linux instance by using SSH](#)

CHAPTER 3. DEPLOYING A RHEL IMAGE AS AN EC2 INSTANCE ON AWS

To use a Red Hat Enterprise Linux (RHEL) image on Amazon Web Services (AWS), you need to convert the RHEL image to an AWS-compatible format, such as Amazon Machine Image (AMI). From an AMI, you can launch an Elastic Cloud Compute (EC2) instance.

You can create, customize, and deploy a RHEL AMI by using RHEL image builder or manual configuration methods.

- Use RHEL image builder. For instructions, see [Preparing and uploading AMI images to AWS](#) and [AWS specific resources list](#).
- Manually create and configure an AMI. This is a more complicated process but offers more granular customization options.

3.1. AVAILABLE RHEL IMAGE TYPES FOR PUBLIC CLOUD

To deploy your RHEL virtual machine VM on a certified cloud service provider (CCSP), you can use several options. The following table lists the available image types, subscriptions, considerations, and sample scenarios for the image types.



NOTE

To deploy customized ISO images, you can use RHEL image builder. With RHEL image builder, you can create, upload, and deploy these custom images specific to your chosen CCSP. For details, see [Composing a Customized RHEL System Image](#).

Table 3.1. Image options

Image types	Subscriptions	Considerations	Sample scenario
Deploy a Red Hat gold image	Use your existing Red Hat subscriptions	The subscriptions include the Red Hat product cost and support for Cloud Access images, while you pay the CCSP for all other instance costs	Select a Red Hat gold image on the CCSP. For details on gold images and how to access them on the CCSP, see the Red Hat Cloud Access Reference Guide
Deploy a custom image that you move to the CCSP	Use your existing Red Hat subscriptions	The subscriptions includes the Red Hat product cost and support for custom RHEL image, while you pay the CCSP for all other instance costs	Upload your custom image and attach your subscriptions

Image types	Subscriptions	Considerations	Sample scenario
Deploy an existing RHEL based custom machine image	The custom machine images include a RHEL image	You pay the CCSP on an hourly basis based on a <i>pay-as-you-go</i> model. For this model, on-demand images are available on the CCSP marketplace. The CCSP provides support for these images, while Red Hat handles updates. The CCSP provides updates through the Red Hat Update Infrastructure (RHUI)	Select a RHEL image when you launch an instance on the CCSP cloud management console, or choose an image from the CCSP marketplace.

To convert an *on-demand, license-included* EC2 instance to a *bring-your-own-license (BYOL)* EC2 instance of RHEL, see [Convert a license type for Linux in License Manager](#) .

Additional resources

- [Red Hat Ecosystem Catalog](#)
- [Red Hat Cloud Access Reference Guide](#)
- [Using gold images on Amazon Web Services \(AWS\)](#)
- [Amazon Web Services Red Hat Ecosystem Catalogue](#)
- [Red Hat Enterprise Linux on Amazon EC2 FAQs](#)
- [Convert a license type for Linux in License Manager](#)

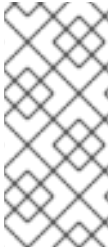
3.2. DEPLOYING A RHEL INSTANCE BY USING A CUSTOM BASE IMAGE

To manually configure a virtual machine (VM), first create a base (starter) image. Then, you can modify configuration settings and add the packages the VM requires to operate on the cloud. You can also make additional configuration changes for your specific application after you upload the image.

Creating a VM from a base image has the following advantages:

- Fully customizable
- High flexibility for any use case
- Lightweight - includes only the operating system and the required runtime libraries

To create a custom base image of RHEL from an ISO image, you can use the command line interface (CLI) or the web console for creating and configuring VM.



NOTE

Verify the following VM configurations.

- SSH - Enable SSH to give remote access to your VM.
- DHCP - Configure the primary virtual adapter to use DHCP.

Prerequisites

- You have [enabled virtualization](#) on the host machine.
- For web console, ensure the following options:
- You have not checked the **Immediately Start VM** option.
- You have already changed the **Memory** size to your preferred settings.
- You have changed the **Model** option under **Virtual Network Interface Settings** to **virtio** and **vCPUs** to the capacity settings for the VM.

Procedure

1. Configure the Red Hat Enterprise Linux (RHEL) VM:
 - a. To install from the command line (CLI), ensure that you set the default memory, network interfaces, and CPUs according to your requirement for the VM. For details, see [Creating virtual machines by using the command line](#)
 - b. To install from the web console, see [Creating virtual machines by using the web console](#)
2. When the installation starts:
 - a. Create a **root** password.
 - b. Create an administrative user account.
3. After the installation completes, reboot the VM and log in to the **root** account.
4. After logging in as **root**, you can configure the image.
5. Register the VM and enable the RHEL repository:

```
# subscription-manager register
```

6. For AMD64 or Intel 64 (x86_64) VMs, install the **nvme**, **xen-netfront**, and **xen-blkfront** drivers:

```
# dracut -f --add-drivers "nvme xen-netfront xen-blkfront"
```

7. For ARM 64 (aarch64) VMs, install the **nvme** driver:

```
# dracut -f --add-drivers "nvme"
```

Including these drivers prevents a **dracut** time-out.

Alternatively, you can add the drivers to `/etc/dracut.conf.d/` and then enter **dracut -f** to overwrite the existing **initramfs** file.

Verification

- Verify if the system has the **cloud-init** package and enable it:

```
# dnf install cloud-init
# systemctl enable --now cloud-init.service
```

- Power off the VM.

3.3. UPLOADING A RHEL IMAGE TO AWS BY USING THE COMMAND LINE

To run a RHEL instance on Amazon Web Services (AWS), you must first upload a RHEL image to AWS. To configure and manage a RHEL EC2 instance on AWS, use the **awscli2** utility.

3.3.1. Installing AWSCLI2

You can use the Amazon Web Services (AWS) command line interface **awscli2** utility to configure and manage Red Hat Enterprise Linux (RHEL) images and Red Hat high availability (HA) cluster on AWS.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have access to an AWS Access Key ID and an AWS Secret Access Key. For details, see [manage access keys](#).

Procedure

- Install **awscli2**:

```
# dnf install awscli2
```

Verification

1. Verify the installation:

```
$ aws --version
aws-cli/1.19.77 Python/3.6.15 Linux/5.14.16-201.fc34.x86_64 botocore/1.20.77
```

2. Configure **awscli2** for AWS credentials and settings:

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

3.3.2. Converting and pushing an image to Amazon S3

You can convert a Red Hat Enterprise Linux (RHEL) image in the **qcow2** image format to **OVA**, **VHD**, **VHDX**, **VMDK**, or **raw** by using the **qemu-img** utility, and then upload it to Amazon S3 storage.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have created [an Amazon S3 bucket by using awscli2](#) to upload the RHEL image.

Procedure

1. Run **qemu-img** to convert **.qcow2** image to **.raw** image format:

```
# qemu-img convert -f qcow2 -O raw rhel-10.0-sample.qcow2 rhel-10.0-sample.raw
```

2. Upload the image to the Amazon S3 bucket:

```
$ aws s3 cp rhel-10.0-sample.raw s3://<example-s3-bucket-name>
```

Verification

- Check the [AWS S3 Console](#) to confirm successful upload.

Additional resources

- [Create a S3 bucket](#)
- [Supported image formats by AWS](#)

3.3.3. Managing a RHEL VM on AWS by using the command line

You can use the Amazon Web Services (AWS) command line interface **awscli2** utility to manage a Red Hat Enterprise Linux (RHEL) Elastic Cloud Compute (EC2) VM on AWS through the command line. You can import a RHEL EC2 image snapshot, create an AMI, launch, and connect to a RHEL EC2 instance.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).

Procedure

1. Use the **vmimport** role: An alternate to import the RHEL image to the Amazon S3 bucket is by using the **vmimport** role. See [Required permissions for VM Import/Export](#).
2. Import RHEL image as a snapshot: You can import RHEL VM image from Amazon S3 as a snapshot to Amazon EC2. For details, see [Start an import snapshot task](#) and [Monitor an import snapshot task](#).

3. Create and launch a RHEL EC2 instance: You can create a RHEL Amazon Machine Image (AMI) from existing snapshot and launch a RHEL EC2 instance. For details, see [create an AMI from snapshot by using awscli2](#) and [launching, listing, and deleting RHEL instance by using awscli2](#) .
4. Configure the private key and connect to the RHEL EC2 instance: You can configure your **<example_key>.pem** file and connect to an RHEL EC2 instance. For details, see [Create a key pair using Amazon EC2](#) and [Connect using the AWS CLI](#).

Additional resources

- [Amazon S3 general purpose buckets](#)
- [Amazon Web Services Red Hat Ecosystem Catalog](#)
- [Red Hat Enterprise Linux on Amazon EC2 FAQs](#)

3.3.4. Attaching Red Hat subscriptions

To register and attach your Red Hat subscription to a RHEL instance, you can use the **subscription-manager** command.

Prerequisites

- You have an active [Red Hat account](#) .

Procedure

1. Register your system:

```
# subscription-manager register
```

2. Attach your subscriptions:

- You can use an activation key to attach subscriptions. See [Creating Red Hat Customer Portal Activation Keys](#) for more information.
- Otherwise, you can manually attach a subscription by using the ID of the subscription pool (Pool ID). See [Attaching a host-based subscription to hypervisors](#) .

3. Optional: To collect various system metrics about the instance in the [Red Hat Hybrid Cloud Console](#), you can register the instance with [Red Hat Lightspeed](#) .

```
# insights-client register --display-name <display_name_value>
```

For information about further configuration of Red Hat Lightspeed, see [Client Configuration Guide for Red Hat Lightspeed](#).

Additional resources

- [Creating Red Hat Customer Portal Activation Keys](#)
- [Client Configuration Guide for Red Hat Lightspeed](#)
- [Red Hat Cloud Access Reference Guide](#)

3.3.5. Setting up automatic registration on AWS gold images

You can deploy Red Hat Enterprise Linux (RHEL) virtual machines (VMs) more efficiently on Amazon Web Services (AWS) by using gold images of RHEL. This ensures that the VMs are automatically registered to the Red Hat Subscription Manager (RHSM).

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have downloaded the latest RHEL gold image for AWS. For instructions, see [Using gold images on AWS](#).



NOTE

You can only attach an AWS account to a single Red Hat account at a time. Therefore, ensure no other users require access to the AWS account before attaching it to your Red Hat one.

Procedure

1. Upload the gold image to AWS. For instructions, see one of the following:
 - [Uploading a RHEL image to AWS by using the command line](#)
 - [Uploading a RHEL image to AWS by using the AWS console](#)
2. Create VMs by using the uploaded image. If your RHSM settings are correct, they will be automatically subscribed to RHSM.

Verification

- In a RHEL VM created using the above instructions, verify the system is registered to RHSM by executing the **subscription-manager identity** command. On a successfully registered system, this displays the UUID of the system. For example:

```
# subscription-manager identity
system identity: fdc46662-c536-43fb-a18a-bbcb283102b7
name: 192.168.122.222
org name: 6340056
org ID: 6340056
```

Additional resources

- [AWS Management Console](#)
- [Adding cloud integrations to the Hybrid Cloud Console](#)

3.4. UPLOADING A RHEL IMAGE TO AWS BY USING THE AWS CONSOLE

To run a RHEL instance on Amazon Web Services (AWS), you must first upload the RHEL image to AWS. To configure and manage the RHEL EC2 instance on AWS, use the **awscli2** utility.

3.4.1. Converting and pushing an image to S3 by using the AWS console

You can convert a RHEL image in the **qcow2** image format to **OVA**, **VHD**, **VHDX**, **VMDK**, or **raw** by using the **qemu-img** utility, and then upload it to Amazon S3 storage by using the AWS console.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have created [an Amazon S3 bucket by using the Amazon S3 console](#) to upload the RHEL image.

Procedure

1. Run **qemu-img** to convert **.qcow2** image to **.raw** image format:

```
# qemu-img convert -f qcow2 -O raw rhel-10.0-sample.qcow2 rhel-10.0-sample.raw
```

2. Upload the image to the S3 bucket by using [Amazon S3 console](#)

Verification

- Check the [AWS S3 Console](#) to confirm successful upload.

