Red Hat Enterprise Linux 10

Deploying and managing RHEL on Microsoft Azure

Obtaining RHEL system images and creating RHEL instances on Azure

Last Updated: 2025-05-22

Red Hat Enterprise Linux 10 Deploying and managing RHEL on Microsoft Azure

Obtaining RHEL system images and creating RHEL instances on Azure

Legal Notice

Abstract

To use Red Hat Enterprise Linux (RHEL) in a public cloud environment, you can create and deploy RHEL system images on various cloud platforms, including Microsoft Azure. You can also create and configure a Red Hat High Availability (HA) cluster on Azure. The following chapters provide instructions for creating cloud RHEL instances and HA clusters on Azure. These processes include installing the required packages and agents, configuring fencing, and installing network resource agents.

Table of Contents

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 provide computing resources as a service. Instead of using on-premises 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

RHEL as a cloud instance located on a public cloud platform has the following benefits over RHEL on-premises physical systems or 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 choose 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 you spend using it. Therefore, you can optimize your costs based on your requirements.

- **Software-controlled configurations**
  The entire configuration of a cloud instance is saved as data on the cloud platform, and is controlled by software. Therefore, you can easily create, remove, clone, or migrate the instance. A cloud instance is also operated remotely in a cloud provider console and is connected 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 virtualized kernel. This kernel is separate from the host operating system and from the *client* system that you use to connect to the instance.

  Therefore, any operating system can be installed on the cloud instance. This means that on a RHEL public cloud instance, you can run RHEL-specific applications that cannot be used on your local operating system.

  In addition, even if the operating system of the instance becomes unstable or is compromised, your client system is not affected in any way.

Additional resources

- What is public cloud?

- What is a hyperscaler?
- Types of cloud computing
- Public cloud use cases for RHEL
- Obtaining RHEL for public cloud deployments

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 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, also known as *scaling up* and *scaling down*. Therefore, using RHEL on public cloud is recommended 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.
- Cloud instances are not affected by what happens in your local environment. Therefore, you can use them for backup and disaster recovery.

Potentially problematic use cases

- You are running an existing environment that cannot be adjusted. Customizing a cloud instance to fit the specific needs of an existing deployment may not be cost-effective 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.

Next steps

- 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-premises 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, may 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.

The general security of RHEL public cloud instances is managed as follows:

- 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?

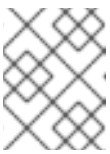
You can use RHEL instances on a public cloud platform regardless of your geographical location. Therefore, you can run your instances in the same region as your on-premises server.

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 may provide 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 a RHEL system in a public cloud environment, you need to:

1. Select the optimal cloud provider for your use case, based on your requirements and the current offer on the market.
   The certified cloud service providers (CCSP) for running RHEL instances are:
   - Amazon Web Services (AWS)
   - Google Cloud Platform (GCP)
   - Microsoft Azure

   NOTE

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

2. Create a RHEL cloud instance on your chosen cloud platform. For details, see Methods for creating RHEL cloud instances.
3. 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 Cataglogue

1.5. METHODS FOR CREATING RHEL CLOUD INSTANCES

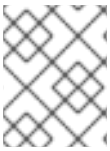
To deploy a RHEL instance on a public cloud platform, you can use one of the following methods:

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 you can build the image manually.
- This method uses your existing RHEL subscription, and 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.
- Your customer service is provided by Red Hat.
- For creating multiple images effectively, you can use the **cloud-init** tool.

Purchase a RHEL instance directly from the cloud provider marketplace

- You post-pay an hourly rate for using the service. Therefore, this method is also referred to as *pay as you go* (PAYG).
- Your customer service is provided by the cloud platform provider.



NOTE

For detailed instructions on using various methods to deploy RHEL instances On Microsoft Azure, see the following chapters in this document.

Additional resources

- What is a golden image?

CHAPTER 2. PREPARING AND UPLOADING VHD IMAGES TO MICROSOFT AZURE

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

2.1. PREPARING TO MANUALLY UPLOAD MICROSOFT AZURE VHD IMAGES

To create a VHD image that you can manually upload to **Microsoft Azure** cloud, you can use RHEL image builder.

Prerequisites

- You must have a Microsoft Azure resource group and storage account.
- You have Python installed. The **AZ CLI** tool depends on python.

Procedure

1. Import the Microsoft repository key:

```
$ sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

2. Create a local **azure-cli.repo** repository with the following information. Save the **azure-cli.repo** repository under **/etc/yum.repos.d/**:

```
[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/vscode
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc
```

3. Install Microsoft Azure CLI. The downloaded version of the Microsoft Azure CLI package can vary depending on the current available version.

```
$ sudo dnf downloader azure-cli
$ sudo rpm -ivh --nodeps azure-cli-2.0.64-1.el7.x86_64.rpm
```

4. Run Microsoft Azure CLI:

```
$ az login
```

The terminal shows the following message **Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code**. Then, the terminal opens a browser with a link to "https://microsoft.com/devicelogin" from where you can login.

NOTE

If you are running a remote (SSH) session, the login page link will not open in the browser. In this case, you can copy the link to a browser and login to authenticate your remote session. To sign in, use a web browser to open the "https://microsoft.com/devicelogin" page and enter the device code to authenticate.

5. List the keys for the storage account in Microsoft Azure and make note of value **key1** from the output of the previous command.

```
$ az storage account keys list --resource-group resource-group-name --account-name account-name
```

Replace ***resource-group-name*** with name of your Microsoft Azure resource group and ***storage-account-name*** with name of your Microsoft Azure storage account.

a. To list the available resources using the following command:

```
$ az resource list
```

6. Create a storage container:

```
$ az storage container create --account-name storage-account-name \
--account-key key1-value --name storage-account-name
```

Replace ***storage-account-name*** with name of the storage account.

Additional resources

- Microsoft Azure CLI

2.2. MANUALLY UPLOADING VHD IMAGES TO MICROSOFT AZURE CLOUD

After you have created your customized VHD image, you can manually upload it to the Microsoft Azure cloud. When you create a **.vhd** image by using the CLI, RHEL image builder writes temporary files to the **/var** subdirectory. To prevent the **.vhd** image creation from failing, increase the **/var** subdirectory capacity to at least 15 to 20 GB free space to ensure availability.

Prerequisites

- Your system must be set up for uploading Microsoft Azure VHD images.
- You must have a Microsoft Azure VHD image created by RHEL image builder.
  - In the GUI, use the **Azure Disk Image (.vhd)** image type.
  - In the CLI, use the **vhd** output type.

Procedure

1. Push the image to Microsoft Azure and create an instance from it:

```
$ az storage blob upload --account-name account_name --container-name container_name -
-file image-disk.vhd --name image-disk.vhd --type page
```

2. After the upload to the Microsoft Azure Blob storage completes, create a Microsoft Azure image from it. Because the images that you create with RHEL image builder generate hybrid images that support to both the **V1 = BIOS** and **V2 = UEFI** instances types, you can specify the **--hyper-v-generation** argument. The default instance type is **V1**.

```
$ az image create --resource-group resource_group_name --name image-disk.vhd --os-type
linux --location location \