Additional resources

- [Create a S3 bucket](#)

3.4.2. Managing a RHEL VM on AWS by using the AWS console

You can manage a Red Hat Enterprise Linux (RHEL) Elastic Cloud Compute (EC2) VM on Amazon Web Services (AWS) by using the AWS console. You can create RHEL EC2 image snapshots, manage Amazon Machine Image (AMI), launch, and connect to a RHEL EC2 instance.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have pushed your RHEL image to the Amazon S3 bucket by using the AWS console. For details, see [Converting and pushing an image to S3 by using the AWS console](#).

Procedure

1. Use the **vmimport** role: An alternate way to import the RHEL image to the Amazon S3 bucket is by using the **vmimport** role. See [Import your VM as an image](#).

2. Import a RHEL image as a snapshot: You can import a RHEL VM image from Amazon S3 as a snapshot to Amazon EC2. For details, see [Importing a disk as a snapshot using VM Import/Export](#) and [Monitor an import snapshot task](#) .
3. Create and launch a RHEL EC2 instance: You can create a RHEL Amazon Machine Image (AMI) from an existing snapshot and launch a RHEL EC2 instance. For details, see [Create an AMI from a snapshot](#) and [Launch an instance using defined parameters](#) .
4. Configure the private key and connect to the RHEL EC2 instance: You can configure your **<example_key>.pem** file and connect to an RHEL EC2 instance. For details, see [Create a key pair using Amazon EC2](#) and [Connect using the Amazon EC2 console](#) .
5. For Red Hat subscriptions, see [Attaching Red Hat subscriptions](#)

Additional resources

- [Amazon S3 general purpose buckets](#)
- [Red Hat Cloud Access Reference Guide](#)
- [Red Hat Enterprise Linux on Amazon EC2 - FAQs](#)
- [Red Hat on Amazon Web Services](#)

CHAPTER 4. CONFIGURING A RED HAT HIGH AVAILABILITY CLUSTER ON AWS

You can configure a high availability (HA) cluster on Amazon Web Services (AWS) to group Red Hat Enterprise Linux (RHEL) nodes and automatically redistribute workloads if a node fails. The process for setting up HA clusters on AWS is comparable to configuring them in traditional, non-cloud environments.

You have several options for obtaining RHEL images for the cluster. For details, see [Available RHEL image types for public cloud](#).

4.1. BENEFITS OF USING HIGH-AVAILABILITY CLUSTERS ON PUBLIC CLOUD PLATFORMS

A high-availability (HA) cluster links a set of computers (called *nodes*) to run a specific workload. HA clusters offer redundancy to handle hardware or software failures. When a node in the HA cluster fails, the Pacemaker cluster resource manager quickly distributes the workload to other nodes, ensuring that services on the cluster continue without noticeable downtime.

You can also run HA clusters on public cloud platforms. In this case, you would use virtual machine (VM) instances in the cloud as the individual cluster nodes. Using HA clusters on a public cloud platform has the following benefits:

- *Improved availability:* In case of a VM failure, the workload is quickly redistributed to other nodes, so running services are not disrupted.
- *Scalability:* You can start additional nodes when demand is high and stop them when demand is low.
- *Cost-effectiveness:* With the pay-as-you-go pricing, you pay only for nodes that are running.
- *Simplified management:* Some public cloud platforms offer management interfaces to make configuring HA clusters easier.

To enable HA on your RHEL systems, Red Hat offers a HA Add-On. You can configure a RHEL cluster with Red Hat HA Add-On to manage HA clusters with groups of RHEL servers. Red Hat HA Add-On gives access to integrated and streamlined tools. With cluster resource manager, fencing agents, and resource agents, you can set up and configure the cluster for automation. The Red Hat HA Add-On offers the following components for automation:

- **Pacemaker**, a cluster resource manager that offers both a command line utility (**pcs**) and a GUI (**pcsd**) to support many nodes
- **Corosync** and **Kronosnet** to create and manage HA clusters
- Resource agents to configure and manage custom applications
- Fencing agents to use cluster on platforms such as bare-metal servers and virtual machines

The Red Hat HA Add-On handles critical tasks such as node failures, load balancing, and node health checks for fault tolerance and system reliability.

Additional resources

- [High Availability Add-On overview](#)

4.2. INSTALLING THE HIGH AVAILABILITY PACKAGES AND AGENTS

To configure a Red Hat High Availability cluster on AWS, install the High Availability packages and agents on each node in the cluster.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have completed the configuration for [Uploading RHEL image to AWS by using the command line](#).

Procedure

1. Remove the AWS Red Hat Update Infrastructure (RHUI) client.

```
$ sudo -i
# dnf -y remove rh-amazon-rhui-client
```

2. Register the VM with Red Hat.

```
# subscription-manager register
```

3. Disable all repositories.

```
# subscription-manager repos --disable=
```

4. Enable the RHEL 10 Server HA repositories.

```
# subscription-manager repos --enable=rhel-10-for-x86_64-highavailability-rpms
```

5. Update the RHEL AWS instance.

```
# dnf update -y
```

6. Install the Red Hat High Availability Add-On software packages, along with the AWS fencing agent from the High Availability channel.

```
# dnf install pcs pacemaker fence-agents-aws
```

7. The user **hacluster** was created during the **pcs** and **pacemaker** installation in the earlier step. Create a password for **hacluster** on all cluster nodes. Use the same password for all nodes.

```
# passwd hacluster
```

8. Add the **high availability** service to the RHEL Firewall if **firewalld.service** is installed.

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --reload
```

9. Start the **pcs** service and enable it to start on boot.

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

10. Edit **/etc/hosts** and add Red Hat Enterprise Linux (RHEL) host names and internal IP addresses. See [How should the /etc/hosts file be set up on RHEL cluster nodes?](#) for details.

Verification

- Ensure the **pcs** service is running.

```
# systemctl status pcsd.service

pcsd.service - PCS GUI and remote configuration interface
Loaded: loaded (/usr/lib/systemd/system/pcsd.service; enabled; vendor preset: disabled)
Active: active (running) since Thu 2018-03-01 14:53:28 UTC; 28min ago
Docs: man:pcsd(8)
man:pcs(8)
Main PID: 5437 (pcsd)
CGroup: /system.slice/pcsd.service
└─5437 /usr/bin/ruby /usr/lib/pcsd/pcsd > /dev/null &
Mar 01 14:53:27 ip-10-0-0-48.ec2.internal systemd[1]: Starting PCS GUI and remote
configuration interface...
Mar 01 14:53:28 ip-10-0-0-48.ec2.internal systemd[1]: Started PCS GUI and remote
configuration interface.
```

4.3. CREATING A HIGH AVAILABILITY CLUSTER

You can create a Red Hat High Availability Add-On cluster. This example uses nodes **z1.example.com** and **z2.example.com**.



NOTE

To display the parameters of a **pcs** command and a description of those parameters, use the **-h** option of the **pcs** command.

Prerequisites

- You have created [a Red Hat account](#)
- You have [signed up and set up an Azure account](#).

Procedure

1. Authenticate the **pcs** user **hacluster** for each node in the cluster on the node from which you will be running **pcs**.

The following command authenticates user **hacluster** on **z1.example.com** for both of the nodes in a two-node cluster that will consist of **z1.example.com** and **z2.example.com**.

```
[root@z1 ~]# pcs host auth z1.example.com z2.example.com
Username: hacluster
Password:
```

```
z1.example.com: Authorized
z2.example.com: Authorized
```

- Execute the following command from **z1.example.com** to create the two-node cluster **my_cluster** that consists of nodes **z1.example.com** and **z2.example.com**. This will propagate the cluster configuration files to both nodes in the cluster. This command includes the **--start** option, which will start the cluster services on both nodes in the cluster.

```
[root@z1 ~]# pcs cluster setup my_cluster --start z1.example.com z2.example.com
```

- Enable the cluster services to run on each node in the cluster when the node is booted.



NOTE

For your particular environment, you can skip this step by keeping the cluster services disabled. If enabled and a node goes down, any issues with your cluster or your resources are resolved before the node rejoins the cluster. If you keep the cluster services disabled, you need to manually start the services when you reboot a node by executing the **pcs cluster start** command on that node.

```
[root@z1 ~]# pcs cluster enable --all
```

- Display the status of the cluster you created with the **pcs cluster status** command. Because there could be a slight delay before the cluster is up and running when you start the cluster services with the **--start** option of the **pcs cluster setup** command, you should ensure that the cluster is up and running before performing any subsequent actions on the cluster and its configuration.

```
[root@z1 ~]# pcs cluster status
Cluster Status:
Stack: corosync
Current DC: z2.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
Last updated: Thu Oct 11 16:11:18 2018
Last change: Thu Oct 11 16:11:00 2018 by hacluster via crmd on z2.example.com
2 Nodes configured
0 Resources configured
```

```
...
```

4.4. CONFIGURING FENCING IN A RED HAT HIGH AVAILABILITY CLUSTER

When a node becomes unresponsive, the cluster must isolate it to prevent data corruption. Since the node cannot be contacted directly, you must configure fencing. An external fence device cuts off the node's access to shared resources or performs a hard reboot.

Without a fence device configured you do not have a way to know that the resources previously used by the disconnected cluster node have been released, and this could prevent the services from running on any of the other cluster nodes. Conversely, the system may assume erroneously that the cluster node has released its resources and this can lead to data corruption and data loss. Without a fence device configured data integrity cannot be guaranteed and the cluster configuration will be unsupported.

When the fencing is in progress no other cluster operation is allowed to run. Normal operation of the cluster cannot resume until fencing has completed or the cluster node rejoins the cluster after the cluster node has been rebooted. For more information about fencing and its importance in a Red Hat High Availability cluster, see the Red Hat Knowledgebase solution [Fencing in a Red Hat High Availability Cluster](#).

4.4.1. Displaying available fence agents and their options

You can view available fencing agents and the available options for specific fencing agents.



NOTE

Your system's hardware determines the type of fencing device to use for your cluster. For information about supported platforms and architectures and the different fencing devices, see the Red Hat Knowledgebase article [Cluster Platforms and Architectures](#) section of the article [Support Policies for RHEL High Availability Clusters](#).

Run the following command to list all available fencing agents. When you specify a filter, this command displays only the fencing agents that match the filter.

```
# pcs stonith list [filter]
```

Run the following command to display the options for the specified fencing agent.

```
# pcs stonith describe [stonith_agent]
```

For example, the following command displays the options for the fence agent for APC over telnet/SSH.

```
# pcs stonith describe fence_apc
Stonith options for: fence_apc
ipaddr (required): IP Address or Hostname
login (required): Login Name
passwd: Login password or passphrase
passwd_script: Script to retrieve password
cmd_prompt: Force command prompt
secure: SSH connection
port (required): Physical plug number or name of virtual machine
identity_file: Identity file for ssh
switch: Physical switch number on device
inet4_only: Forces agent to use IPv4 addresses only
inet6_only: Forces agent to use IPv6 addresses only
ipport: TCP port to use for connection with device
action (required): Fencing Action
verbose: Verbose mode
debug: Write debug information to given file
version: Display version information and exit
help: Display help and exit
separator: Separator for CSV created by operation list
power_timeout: Test X seconds for status change after ON/OFF
shell_timeout: Wait X seconds for cmd prompt after issuing command
login_timeout: Wait X seconds for cmd prompt after login
power_wait: Wait X seconds after issuing ON/OFF
delay: Wait X seconds before fencing is started
retry_on: Count of attempts to retry power on
```

**WARNING**

For fence agents that provide a **method** option, with the exception of the **fence_sbd** agent a value of **cycle** is unsupported and should not be specified, as it may cause data corruption. Even for **fence_sbd**, however, you should not specify a method and instead use the default value.

4.4.2. Creating a fence device

Create a fence device using the **pcs stonith create** command. To view all available creation options, use the **pcs stonith -h** command.

Procedure

- Create a fence device:

```
# pcs stonith create stonith_id stonith_device_type [stonith_device_options] [operation_action operation_options]
```

- The following command creates a single fencing device for a single node:

```
# pcs stonith create MyStonith fence_virt pcmk_host_list=f1 op monitor interval=30s
```

Some fence devices can fence only a single node, while other devices can fence multiple nodes. The parameters you specify when you create a fencing device depend on what your fencing device supports and requires.

- Some fence devices can automatically determine what nodes they can fence.
- You can use the **pcmk_host_list** parameter when creating a fencing device to specify all of the machines that are controlled by that fencing device.
- Some fence devices require a mapping of host names to the specifications that the fence device understands. You can map host names with the **pcmk_host_map** parameter when creating a fencing device.

Additional resources

- [General properties of fencing devices](#).
- [Testing a fence device](#).

4.4.3. General properties of fencing devices

Configure fencing behavior using device-specific options and cluster-wide properties. Device options define agent settings, such as IP addresses, and metadata like delays. Cluster properties manage global logic, including timeouts and the **stonith-enabled** parameter.

Any cluster node can fence any other cluster node with any fence device, regardless of whether the fence resource is started or stopped. Whether the resource is started controls only the recurring monitor for the device, not whether it can be used, with the following exceptions:

- You can disable a fencing device by running the **pcs stonith disable stonith_id** command. This will prevent any node from using that device.
- To prevent a specific node from using a fencing device, you can configure location constraints for the fencing resource with the **pcs constraint location ... avoids** command.
- Configuring **stonith-enabled=false** will disable fencing altogether. Note, however, that Red Hat does not support clusters when fencing is disabled, as it is not suitable for a production environment.

The following table describes the general properties you can set for fencing devices.

Table 4.1. General Properties of Fencing Devices

Field	Type	Default	Description
pcmk_host_map	string		A mapping of host names to port numbers for devices that do not support host names. For example: node1:1;node2:2,3 tells the cluster to use port 1 for node1 and ports 2 and 3 for node2. The pcmk_host_map property supports special characters inside pcmk_host_map values using a backslash in front of the value. For example, you can specify pcmk_host_map="node3:plug\1" to include a space in the host alias.
pcmk_host_list	string		A list of machines controlled by this device (Optional unless pcmk_host_check=static-list).
pcmk_host_check	string	<ul style="list-style-type: none"> * static-list if either pcmk_host_list or pcmk_host_map is set * Otherwise, dynamic-list if the fence device supports the list action * Otherwise, status if the fence device supports the status action * Otherwise, none. 	How to determine which machines are controlled by the device. Allowed values: dynamic-list (query the device), static-list (check the pcmk_host_list attribute), none (assume every device can fence every machine)

The following table summarizes additional properties you can set for fencing devices. Note that these properties are for advanced use only.

Table 4.2. Advanced Properties of Fencing Devices

Field	Type	Default	Description
pcmk_host_argument	string	port	An alternate parameter to supply instead of port. Some devices do not support the standard port parameter or may provide additional ones. Use this to specify an alternate, device-specific parameter that should indicate the machine to be fenced. A value of none can be used to tell the cluster not to supply any additional parameters.
pcmk_reboot_action	string	reboot	An alternate command to run instead of reboot . Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific, command that implements the reboot action.
pcmk_reboot_timeout	time	60s	Specify an alternate timeout to use for reboot actions instead of stonith-timeout . Some devices need much more/less time to complete than normal. Use this to specify an alternate, device-specific, timeout for reboot actions.
pcmk_reboot_retries	integer	2	The maximum number of times to retry the reboot command within the timeout period. Some devices do not support multiple connections. Operations may fail if the device is busy with another task so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries reboot actions before giving up.
pcmk_off_action	string	off	An alternate command to run instead of off . Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific, command that implements the off action.

Field	Type	Default	Description
pcmk_off_timeout	time	60s	Specify an alternate timeout to use for off actions instead of stonith-timeout . Some devices need much more or much less time to complete than normal. Use this to specify an alternate, device-specific, timeout for off actions.
pcmk_off_retries	integer	2	The maximum number of times to retry the off command within the timeout period. Some devices do not support multiple connections. Operations may fail if the device is busy with another task so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries off actions before giving up.
pcmk_list_action	string	list	An alternate command to run instead of list . Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific, command that implements the list action.
pcmk_list_timeout	time	60s	Specify an alternate timeout to use for list actions. Some devices need much more or much less time to complete than normal. Use this to specify an alternate, device-specific, timeout for list actions.
pcmk_list_retries	integer	2	The maximum number of times to retry the list command within the timeout period. Some devices do not support multiple connections. Operations may fail if the device is busy with another task so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries list actions before giving up.
pcmk_monitor_action	string	monitor	An alternate command to run instead of monitor . Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific, command that implements the monitor action.

Field	Type	Default	Description
pcmk_monitor_timeout	time	60s	Specify an alternate timeout to use for monitor actions instead of stonith-timeout . Some devices need much more or much less time to complete than normal. Use this to specify an alternate, device-specific, timeout for monitor actions.
pcmk_monitor_retries	integer	2	The maximum number of times to retry the monitor command within the timeout period. Some devices do not support multiple connections. Operations may fail if the device is busy with another task so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries monitor actions before giving up.
pcmk_status_action	string	status	An alternate command to run instead of status . Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific, command that implements the status action.
pcmk_status_timeout	time	60s	Specify an alternate timeout to use for status actions instead of stonith-timeout . Some devices need much more or much less time to complete than normal. Use this to specify an alternate, device-specific, timeout for status actions.
pcmk_status_retries	integer	2	The maximum number of times to retry the status command within the timeout period. Some devices do not support multiple connections. Operations may fail if the device is busy with another task so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries status actions before giving up.
pcmk_delay_base	string	0s	Enables a base delay for fencing actions and specifies a base delay value. You can specify different values for different nodes with the pcmk_delay_base parameter. For general information about fencing delay parameters and their interactions, see Fencing delays .

Field	Type	Default	Description
pcmk_delay_max	time	0s	Enables a random delay for fencing actions and specifies the maximum delay, which is the maximum value of the combined base delay and random delay. For example, if the base delay is 3 and pcmk_delay_max is 10, the random delay will be between 3 and 10. For general information about fencing delay parameters and their interactions, see Fencing delays .
pcmk_action_limit	integer	1	The maximum number of actions that can be performed in parallel on this device. The cluster property concurrent-fencing=true needs to be configured first (this is the default value). A value of -1 is unlimited.
pcmk_on_action	string	on	For advanced use only: An alternate command to run instead of on . Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific, command that implements the on action.
pcmk_on_timeout	time	60s	For advanced use only: Specify an alternate timeout to use for on actions instead of stonith-timeout . Some devices need much more or much less time to complete than normal. Use this to specify an alternate, device-specific, timeout for on actions.
pcmk_on_retries	integer	2	For advanced use only: The maximum number of times to retry the on command within the timeout period. Some devices do not support multiple connections. Operations may fail if the device is busy with another task so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries on actions before giving up.

In addition to the properties you can set for individual fence devices, there are also cluster properties you can set that determine fencing behavior, as described in the following table.

Table 4.3. Cluster Properties that Determine Fencing Behavior

Option	Default	Description
--------	---------	-------------

Option	Default	Description
stonith-enabled	true	<p>Indicates that failed nodes and nodes with resources that cannot be stopped should be fenced. Protecting your data requires that you set this true.</p> <p>If true, or unset, the cluster will refuse to start resources unless one or more STONITH resources have been configured also.</p> <p>Red Hat only supports clusters with this value set to true.</p>
stonith-action	reboot	Action to send to fencing device. Allowed values: reboot , off . The value poweroff is also allowed, but is only used for legacy devices.
stonith-timeout	60s	How long to wait for a STONITH action to complete.
stonith-max-attempts	10	How many times fencing can fail for a target before the cluster will no longer immediately re-attempt it.
stonith-watchdog-timeout		The maximum time to wait until a node can be assumed to have been killed by the hardware watchdog. It is recommended that this value be set to twice the value of the hardware watchdog timeout. This option is needed only if watchdog-only SBD configuration is used for fencing.
concurrent-fencing	true	Allow fencing operations to be performed in parallel.

Option	Default	Description
fence-reaction	stop	<p>Determines how a cluster node should react if notified of its own fencing. A cluster node may receive notification of its own fencing if fencing is misconfigured, or if fabric fencing is in use that does not cut cluster communication. Allowed values are stop to attempt to immediately stop Pacemaker and stay stopped, or panic to attempt to immediately reboot the local node, falling back to stop on failure.</p> <p>Although the default value for this property is stop, the safest choice for this value is panic, which attempts to immediately reboot the local node. If you prefer the stop behavior, as is most likely to be the case in conjunction with fabric fencing, it is recommended that you set this explicitly.</p>
priority-fencing-delay	0 (disabled)	<p>Sets a fencing delay that allows you to configure a two-node cluster so that in a split-brain situation the node with the fewest or least important resources running is the node that gets fenced. For general information about fencing delay parameters and their interactions, see Fencing delays.</p>

For information about setting cluster properties, see https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/10/html/configuring_and_managing_cluster-properties

4.4.4. Fencing delays

In two-node clusters, simultaneous communication loss can cause nodes to fence each other, shutting down the entire cluster. Configure a fencing delay to prevent this race condition. Delays are unnecessary in larger clusters, where quorum determines fencing authority.

You can set different types of fencing delays, depending on your system requirements.

- **static fencing delays**

A static fencing delay is a fixed, predetermined delay. Setting a static delay on one node makes that node more likely to be fenced because it increases the chances that the other node will initiate fencing first after detecting lost communication. In an active/passive cluster, setting a delay on a passive node makes it more likely that the passive node will be fenced when communication breaks down. You configure a static delay by using the **pcs_delay_base** cluster property. You can set this property when a separate fence device is used for each node or when a single fence device is used for all nodes.

- **dynamic fencing delays**

A dynamic fencing delay is random. It can vary and is determined at the time fencing is needed.

You configure a random delay and specify a maximum value for the combined base delay and random delay with the **pcs_delay_max** cluster property. When the fencing delay for each node is random, which node is fenced is also random. You may find this feature useful if your cluster is configured with a single fence device for all nodes in an active/active design.

- **priority fencing delays**

A priority fencing delay is based on active resource priorities. If all resources have the same priority, the node with the fewest resources running is the node that gets fenced. In most cases, you use only one delay-related parameter, but it is possible to combine them. Combining delay-related parameters adds the priority values for the resources together to create a total delay. You configure a priority fencing delay with the **priority-fencing-delay** cluster property. You may find this feature useful in an active/active cluster design because it can make the node running the fewest resources more likely to be fenced when communication between the nodes is lost.

The **pcmk_delay_base** cluster property

Setting the **pcmk_delay_base** cluster property enables a base delay for fencing and specifies a base delay value.

When you set the **pcmk_delay_max** cluster property in addition to the **pcmk_delay_base** property, the overall delay is derived from a random delay value added to this static delay so that the sum is kept below the maximum delay. When you set **pcmk_delay_base** but do not set **pcmk_delay_max**, there is no random component to the delay and it will be the value of **pcmk_delay_base**.

You can specify different values for different nodes with the **pcmk_delay_base** parameter. This allows a single fence device to be used in a two-node cluster, with a different delay for each node. You do not need to configure two separate devices to use separate delays. To specify different values for different nodes, you map the host names to the delay value for that node using a similar syntax to **pcmk_host_map**. For example, **node1:0;node2:10s** would use no delay when fencing **node1** and a 10-second delay when fencing **node2**.

The **pcmk_delay_max** cluster property

Setting the **pcmk_delay_max** cluster property enables a random delay for fencing actions and specifies the maximum delay, which is the maximum value of the combined base delay and random delay. For example, if the base delay is 3 and **pcmk_delay_max** is 10, the random delay will be between 3 and 10.

When you set the **pcmk_delay_base** cluster property in addition to the **pcmk_delay_max** property, the overall delay is derived from a random delay value added to this static delay so that the sum is kept below the maximum delay. When you set **pcmk_delay_max** but do not set **pcmk_delay_base** there is no static component to the delay.

The **priority-fencing-delay** cluster property

Setting the **priority-fencing-delay** cluster property allows you to configure a two-node cluster so that in a split-brain situation the node with the fewest or least important resources running is the node that gets fenced.

The **priority-fencing-delay** property can be set to a time duration. The default value for this property is 0 (disabled). If this property is set to a non-zero value, and the priority meta-attribute is configured for at least one resource, then in a split-brain situation the node with the highest combined priority of all resources running on it will be more likely to remain operational. For example, if you set **pcs resource defaults update priority=1** and **pcs property set priority-fencing-delay=15s** and no other priorities

are set, then the node running the most resources will be more likely to remain operational because the other node will wait 15 seconds before initiating fencing. If a particular resource is more important than the rest, you can give it a higher priority.

The node running the promoted role of a promotable clone gets an extra 1 point if a priority has been configured for that clone.

Interaction of fencing delays

Setting more than one type of fencing delay yields the following results:

- Any delay set with the **priority-fencing-delay** property is added to any delay from the **pcmk_delay_base** and **pcmk_delay_max** fence device properties. This behavior allows some delay when both nodes have equal priority, or both nodes need to be fenced for some reason other than node loss, as when **on-fail=fencing** is set for a resource monitor operation. When setting these delays in combination, set the **priority-fencing-delay** property to a value that is significantly greater than the maximum delay from **pcmk_delay_base** and **pcmk_delay_max** to be sure the prioritized node is preferred. Setting this property to twice this value is always safe.
- Only fencing scheduled by Pacemaker itself observes fencing delays. Fencing scheduled by external code such as **dlm_controld** and fencing implemented by the **pcs stonith fence** command do not provide the necessary information to the fence device.
- Some individual fence agents implement a delay parameter, with a name determined by the agent and which is independent of delays configured with a **pcmk_delay_*** property. If both of these delays are configured, they are added together and would generally not be used in conjunction.

4.4.5. Testing a fence device

Validate fence devices to ensure the cluster can successfully recover from node failures and prevent data corruption. A complete testing strategy involves verifying network connectivity, executing the fence agent script directly, triggering the fence action through the cluster manager, and simulating a physical node failure.



NOTE

When a Pacemaker cluster node or Pacemaker remote node is fenced a hard kill should occur and not a graceful shutdown of the operating system. If a graceful shutdown occurs when your system fences a node, disable ACPI soft-off in the **/etc/systemd/logind.conf** file so that your system ignores any power-button-pressed signal. For instructions on disabling ACPI soft-off in the **logind.conf** file, see [Disabling ACPI soft-off in the logind.conf file](#)

Use the following procedure to test a fence device.

Procedure

1. Use SSH, Telnet, HTTP, or whatever remote protocol is used to connect to the device to manually log in and test the fence device or see what output is given. For example, if you will be configuring fencing for an IPMI-enabled device, then try to log in remotely with **ipmitool**. Take note of the options used when logging in manually because those options might be needed when using the fencing agent.

If you are unable to log in to the fence device, verify that the device is pingable, there is nothing such as a firewall configuration that is preventing access to the fence device, remote access is enabled on the fencing device, and the credentials are correct.

2. Run the fence agent manually, using the fence agent script. This does not require that the cluster services are running, so you can perform this step before the device is configured in the cluster. This can ensure that the fence device is responding properly before proceeding.



NOTE

These examples use the **fence_ipmilan** fence agent script for an iLO device. The actual fence agent you will use and the command that calls that agent will depend on your server hardware. You should consult the man page for the fence agent you are using to determine which options to specify. You will usually need to know the login and password for the fence device and other information related to the fence device.

The following example shows the format you would use to run the **fence_ipmilan** fence agent script with **-o status** parameter to check the status of the fence device interface on another node without actually fencing it. This allows you to test the device and get it working before attempting to reboot the node. When running this command, you specify the name and password of an iLO user that has power on and off permissions for the iLO device.

```
# fence_ipmilan -a ipaddress -l username -p password -o status
```

The following example shows the format you would use to run the **fence_ipmilan** fence agent script with the **-o reboot** parameter. Running this command on one node reboots the node managed by this iLO device.

```
# fence_ipmilan -a ipaddress -l username -p password -o reboot
```

If the fence agent failed to properly do a status, off, on, or reboot action, you should check the hardware, the configuration of the fence device, and the syntax of your commands. In addition, you can run the fence agent script with the debug output enabled. The debug output is useful for some fencing agents to see where in the sequence of events the fencing agent script is failing when logging into the fence device.

```
# fence_ipmilan -a ipaddress -l username -p password -o status -D /tmp/$(hostname)-fence_agent.debug
```

When diagnosing a failure that has occurred, you should ensure that the options you specified when manually logging in to the fence device are identical to what you passed on to the fence agent with the fence agent script.

For fence agents that support an encrypted connection, you may see an error due to certificate validation failing, requiring that you trust the host or that you use the fence agent's **ssl-insecure** parameter. Similarly, if SSL/TLS is disabled on the target device, you may need to account for this when setting the SSL parameters for the fence agent.



NOTE

If the fence agent that is being tested is a **fence_drac**, **fence_ilo**, or some other fencing agent for a systems management device that continues to fail, then fall back to trying **fence_ipmilan**. Most systems management cards support IPMI remote login and the only supported fencing agent is **fence_ipmilan**.

3. Once the fence device has been configured in the cluster with the same options that worked manually and the cluster has been started, test fencing with the **pcs stonith fence** command from any node (or even multiple times from different nodes), as in the following example. The **pcs stonith fence** command reads the cluster configuration from the CIB and calls the fence agent as configured to execute the fence action. This verifies that the cluster configuration is correct.

```
# pcs stonith fence node_name
```

If the **pcs stonith fence** command works properly, that means the fencing configuration for the cluster should work when a fence event occurs. If the command fails, it means that cluster management cannot invoke the fence device through the configuration it has retrieved. Check for the following issues and update your cluster configuration as needed.

- Check your fence configuration. For example, if you have used a host map you should ensure that the system can find the node using the host name you have provided.
- Check whether the password and user name for the device include any special characters that could be misinterpreted by the bash shell. Making sure that you enter passwords and user names surrounded by quotation marks could address this issue.
- Check whether you can connect to the device using the exact IP address or host name you specified in the **pcs stonith** command. For example, if you give the host name in the stonith command but test by using the IP address, that is not a valid test.
- If the protocol that your fence device uses is accessible to you, use that protocol to try to connect to the device. For example many agents use ssh or telnet. You should try to connect to the device with the credentials you provided when configuring the device, to see if you get a valid prompt and can log in to the device.

If you determine that all your parameters are appropriate but you still have trouble connecting to your fence device, you can check the logging on the fence device itself, if the device provides that, which will show if the user has connected and what command the user issued. You can also search through the **/var/log/messages** file for instances of stonith and error, which could give some idea of what is transpiring, but some agents can provide additional information.

4. Once the fence device tests are working and the cluster is up and running, test an actual failure. To do this, take an action in the cluster that should initiate a token loss.
 - Take down a network. How you take a network depends on your specific configuration. In many cases, you can physically pull the network or power cables out of the host. For information about simulating a network failure, see the Red Hat Knowledgebase solution [What is the proper way to simulate a network failure on a RHEL Cluster?](#) .

**NOTE**

Disabling the network interface on the local host rather than physically disconnecting the network or power cables is not recommended as a test of fencing because it does not accurately simulate a typical real-world failure.

- Block corosync traffic both inbound and outbound using the local firewall. The following example blocks corosync, assuming the default corosync port is used, **firewalld** is used as the local firewall, and the network interface used by corosync is in the default firewall zone:

```
# firewall-cmd --direct --add-rule ipv4 filter OUTPUT 2 -p udp --dport=5405 -j DROP
# firewall-cmd --add-rich-rule='rule family="ipv4" port port="5405" protocol="udp" drop
```

- Simulate a crash and panic your machine with **sysrq-trigger**. Note, however, that triggering a kernel panic can cause data loss; it is recommended that you disable your cluster resources first.

```
# echo c > /proc/sysrq-trigger
```

4.4.6. Configuring fencing levels

Pacemaker supports fencing nodes with multiple devices through a feature called fencing topologies. To implement topologies, create the individual devices as you normally would and then define one or more fencing levels in the fencing topology section in the configuration.

Pacemaker processes fencing levels as follows:

- Each level is attempted in ascending numeric order, starting at 1.
- If a device fails, processing terminates for the current level. No further devices in that level are exercised and the next level is attempted instead.
- If all devices are successfully fenced, then that level has succeeded and no other levels are tried.
- The operation is finished when a level has passed (success), or all levels have been attempted (failed).

Use the following command to add a fencing level to a node. The devices are given as a comma-separated list of **stonith** ids, which are attempted for the node at that level.

```
pcs stonith level add level node devices
```

The following example sets up fence levels so that if the device **my_ilo** fails and is unable to fence the node, then Pacemaker attempts to use the device **my_apc**.

Prerequisites

- You have configured an ilo fence device called **my_ilo** for node **rh7-2**.
- You have configured an apc fence device called **my_apc** for node **rh7-2**.

Procedure

1. Add a fencing level of 1 for fence device **my_ilo** on node **rh7-2**.

```
# pcs stonith level add 1 rh7-2 my_ilo
```

2. Add a fencing level of 2 for fence device **my_apc** on node **rh7-2**.

```
# pcs stonith level add 2 rh7-2 my_apc
```

3. List the currently configured fencing levels.

```
# pcs stonith level
Node: rh7-2
Level 1 - my_ilo
Level 2 - my_apc
```

Additional resources

[Configuring node-specific values using node attributes](#) .

4.4.7. Removing a fence level

You can remove the fence level for the specified node and device. If no nodes or devices are specified then the fence level you specify is removed from all nodes.

Procedure

- Remove the fence level for the specified node and device:

```
# pcs stonith level remove level [node_id] [stonith_id] ... [stonith_id]
```

4.4.8. Clearing fence levels

You can clear the fence levels on the specified node or stonith id. If you do not specify a node or stonith id, all fence levels are cleared.

Procedure

- Clear the fence levels on the specified node or stonith id:

```
# pcs stonith level clear [node]|stonith_id(s)]
```

- If you specify more than one stonith id, they must be separated by a comma and no spaces, as in the following example.

```
# pcs stonith level clear dev_a,dev_b
```

4.4.9. Verifying nodes and devices in fence levels

You can verify that all fence devices and nodes specified in fence levels exist.

Procedure

- Use the following command to verify that all fence devices and nodes specified in fence levels exist:

```
# pcs stonith level verify
```

4.4.10. Specifying nodes in fencing topology

You can specify nodes in fencing topology by a regular expression applied on a node name and by a node attribute and its value.

Procedure

- The following commands configure nodes **node1**, **node2**, and **node3** to use fence devices **apc1** and **apc2**, and nodes **node4**, **node5**, and **node6** to use fence devices **apc3** and **apc4**:

```
# pcs stonith level add 1 "regexp%node[1-3]" apc1,apc2
# pcs stonith level add 1 "regexp%node[4-6]" apc3,apc4
```

- The following commands yield the same results by using node attribute matching:

```
# pcs node attribute node1 rack=1
# pcs node attribute node2 rack=1
# pcs node attribute node3 rack=1
# pcs node attribute node4 rack=2
# pcs node attribute node5 rack=2
# pcs node attribute node6 rack=2
# pcs stonith level add 1 attrib%rack=1 apc1,apc2
# pcs stonith level add 1 attrib%rack=2 apc3,apc4
```

4.4.11. Configuring fencing for redundant power supplies

When configuring fencing for redundant power supplies, the cluster must ensure that when attempting to reboot a host, both power supplies are turned off before either power supply is turned back on.

If the node never completely loses power, the node may not release its resources. This opens up the possibility of nodes accessing these resources simultaneously and corrupting them.

You need to define each device only once and to specify that both are required to fence the node.

Procedure

1. Create the first fence device.

```
# pcs stonith create apc1 fence_apc_snmp ipaddr=apc1.example.com login=user
passwd='7a4D#1j!pz864'
pcmk_host_map="node1.example.com:1;node2.example.com:2"
```

2. Create the second fence device.

```
# pcs stonith create apc2 fence_apc_snmp ipaddr=apc2.example.com login=user
passwd='7a4D#1j!pz864'
pcmk_host_map="node1.example.com:1;node2.example.com:2"
```

3. Specify that both devices are required to fence the node.

```
# pcs stonith level add 1 node1.example.com apc1,apc2
# pcs stonith level add 1 node2.example.com apc1,apc2
```

4.4.12. Administering fence devices

The **pcs** command-line interface provides a variety of commands you can use to administer your fence devices after you have configured them.

4.4.12.1. Displaying configured fence devices

The following command shows all currently configured fence devices. If a *stonith_id* is specified, the command shows the options for that configured fencing device only. If the **--full** option is specified, all configured fencing options are displayed.

```
pcs stonith config [stonith_id] [--full]
```

4.4.12.2. Exporting fence devices as pcs commands

You can display the **pcs** commands that can be used to re-create configured fence devices on a different system using the **--output-format=cmd** option of the **pcs stonith config** command.

The following commands create a **fence_apc_snmp** fence device and display the **pcs** command you can use to re-create the device.

```
# pcs stonith create myapc fence_apc_snmp ip="zapc.example.com"
pcmk_host_map="z1.example.com:1;z2.example.com:2" username="apc" password="apc"
# pcs stonith config --output-format=cmd
Warning: Only 'text' output format is supported for stonith levels
pcs stonith create --no-default-ops --force -- myapc fence_apc_snmp \
  ip=zapc.example.com password=apc 'pcmk_host_map=z1.example.com:1;z2.example.com:2'
username=apc \
  op \
  monitor interval=60s id=myapc-monitor-interval-60s
```

4.4.12.3. Exporting fence level configuration

The **pcs stonith config** and the **pcs stonith level config** commands support the **--output-format=** option to export the fencing level configuration in JSON format and as **pcs** commands.

- Specifying **--output-format=cmd** displays the **pcs** commands created from the current cluster configuration that configure fencing levels. You can use these commands to re-create configured fencing levels on a different system.
- Specifying **--output-format=json** displays the fencing level configuration in JSON format, which is suitable for machine parsing.

4.4.12.4. Modifying and deleting fence devices

Modify or add options to a currently configured fencing device with the following command.

```
pcs stonith update stonith_id [stonith_device_options]
```

Updating a SCSI fencing device with the **pcs stonith update** command causes a restart of all resources running on the same node where the fencing resource was running. You can use either version of the following command to update SCSI devices without causing a restart of other cluster resources. SCSI fencing devices can be configured as multipath devices.

```
pcs stonith update-scsi-devices stonith_id set device-path1 device-path2
pcs stonith update-scsi-devices stonith_id add device-path1 remove device-path2
```

Use the following command to remove a fencing device from the current configuration.

```
pcs stonith delete stonith_id
```

4.4.12.5. Manually fencing a cluster node

You can fence a node manually with the following command. If you specify the **--off** option this will use the **off** API call to stonith which will turn the node off instead of rebooting it.

```
pcs stonith fence node [--off]
```

In a situation where no fence device is able to fence a node even if it is no longer active, the cluster may not be able to recover the resources on the node. If this occurs, after manually ensuring that the node is powered down you can enter the following command to confirm to the cluster that the node is powered down and free its resources for recovery.



WARNING

If the node you specify is not actually off, but running the cluster software or services normally controlled by the cluster, data corruption and cluster failure occurs.

```
pcs stonith confirm node
```

4.4.12.6. Disabling a fence device

To disable a fencing device, run the **pcs stonith disable** command.

The following command disables the fence device **myapc**.

```
# pcs stonith disable myapc
```

4.4.12.7. Preventing a node from using a fencing device

To prevent a specific node from using a fencing device, you can configure location constraints for the fencing resource.

The following example prevents fence device **node1-ipmi** from running on **node1**.

```
# pcs constraint location node1-ipmi avoids node1
```

4.4.13. Configuring ACPI for use with integrated fence devices

If your cluster uses integrated fence devices, you must configure ACPI (Advanced Configuration and Power Interface) to ensure immediate and complete fencing.

If a cluster node is configured to be fenced by an integrated fence device, disable ACPI Soft-Off for that node. Disabling ACPI Soft-Off allows an integrated fence device to turn off a node immediately and completely rather than attempting a clean shutdown (for example, **shutdown -h now**). Otherwise, if ACPI Soft-Off is enabled, an integrated fence device can take four or more seconds to turn off a node (see the note that follows). In addition, if ACPI Soft-Off is enabled and a node panics or freezes during shutdown, an integrated fence device may not be able to turn off the node. Under those circumstances, fencing is delayed or unsuccessful. Consequently, when a node is fenced with an integrated fence device and ACPI Soft-Off is enabled, a cluster recovers slowly or requires administrative intervention to recover.



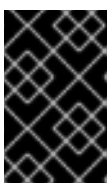
NOTE

The amount of time required to fence a node depends on the integrated fence device used. Some integrated fence devices perform the equivalent of pressing and holding the power button; therefore, the fence device turns off the node in four to five seconds. Other integrated fence devices perform the equivalent of pressing the power button momentarily, relying on the operating system to turn off the node; therefore, the fence device turns off the node in a time span much longer than four to five seconds.

- The preferred way to disable ACPI Soft-Off is to change the BIOS setting to "instant-off" or an equivalent setting that turns off the node without delay, as described in [Disabling ACPI Soft-Off with the Bios](#).

Disabling ACPI Soft-Off with the BIOS may not be possible with some systems. If disabling ACPI Soft-Off with the BIOS is not satisfactory for your cluster, you can disable ACPI Soft-Off with one of the following alternate methods:

- Setting **HandlePowerKey=ignore** in the **/etc/systemd/logind.conf** file and verifying that the node turns off immediately when fenced, as described in [Disabling ACPI soft-off in the logind.conf file](#). This is the first alternate method of disabling ACPI Soft-Off.
- Appending **acpi=off** to the kernel boot command line, as described in [Disabling ACPI completely in the GRUB 2 file](#). This is the second alternate method of disabling ACPI Soft-Off, if the preferred or the first alternate method is not available.



IMPORTANT

This method completely disables ACPI; some computers do not boot correctly if ACPI is completely disabled. Use this method *only* if the other methods are not effective for your cluster.

4.4.13.1. Disabling ACPI Soft-Off with the BIOS

You can disable ACPI Soft-Off by configuring the BIOS of each cluster node.



NOTE

The procedure for disabling ACPI Soft-Off with the BIOS may differ among server systems. You should verify this procedure with your hardware documentation.

Procedure

- 1. Reboot the node and start the **BIOS CMOS Setup Utility** program.
- 2. Navigate to the Power menu (or equivalent power management menu).
- 3. At the Power menu, set the **Soft-Off by PWR-BTTN** function (or equivalent) to **Instant-Off** (or the equivalent setting that turns off the node by means of the power button without delay). The **BIOS CMOS Setup Utility** example below shows a Power menu with **ACPI Function** set to **Enabled** and **Soft-Off by PWR-BTTN** set to **Instant-Off**.



NOTE

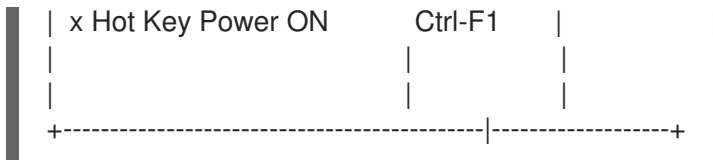
The equivalents to **ACPI Function**, **Soft-Off by PWR-BTTN**, and **Instant-Off** may vary among computers. However, the objective of this procedure is to configure the BIOS so that the computer is turned off by means of the power button without delay.

- 4. Exit the **BIOS CMOS Setup Utility** program, saving the BIOS configuration.
- 5. Verify that the node turns off immediately when fenced. For information about testing a fence device, see [Testing a fence device](#).

BIOS CMOS Setup Utility:

`Soft-Off by PWR-BTTN` set to
`Instant-Off`

+-----+-----+			
	ACPI Function	[Enabled]	Item Help
	ACPI Suspend Type	[S1(POS)]	-----
	x Run VGABIOS if S3 Resume	Auto	Menu Level *
	Suspend Mode	[Disabled]	
	HDD Power Down	[Disabled]	
	Soft-Off by PWR-BTTN	[Instant-Off]	
	CPU THRM-Throttling	[50.0%]	
	Wake-Up by PCI card	[Enabled]	
	Power On by Ring	[Enabled]	
	Wake Up On LAN	[Enabled]	
	x USB KB Wake-Up From S3	Disabled	
	Resume by Alarm	[Disabled]	
	x Date(of Month) Alarm	0	
	x Time(hh:mm:ss) Alarm	0 : 0 :	
	POWER ON Function	[BUTTON ONLY]	
	x KB Power ON Password	Enter	



This example shows **ACPI Function** set to **Enabled**, and **Soft-Off by PWR-BTTN** set to **Instant-Off**.

4.4.13.2. Disabling ACPI Soft-Off in the logind.conf file

You can disable power-key handing in the `/etc/systemd/logind.conf` file.

Procedure

1. Define the following configuration in the `/etc/systemd/logind.conf` file:

```
HandlePowerKey=ignore
```

2. Restart the **systemd-logind** service:

```
# systemctl restart systemd-logind.service
```

Verification

1. Verify that the node turns off immediately when fenced. For information about testing a fence device, see [Testing a fence device](#).

4.4.13.3. Disabling ACPI completely in the GRUB 2 file

You can disable ACPI Soft-Off by appending **acpi=off** to the GRUB menu entry for a kernel.



IMPORTANT

This method completely disables ACPI; some computers do not boot correctly if ACPI is completely disabled. Use this method *only* if the other methods are not effective for your cluster.

Procedure

1. Use the **--args** option in combination with the **--update-kernel** option of the **grubby** tool to change the **grub.cfg** file of each cluster node as follows:

```
# grubby --args=acpi=off --update-kernel=ALL
```

2. Reboot the node.

Verification

1. Verify that the node turns off immediately when fenced. For information about testing a fence device, see [Testing a fence device](#)

4.5. SETTING UP IP ADDRESS RESOURCES ON AWS

To manage network access for cluster resources during failover in a high availability (HA) cluster, you can configure IP address resources. The Red Hat High Availability Add-On offers resource agents for different Amazon Web Services (AWS) IP address types.

This includes internet-exposed addresses, single-zone addresses, and multi-zone addresses.

- Exposed to the internet: Use the **awseip** network resource.
- Limited to a single AWS Availability Zone (AZ): Use the **awsvip** and **IPAddr2** network resources.
- Reassigns to many AWS AZs within the same AWS region: Use the **aws-vpc-move-ip** network resource.



NOTE

If the HA cluster does not manage any IP addresses, the resource agents for managing virtual IP addresses on AWS are not required. If you need further guidance for your specific deployment, consult with AWS.

4.5.1. Creating an IP address resource to manage an IP address exposed to the internet

To ensure that high-availability (HA) clients can access a Red Hat Enterprise Linux (RHEL) node that uses public-facing internet connections, configure an *AWS Secondary Elastic IP Address* (**awseip**) resource to use an elastic IP address.

Prerequisites

- You have a configured cluster.
- Your cluster nodes must have access to the RHEL HA repositories. For details, see [Installing the High Availability packages and agents](#).
- You have set up the AWS CLI2. For details, see [Installing AWSCLI2](#).

Procedure

1. Add the two resources to the [same group](#) that you have already created to enforce **order** and **colocation** constraints.
2. Install the **resource-agents** package:

```
# dnf install resource-agents
```

3. Create an elastic IP address:

```
[root@ip-10-0-0-48 ~]# aws ec2 allocate-address --domain vpc --output text
eipalloc-4c4a2c45 vpc 35.169.153.122
```

4. Optional: Display the description of **awseip**. This shows the options and default operations for this agent.

```
# pcs resource describe awseip
```

5. Create the Secondary Elastic IP address resource with the allocated IP address in the 2nd step:

```
# pcs resource create <resource_id> awseip elastic_ip=<elastic_ip_address>
allocation_id=<elastic_ip_association_id> --group networking-group
```

Example:

```
# pcs resource create elastic awseip elastic_ip=35.169.153.122 allocation_id=eipalloc-
4c4a2c45 --group networking-group
```

Verification

1. Verify the cluster status to ensure resources are available:

```
[root@ip-10-0-0-58 ~]# pcs status

Cluster name: newcluster
Stack: corosync
Current DC: ip-10-0-0-58 (version 1.1.18-11.el7-2b07d5c5a9) - partition with quorum
Last updated: Mon Mar  5 16:27:55 2018
Last change: Mon Mar  5 15:57:51 2018 by root via cibadmin on ip-10-0-0-46

3 nodes configured
4 resources configured

Online: [ ip-10-0-0-46 ip-10-0-0-48 ip-10-0-0-58 ]

Full list of resources:

clusterfence (stonith:fence_aws): Started ip-10-0-0-46
Resource Group: networking-group
  vip (ocf::heartbeat:IPaddr2): Started ip-10-0-0-48
  elastic (ocf::heartbeat:awseip): Started ip-10-0-0-48

Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

In this example, **newcluster** is an active cluster where resources such as **vip** and **elastic** are part of the **networking-group** resource group.

2. Launch an SSH session from your local workstation to the elastic IP address that you have already created:

```
$ ssh -l ec2-user -i ~/.ssh/cluster-admin.pem 35.169.153.122
```

3. Verify that the SSH connected host is same as the host with the elastic resources.

4.5.2. Creating an IP address resource to manage a private IP address limited to a single AWS Availability Zone

To use a private IP address that is limited to a single Amazon Web Services (AWS) Availability Zone (AZ) in a Red Hat Enterprise Linux (RHEL) high-availability (HA) cluster, you can configure an *AWS Secondary Private IP Address* (**awsvip**) resource.

Prerequisites

- You have a configured cluster.
- Your cluster nodes have access to the RHEL HA repositories. For details, see [Installing the High Availability packages and agents](#).
- You have set up the AWS CLI. For instructions, see [Installing AWSCLI2](#).

Procedure

1. Install the **resource-agents** package.

```
# dnf install resource-agents
```

2. Optional: View the options and default operations for **awsvip**:

```
# pcs resource describe awsvip
```

3. Create a Secondary Private IP address with an unused private IP address in the **VPC CIDR** block:

```
[root@ip-10-0-0-48 ~]# pcs resource create privip awsvip secondary_private_ip=10.0.0.68 --group networking-group
```

Here, secondary private IP address is a part of gets included in a resource group

4. Create a virtual IP resource with the **vip** resource ID and the **networking-group** group name:

```
root@ip-10-0-0-48 ~]# pcs resource create vip IPAddr2 ip=10.0.0.68 --group networking-group
```

This is a Virtual Private Cloud (VPC) IP address that maps from the fence node to the failover node, masking the failure of the fence node within the subnet. Ensure that the virtual IP belongs to the same resource group as the Secondary Private IP address you created in the last step.

Verification

- Verify the cluster status to ensure resources are available:

```
[root@ip-10-0-0-48 ~]# pcs status
Cluster name: newcluster
Stack: corosync
Current DC: ip-10-0-0-46 (version 1.1.18-11.el7-2b07d5c5a9) - partition with quorum
Last updated: Fri Mar  2 22:34:24 2018
Last change: Fri Mar  2 22:14:58 2018 by root via cibadmin on ip-10-0-0-46

3 nodes configured
3 resources configured
```

```
Online: [ ip-10-0-0-46 ip-10-0-0-48 ip-10-0-0-58 ]
```

Full list of resources:

```
clusterfence (stonith:fence_aws): Started ip-10-0-0-46
```

```
Resource Group: networking-group
```

```
privip (ocf::heartbeat:awsvip): Started ip-10-0-0-48
```

```
vip (ocf::heartbeat:IPAddr2): Started ip-10-0-0-58
```

Daemon Status:

```
corosync: active/disabled
```

```
pacemaker: active/disabled
```

```
pcsd: active/enabled
```

In this example, **newcluster** is an active cluster where resources such as **vip** and **elastic** are part of the **networking-group** resource group.

4.5.3. Creating an IP address resource to manage an IP address that can move across multiple AWS Availability Zones

To use an elastic IP address on Amazon Web Services (AWS), you can configure a Red Hat Enterprise Linux (RHEL) Overlay IP (**aws-vpc-move-ip**) resource agent. With **aws-vpc-move-ip**, you can move a RHEL node within a single region of AWS across multiple availability zones (AZ) to ensure high-availability (HA) clients.

Prerequisites

- You have an already configured cluster.
- Your cluster nodes have access to the RHEL HA repositories. For more information, see [Installing the High Availability packages and agents](#).
- You have set up the AWS CLI. For instructions, see [Installing AWSCLI2](#).
- You have configured an Identity and Access Management (IAM) user on your cluster with the following permissions:
 - Modify routing tables
 - Create security groups
 - Create IAM policies and roles

Procedure

1. Install the **resource-agents** package:

```
# dnf install resource-agents
```

2. Optional: View the options and default operations for **awsvip**:

```
# pcs resource describe aws-vpc-move-ip
```

3. Set up an **OverlayIPAgent** IAM policy for the IAM user.

- a. In the AWS console, navigate to **Services → IAM → Policies → Create OverlayIPAgent Policy**
- b. Input the following configuration, and change the `<region>`, `<account_id>`, and `<cluster_route_table_id>` values to correspond with your cluster.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1424870324000",
      "Effect": "Allow",
      "Action": "ec2:DescribeRouteTables",
      "Resource": "*"
    },
    {
      "Sid": "Stmt1424860166260",
      "Action": [
        "ec2:CreateRoute",
        "ec2:ReplaceRoute"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ec2:_{<region>}:_{<account_id>}:route-
table/_<cluster_route_table_id>_"
    }
  ]
}
```

4. In the AWS console, disable the **Source/Destination Check** function on all nodes in the cluster. To do this, right-click each node → **Networking → Change Source/Destination Checks**. In the pop-up message that is displayed, click **Yes, Disable**.
5. Create a route table for the cluster. To do so, use the following command on one node in the cluster:

```
# aws ec2 create-route --route-table-id <cluster_route_table_id> --destination-cidr-block
<new_cidr_block_ip/net_mask> --instance-id <cluster_node_id>
```

In the command, replace values as follows:

- **ClusterRouteTableID**: The route table ID for the existing cluster Virtual Private Cloud (VPC) route table.
 - **NewCIDRblockIP**: A new IP address and netmask outside of the VPC classless inter-domain routing (CIDR) block. For example, if the VPC CIDR block is **172.31.0.0/16**, the new IP address or netmask can be **192.168.0.15/32**.
 - **ClusterNodeID**: The instance ID for another node in the cluster.
6. On one of the nodes in the cluster, create a **aws-vpc-move-ip** resource that uses a free IP address that is accessible to the client. The following example creates a resource named **vpcip** that uses IP **192.168.0.15**.

```
# pcs resource create vpcip aws-vpc-move-ip ip=192.168.0.15 interface=eth0
routing_table=<cluster_route_table_id>
```

7. On all nodes in the cluster, edit the `/etc/hosts/` file, and add a line with the IP address of the newly created resource. For example:

```
192.168.0.15 vpcip
```

Verification

1. Test the failover ability of the new **aws-vpc-move-ip** resource:

```
# pcs resource move vpcip
```

2. If the failover succeeded, remove the automatically created constraint after the move of the **vpcip** resource:

```
# pcs resource clear vpcip
```

Additional resources

- [Configure the OverlayIP Resource Agent \(Red Hat Knowledgebase\)](#)
- [IAM users](#)

4.5.4. Additional resources

- [High Availability Add-On overview](#)

4.6. CONFIGURING SHARED BLOCK STORAGE

To create extra storage resources, you can use Amazon Elastic Block Storage (EBS) Multi-Attach volumes to configure shared block storage for a Red Hat High Availability cluster.

Prerequisites

- You have created a [Red Hat account](#).
- You have [signed up and set up an AWS account](#).
- You have 3 RHEL instances running in a cluster as a three-node cluster with a 1 TB shared disk.
- You have access to an [AWS Nitro System-based Amazon EC2 instance](#).

Procedure

1. Create a shared block volume in the **us-east-1a** availability zone:

```
$ aws ec2 create-volume --availability-zone <us_east_1a> --no-encrypted --size <1024> --
volume-type <io1> --iops <51200> --multi-attach-enabled

{
  "AvailabilityZone": "us-east-1a",
  "CreateTime": "2020-08-27T19:16:42.000Z",
  "Encrypted": false,
  "Size": 1024,
```

```

    "SnapshotId": "",
    "State": "creating",
    "VolumeId": "vol-042a5652867304f09",
    "Iops": 51200,
    "Tags": [ ],
    "VolumeType": "io1"
  }

```

2. For each instance in your cluster, attach a shared block volume. Use your **<instance_id>** and **<volume_id>**, for example **vol-042a5652867304f09** to **instance i-0eb803361c2c887f2**:

```

$ aws ec2 attach-volume --device /dev/xvdd --instance-id i-0eb803361c2c887f2 --volume-id vol-042a5652867304f09

{
  "AttachTime": "2020-08-27T19:26:16.086Z",
  "Device": "/dev/xvdd",
  "InstanceId": "i-0eb803361c2c887f2",
  "State": "attaching",
  "VolumeId": "vol-042a5652867304f09"
}

```

Verification

1. Verify the block device is available by using the **ssh** command with your instance IP **198.51.100.3** for each instance in the cluster:

```

# ssh 198.51.100.3 "hostname ; lsblk -d | grep ' 1T '"

node a
nvme2n1 259:1   0   1T  0 disk

```

2. Use the **ssh** command to verify that each instance in your cluster uses the same shared disk:

```

# ssh 198.51.100.3 "hostname ; lsblk -d | grep ' 1T ' | awk '{print \$1}' | xargs -i udevadm info -
-query=all --name=/dev/{ } | grep '^E: ID_SERIAL='"

nodea
E: ID_SERIAL=Amazon Elastic Block Store_vol0fa5342e7aedef09f7

```

CHAPTER 5. CONFIGURING RHEL ON AWS WITH SECURE BOOT

To provide security at boot time for Red Hat Enterprise Linux (RHEL) instances on Amazon Web Services (AWS), you can configure Secure Boot. During startup, Secure Boot verifies digital signatures of the boot loader and other components. Then, Secure Boot allows only trusted programs to load and blocks unauthorized programs.

You can set up RHEL instances by using either AWS Marketplace RHEL Amazon Machine Image (AMI) or custom RHEL AMI with Secure Boot enabled.

5.1. TYPES OF RHEL AMIS ON AWS

AWS Marketplace RHEL AMI

The AWS Marketplace offers pre-configured Red Hat Enterprise Linux (RHEL) Amazon Machine Image (AMI) designed for specific use cases, such as data processing, system management, and web development. These ready-to-use images help reduce setup time by minimizing the manual installation and configuration required for operating systems and software packages.

Custom RHEL AMI

A custom RHEL AMI offers flexibility to customers and organizations to build and deploy tailored environments that meet specific application and workflow requirements. By creating custom RHEL AMI, you can use RHEL instances that are pre-installed with necessary tools, configurations, and security policies. This customization aims at greater control over the infrastructure.

5.2. UNDERSTANDING SECURE BOOT FOR RHEL ON CLOUD

Secure Boot is a Unified Extensible Firmware Interface (UEFI) feature that verifies digital signatures of boot components, such as boot loader and kernel, against trusted keys stored in hardware. Secure Boot prevents unauthorized or tampered software from running during boot, protecting your system from malicious code.

If Secure Boot detects any tampered components or components signed by untrusted entities, it aborts the boot process. Secure Boot plays a critical role in configuring a Confidential Virtual Machine (CVM) by ensuring that only trusted entities participate in the boot chain. It authenticates access to specific device paths through defined interfaces, enforces the use of the latest configuration, and permanently overwrites earlier configurations. When the Red Hat Enterprise Linux (RHEL) kernel boots with Secure Boot enabled, it enters the **lockdown** mode, allowing only kernel modules signed by a trusted vendor to load. Therefore, Secure Boot strengthens the security of the operating system boot sequence.

5.2.1. Components of Secure Boot

The Secure Boot mechanism consists of firmware, signature databases, cryptographic keys, boot loader, hardware modules, and the operating system. The following are the components of the UEFI trusted variables:

- Key Exchange Key database (KEK): An exchange of public keys to establish trust between the RHEL operating system and the VM firmware. You can also update Allowed Signature database (**db**) and Forbidden Signature database (**dbx**) by using these keys.
- Platform Key database (PK): A self-signed single-key database to establish trust between the VM firmware and the cloud platform. The PK also updates the KEK database.
- Allowed Signature database (**db**): A database that maintains a list of certificates or binary

hashes to check whether the binary file can boot on the system. Additionally, all certificates from **db** are imported to the **.platform** keyring of the RHEL kernel. With this feature, you can add and load signed third party kernel modules in the **lockdown** mode.

- Forbidden Signature database (**dbx**): A database that maintains a list of certificates or binary hashes that are not allowed to boot on the system.



NOTE

Binary files check against the **dbx** database and the Secure Boot Advanced Targeting (SBAT) mechanism. With SBAT, you can revoke older versions of specific binaries by keeping the certificate that has signed binaries as valid.

5.2.2. Stages of Secure Boot for RHEL on Cloud

When a RHEL instance boots in the Unified Kernel Image (UKI) mode and with Secure Boot enabled, the RHEL instance interacts with the cloud service infrastructure in the following sequence:

1. *Initialization*: When a RHEL instance boots, the cloud-hosted firmware initially boots and implements the Secure Boot mechanism.
2. *Variable store initialization*: The firmware initializes UEFI variables from a variable store, a dedicated storage area for information that firmware needs to manage for the boot process and runtime operations. When the RHEL instance boots for the first time, the store initializes from default values associated with the VM image.
3. *Boot loader*: When booted, the firmware loads the first stage boot loader. For the RHEL instance in a x86 UEFI environment, the first stage boot loader is shim. The shim boot loader authenticates and loads the next stage of the boot process and acts as a bridge between UEFI and GRUB.
 - a. The shim x86 binary in RHEL is currently signed by the **Microsoft Corporation UEFI CA 2011** Microsoft certificate so that the RHEL instance can boot in the Secure Boot enabled mode on various hardware and virtualized platforms where the Allowed Signature database (**db**) has the default Microsoft certificates.
 - b. The shim binary extends the list of trusted certificates with Red Hat Secure Boot CA and optionally, with Machine Owner Key (**MOK**).
4. *UKI*: The shim binary loads the RHEL UKI (the **kernel-uki-virt** package). The corresponding certificate, **Red Hat Secure Boot Signing 504** on the x86_64 architecture, signs the UKI. You can find this certificate in the **redhat-sb-certs** package. Red Hat Secure Boot CA signs this certificate, so the check succeeds.
5. *UKI add-ons*: When you use the UKI **cmdline** extensions, the RHEL kernel actively checks their signatures against **db**, **MOK**, and certificates shipped with shim. This process ensures that either the operating system vendor RHEL or a user has signed the extensions.

When the RHEL kernel boots in the Secure Boot mode, it enters **lockdown** mode. After entering **lockdown**, the RHEL kernel adds the **db** keys to the **.platform** keyring and the **MOK** keys to the **.machine** keyring. During the kernel build process, the build system works with an ephemeral key, which consists of private and public keys. The build system signs standard RHEL kernel modules, such as **kernel-modules-core**, **kernel-modules**, and **kernel-modules-extra**. After the completion of each kernel build, the private key becomes obsolete to sign third-party modules. You can use certificates from **db** and **MOK** for this purpose.

Additional resources

- [SBAT mechanism](#)
- [Red Hat Enterprise Linux and Secure Boot in cloud](#)

5.3. CONFIGURING A RHEL INSTANCE ON THE AWS MARKETPLACE WITH SECURE BOOT

To ensure that a Red Hat Enterprise Linux (RHEL) instance on AWS has a secured operating system booting process, use Secure Boot. In the AWS Marketplace, you can configure Secure Boot on the RHEL instance launched by using an Amazon Machine Image (AMI) with the **uefi-preferred** boot mode enabled.

Without UEFI, the Secure Boot feature does not work.



WARNING

To avoid security issues, generate and keep private keys apart from the current RHEL instance. If Secure Boot secrets are stored on the same instance on which they are used, intruders can gain access to secrets for escalating their privileges. For more information on launching an AWS EC2 instance, see [Get started with Amazon EC2](#).

Prerequisites

1. The RHEL AMI has the **uefi-preferred** option enabled in boot settings:

```
$ aws ec2 describe-images --image-id <ami-099f85fc24d27c2a7> --region <us-east-2> |
grep -E '"ImageId"|"Name"|"BootMode"'

"ImageId": "ami-099f85fc24d27c2a7",
"Name": "RHEL-10.0.0_HVM_GA-20250423-x86_64-0-Hourly2-GP3",
"BootMode": "uefi-preferred"
```

2. You have installed the following packages on the RHEL instance:

- **awscli2**
- **python3**
- **openssl**
- **efivar**
- **keyutils**
- **edk2-ovmf**
- **python3-virt-firmware**

Procedure

1. Check the platform status of the RHEL Marketplace AMI instance:

```
$ mokutil --sb-state

SecureBoot disabled
Platform is in Setup Mode
```

The **setup** mode allows updating the Secure Boot UEFI variables within the instance.

2. Create a new random universally unique identifier (UUID) and store it in a system-generated text file:

```
$ uuidgen --random > GUID.txt
```

3. Generate a new **PK.key** RSA private key and a self-signed **PK.cer** X.509 certificate for the Platform Key database:

```
$ openssl req -quiet \
-newkey rsa:3072 \
-nodes -keyout PK.key \
-new -x509 -sha256 \
-days 3650 \
-subj "/CN=Platform key/" \
-outform DER -out PK.cer
```

The **openssl** utility generates a common name **Platform key** for the certificate by setting output format to Distinguished Encoding Rules (DER).

4. Generate a new **KEK.key** RSA private key and a self-signed **KEK.cer** X.509 certificate for the Key Exchange Key database:

```
$ openssl req -quiet \
-newkey rsa:3072 \
-nodes -keyout KEK.key \
-new -x509 -sha256 \
-days 3650 \
-subj "CN=Key Exchange Key/" \
-outform DER -out KEK.cer
```

5. Generate a **custom_db.cer** custom certificate:

```
$ openssl req -quiet \
-newkey rsa:3072 \
-nodes -keyout custom_db.key \
-new -x509 -sha256 \
-days 3650 \
-subj "/CN=Signature Database key/" \
--outform DER -out custom_db.cer
```

6. Download the **Microsoft Corporation UEFI CA 2011** Certificate:

```
$ wget https://go.microsoft.com/fwlink/p/?linkid=321194 --user-agent="Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36" -O MicCorUEFCA2011_2011-06-27.crt
```

7. Download the updated UEFI Revocation List File of forbidden signatures (**dbx**) for x64 bits system:

```
$ wget https://uefi.org/sites/default/files/resources/x64_DBXUpdate.bin
```

8. Generate UEFI variables file using the **virt-fw-vars** utility:

```
$ virt-fw-vars --set-pk "$(< GUID.txt)" PK.cer \
--add-kek "$(< GUID.txt)" KEK.cer \
--add-db "$(< GUID.txt)" custom_db.cer \
--add-db 77fa9abd-0359-4d32-bd60-28f4e78f784b MicCorUEFCA2011_2011-06-27.crt \
--set-dbx DBXUpdate.bin -i /usr/share/edk2/ovmf/OVMF_VARS.secboot.fd \
--output VARS
```

For details, see the **virt-fw-vars(1)** man page on your system.

9. Convert UEFI variables to the Extensible Firmware Interface (EFI) Signature List (ESL) format:

```
$ python3 /usr/share/doc/python3-virt-firmware/experimental/authfiles.py \
--input VARS \
--outdir .
$ for f in PK KEK db dbx; do tail -c +41 $f.auth > $f.esl; done
```

NOTE

Each GUID is an assigned value and represents an EFI parameter

- **8be4df61-93ca-11d2-aa0d-00e098032b8c:**
EFI_GLOBAL_VARIABLE_GUID
- **d719b2cb-3d3a-4596-a3bc-dad00e67656f:**
EFI_IMAGE_SECURITY_DATABASE_GUID

The **EFI_GLOBAL_VARIABLE_GUID** parameter maintains settings of the bootable devices and boot managers, while the **EFI_IMAGE_SECURITY_DATABASE_GUID** parameter represents the image security database for Secure Boot variables **db**, **dbx**, and storage of required keys and certificates.

10. Transfer the database certificates to the target instance, use the **efivar** utility to manage UEFI environment variables.

- a. To transfer **PK.esl**, enter:

```
# efivar -w -n 8be4df61-93ca-11d2-aa0d-00e098032b8c-PK -f PK.esl
```

- b. To transfer **KEK.esl**, enter:

```
# efivar -w -n 8be4df61-93ca-11d2-aa0d-00e098032b8c-KEK -f KEK.esl
```

-
- c. To transfer **db.esl**, enter:

```
# efivar -w -n d719b2cb-3d3a-4596-a3bc-dad00e67656f-db -f db.esl
```

- d. To transfer the **dbx.esl** UEFI revocation list file for x64 architecture, enter:

```
# efivar -w -n d719b2cb-3d3a-4596-a3bc-dad00e67656f-dbx -f dbx.esl
```

11. Reboot the instance from the AWS console.

Verification

1. Verify if Secure Boot is enabled:

```
$ mokutil --sb-state

SecureBoot enabled
```

2. Use the **keyctl** utility to verify the kernel keyring for the custom certificate:

```
$ sudo keyctl list %:.platform

4 keys in keyring:
786569360: ---lsrvr  0  0 asymmetric: Signature Database key:
5856827178d376838611787277dc1d090c575759
...
```

Additional resources

- [UEFI Revocation List File](#)
- [NitroTPM](#)

5.4. CONFIGURING A RHEL INSTANCE FROM A CUSTOM RHEL IMAGE WITH SECURE BOOT

To ensure that your Red Hat Enterprise Linux (RHEL) instance on AWS has a secured operating system booting process, use Secure Boot. During registration of a custom RHEL Amazon Machine Image (AMI), you can configure the Secure Boot mechanism.

This image consists of pre-stored Unified Extensible Firmware Interface (UEFI) variables. Therefore, instances launched from this AMI use the Secure Boot mechanism with the required variables during the first boot.

Prerequisites

1. You have created and uploaded an AWS AMI image. For details, see [Preparing and uploading AWS AMI](#).
2. You have installed the following packages:
 - **awscli2**

- **python3**
- **openssl**
- **efivar**
- **keyutils**
- **python3-virt-firmware**

Procedure

1. Create a new random universally unique identifier (UUID) and store it in a system-generated text file:

```
$ uuidgen --random > GUID.txt
```

2. Generate a new RSA private key **PK.key** and a self-signed X.509 certificate **PK.cer** for the platform key database:

```
$ openssl req -quiet \
  -newkey rsa:3072 \
  -nodes -keyout PK.key \
  -new -x509 -sha256 \
  -days 3650 \
  -outform DER -out PK.cer
```

The **openssl** utility generates a common name platform key for the certificate by setting output format to Distinguished Encoding Rules (DER).

3. Generate a new RSA private key **KEK.key** and a self-signed X.509 certificate **KEK.cer** for the Key Exchange Key database:

```
$ openssl req -quiet \
  -newkey rsa:3072 \
  -nodes -keyout KEK.key \
  -new -x509 -sha256 \
  -days 3650 \
  -subj "/CN=Key Exchange Key/" \
  -outform DER -out KEK.cer
```

4. Generate a custom certificate **custom_db.cer**:

```
$ openssl req -quiet \
  -newkey rsa:3072 \
  -nodes -keyout custom_db.key \
  -new -x509 -sha256 \
  -days 3650 -subj "/CN=Signature Database key/" \
  --outform DER -out custom_db.cer
```

5. Download the updated UEFI Revocation List File of forbidden signatures (**dbx**) for 64 bit system:

```
$ wget https://uefi.org/sites/default/files/resources/x64_DBXUpdate.bin
```

6. Use the **virt-fw-vars** utility to generate the **aws_blob.bin** binary file from keys, database certificates, and the UEFI variable store:

```
$ virt-fw-vars --output-aws aws_blob.bin \
--set-pk "$(< GUID.txt)" PK.cer \
--add-kek "$(< GUID.txt)" KEK.cer \
--add-db "$(< GUID.txt)" custom_db.cer \
--add-db 77fa9abd-0359-4d32-bd60-28f4e78f784b MicCorUEFCA2011_2011-06-27.crt \
--set-dbx x64_DBXUpdate.bin
```

The customized blob consists of:

- **PK.cer** with a self-signed X.509 certificate
 - **KEK.cer** and **custom_db.cer** with owner group GUID and Privacy Enhanced Mail (**pem**) format
 - **x64_DBXUpdate.bin** list downloaded from database of excluded signatures (**dbx**).
 - The **77fa9abd-0359-4d32-bd60-28f4e78f784b** UUID is for **MicCorUEFCA2011_2011-06-27.crt** Microsoft Corporation UEFI Certification Authority 2011.
7. Use the **awscli2** utility to create and register the AMI from a disk snapshot with the required Secure Boot variables:

```
$ aws ec2 register-image \
--name rhel-10.0-secure-boot \
--architecture x86_64 \
--virtualization-type hvm \
--root-device-name "/dev/sda1" \
--block-device-mappings "{\"DeviceName\": \"'/dev/sda1'\",\"Ebs\": {\"SnapshotId\": \"<snap-02d4db3813ff9b98e>\"}}\" \
--ena-support --boot-mode uefi \
--region eu-central-1 \
--uefi-data $(cat aws_blob.bin)
{
  "ImageId": "<ami-09125e5689d5fdf2b>"
}
```

8. Reboot the instance from the AWS Console.

Verification

1. Verify Secure Boot functionality:

```
$ mokutil --sb-state
SecureBoot enabled
```

2. Use the **keyctl** utility to verify the kernel keyring for the custom certificate:

```
$ sudo keyctl list %:.platform

4 keys in keyring:
216534498: ---lsrvr 0 0 asymmetric: Signature Database key:
5856827178d376838611787277dc1d090c575759
```

| ...

CHAPTER 6. CONFIGURING RHEL ON PUBLIC CLOUD PLATFORMS WITH AMD SEV SNP

AMD Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) aims to prevent VM integrity-based attacks and reduce memory integrity violations. AMD processors offer three hardware-based security mechanisms: SEV, SEV-ES, and SEV-SNP.

- **SEV:** The SEV mechanism encrypts virtual machine (VM) memory to prevent the hypervisor from accessing VM data.
- **SEV-ES:** SEV with Encrypted State (SEV-ES) extends SEV by encrypting CPU register states. This mechanism prevents the hypervisor from accessing or modifying VM CPU registers. Despite providing isolation between hypervisor and VM, it is still vulnerable to memory integrity attacks.
- **SEV-SNP:** SEV-SNP is an enhancement to SEV-ES that adds memory integrity protection along with VM encryption. This mechanism prevents the hypervisor from modifying page tables to redirect VM memory access, protecting against replay attacks and memory tampering.



NOTE

Before deploying Red Hat Enterprise Linux (RHEL) on a public cloud platform, always check with the corresponding cloud service provider for the support status and certification of the particular RHEL instance type.

6.1. PROPERTIES OF SEV-SNP

- **Secure Processor:** The AMD **EPYC** processor integrates a Secure Processor (SP) subsystem. AMD SP is a dedicated hardware component to manage keys and encryption operations.
- **Memory Integrity:** For managing virtualization and isolation, memory management unit (MMU) utilizes page tables to translate virtual addresses to guest-physical addresses. SEV-SNP uses nested page tables for translating guest-physical addresses to host-physical addresses. Once nested page tables are defined, the hypervisor or host cannot alter page tables to modify the VM into accessing different pages, resulting in protection of memory integrity. SEV-SNP uses this method to offer protection against replay attacks and malicious modifications to VM memory.
- **Memory Encryption:** The AMD **EPYC** processor hides the memory encryption key, which remains hidden from both host and VM.
- **Attestation report for verification:** A CPU-generated report about RHEL instance information in an authorized cryptographic format. This process confirms the authenticity and reliability of the initial CPU and memory state of the RHEL instance and AMD processor.



NOTE

Even if a hypervisor creates the primary memory and CPU register state of the VM, they remain hidden and inaccessible to the hypervisor after initialization of that VM.

6.2. UNDERSTANDING AMD SEV SNP SECURE BOOT PROCESS

1. **Initialization and measurement:** A SEV-SNP enabled hypervisor sets the initial state of a VM.

This hypervisor loads firmware binary into the VM memory and sets the initial register state. AMD Secure Processor (SP) measures the initial state of the VM and provides details to verify the initial state of the VM.

2. **Firmware:** The VM initiates the UEFI firmware. The firmware might include either stateful or stateless Virtual Trusted Platform Module (vTPM) implementation. Stateful vTPM maintains persistent cryptographic state across VM reboots and migrations, whereas stateless vTPM generates fresh cryptographic state for each VM session without persistence. Virtual Machine Privilege Levels (VMPL) technology isolates vTPM from the guest. VMPL offers hardware-enforced privilege isolation between different VM components and the hypervisor.
3. **vTPM:** Depending on your cloud service provider, for stateful vTPM implementation, the UEFI firmware might perform a remote attestation to decrypt the persistent state of vTPM.
 - a. The vTPM also measures facts about the boot process such as Secure Boot state, certificates used for signing boot artifacts, UEFI binary hashes, and so on.
4. **Shim:** When the UEFI firmware finishes the initialization process, it searches for the extended firmware interface (EFI) system partition. Then, the UEFI firmware verifies and executes the first stage boot loader from there. For RHEL, this is **shim**. The **shim** program allows non-Microsoft operating systems to load the second stage boot loader from the EFI system partition.
 - a. **shim** uses a Red Hat certificate to verify the second stage boot loader (**grub**) or Red Hat Unified Kernel Image (UKI).
 - b. **grub** or **UKI** unpacks, verifies, and executes Linux kernel and initial RAM filesystem (**initramfs**), and the kernel command line. This process ensures that the Linux kernel is loaded in a trusted and secured environment.
5. **Initramfs:** In **initramfs**, vTPM information automatically unlocks the encrypted root partition in case of full disk encryption technology.
 - a. When the root volume becomes available, **initramfs** transfers the execution flow to the root volume.
6. **Attestation:** The VM tenant gets access to the system and can perform a remote attestation to ensure that the accessed VM is an untampered Confidential Virtual Machine (CVM). Attestation is performed based on information from AMD SP and vTPM. This process confirms the authenticity and reliability of the initial CPU and memory state of the RHEL instance and AMD processor.
7. **TEE:** This process creates a Trusted Execution Environment (TEE) to ensure that booting of the VM is in a trusted and secured environment.

6.3. CONFIGURING A RHEL INSTANCE ON AMAZON WEB SERVICES WITH AMD SEV SNP

You can configure AMD Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) on Red Hat Enterprise Linux (RHEL) instances on Amazon Web Services (AWS) to create a trusted boot environment.

SEV-SNP is a security feature of confidential virtual machines (CVMs). SEV-SNP aims to protect your data from access by the hypervisor and cloud service provider, and is available only for the AMD EPYC processor family.

Prerequisites

- You have installed the **awscli2**, **openssh**, and **openssh-clients** packages.
- You have created an AWS EC2 instance using an AMD EPYC processor-based machine type from the supported list. For details, see [supported instance types](#).

Procedure

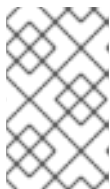
1. Check if SEV-SNP is enabled for the RHEL instance:

```
$ aws ec2 describe-instances --instance-ids <example_instance_id> \
--region <example_region>
```

```
...
"CpuOptions": {
  "CoreCount": 2,
  "ThreadsPerCore": 2,
  "AmdSevSnp": "enabled"
},
...
```

2. If SEV-SNP is not enabled, get ID of a RHEL Amazon Machine Image (AMI):

```
$ aws ec2 describe-images \
--owners 309956199498 \
--query 'sort_by(Images, &Name)[].[CreationDate,Name,ImageId]' \*
--filters "Name=name,Values=RHEL-10" \*
--region us-east-1 \
--output table
```



NOTE

Do not modify the command option **--owners 309956199498**. This is the account ID for displaying Red Hat images. If you need to list images for AWS GovCloud, use **--region us-gov-west-1** and **--owners 219670896067**.

3. Launch a RHEL instance with SEV-SNP enabled:

```
$ aws ec2 run-instances \
--image-id <example-rhel-10-ami-id> \
--instance-type m6a.4xlarge \
--key-name <example_key_pair_name> \
--subnet-id <example_subnet_id> \
--cpu-options AmdSevSnp=enabled
```

Verification

- Check kernel logs to verify status of SEV-SNP:

```
$ dmesg | grep -i sev
```

```
...
```

```
[ 7.509546] Memory Encryption Features active: AMD SEV SEV-ES SEV-SNP
[ 8.469487] SEV: Using SNP CUID table, 64 entries present.
[ 9.433348] SEV: SNP guest platform device initialized.
[ 33.314380] sev-guest sev-guest: Initialized SEV guest driver (using vmpck_id 0)
...
```

Additional resources

- [RHEL AMI on Amazon Web Services](#)

CHAPTER 7. CONFIGURING RHEL ON PUBLIC CLOUD PLATFORMS WITH UKI

To ensure that a Red Hat Enterprise Linux (RHEL) instance has a secured boot process from an untrusted storage such as confidential virtual machine (CVM) on a public cloud platform, use Unified Kernel Image (UKI).

7.1. INTRODUCTION TO UNIFIED KERNEL IMAGE

To extend the secure boot protection throughout the entire boot chain, use Unified Kernel Image (UKI).

Components of UKI

Unified Kernel Image (UKI) is a Unified Extensible Firmware Interface (UEFI) Portable Executable (PE) binary for the UEFI environment that bundles together the essential components of an operating system. UKI binary components extend the Secure Boot coverage by including the **initramfs** and the kernel command line. **Initramfs** is a part of the Linux startup process, while the kernel command line gives you limited access to define parameters. The primary components contained within the UKI binary are:

- The **.linux** section stores the Linux kernel image.
- The **.initrd** section stores the initial RAM filesystem **initramfs**.
- The **.cmdline** section stores the kernel command line.
- Additional sections, such as **.sbat**.
- The Red Hat signature.

Features of RHEL UKI with pre-built **initramfs**

- Prohibits any malicious agent or component to alter any objects in the boot chain.
- Due to pre-built **initramfs**, the user does not need to build its custom **initramfs**, which results in a faster kernel installation.
- Provides support for the pre-built **initramfs** systems as it is similar in all installations such as virtual machine (VMs), containers, or cloud instances.
- Provides support for the **x86_64** architecture.
- Includes the **kernel-uki-virt** package.
- Built for virtual machines and cloud instances.

Limitation of UKI because of the reduced flexibility of the boot process

- When building the UKI, the operating system vendor creates **initramfs**. As a consequence, the listed and included kernel modules are static. You can use the **systemd** system and configuration extensions to address this limitation.
- The kernel command line parameters are static, which limits the use of parameters for different instance sizes or debugging options.

You can use the UKI command line extensions to overcome this limitation.

7.2. UNDERSTANDING THE UKI SECURE BOOT PROCESS

To protect your system against unauthorized boot-time modifications, use the secure boot mechanism with Unified Kernel Image (UKI).

When using UKI with secure boot, the system verifies each component in the boot chain to ensure system integrity and prevent malicious code execution.

Procedure

1. **UEFI Firmware:** The boot process starts from the Unified Extensible Firmware Interface (UEFI) firmware. For boot, Red Hat Enterprise Linux (RHEL) UKI requires UEFI firmware, because legacy basic input/output system (BIOS) firmware is not supported.
2. **Shim boot loader:** Use the **shim** boot loader for booting rather than directly booting the UKI (a PE binary) from the UEFI firmware. **shim** includes additional security mechanisms such as Machine Owner Key (**MOK**) and Secure Boot Advanced Targeting (SBAT).
3. **Signature verification (Secure Boot UEFI mechanism):** During boot, **shim** reads the UKI binary and the secure boot UEFI mechanism verifies the signature of UKI against trusted keys stored in the Secure Boot Allowed Signature Database (**db**) of the system, the **MOK** database, and the built-in database of the **shim** binary. If the signature key is valid, the verification passes.
4. **SBAT verification:** Immediately after signature verification, the **shim** boot loader verifies the SBAT rules at startup.
During SBAT verification, the system compares generation numbers, for components such as **systemd.rhel** or **linux.rhel**, embedded in the UKI by using the **.sbat** section against values in the **shim** boot loader. If the generation number for a component in the **shim** is higher than the generation number in the UKI, the binary gets automatically discarded, even if it was signed by a trusted key.

Note that the generation number is a version identifier for UEFI applications, such as **shim** and **grub**.

5. **Unpacking and Execution:** If verification passes, control passes from **shim** to the **systemd-stub** code in the UKI to continue the boot process.
6. **systemd-stub add-ons:** During execution, **systemd-stub** unpacks and extracts the contents of the **.cmdline** section, the plain text kernel command line, and the **.initrd** section, the temporary root file system, for the boot process.
Note that **systemd-stub** reads UKI add-ons, which are also PE binaries, and verifies their signatures to safely extend the kernel command line of UKI by appending the **.cmdline** content from add-ons. **systemd-stub** reads add-ons from two locations:
 - Global (UKI-independent) add-ons from the **/loader/addons/** directory on the Extensible Firmware Interface (EFI) System Partition (ESP).
 - Per-UKI add-ons from the **/EFI/Linux/<UKI-name>.extra.d/** directory on the ESP.
7. Control passes from **systemd-stub** to the Linux kernel and the operating system boot process continues.
From this point, secure boot with UKI mechanisms follows the standard kernel boot process.

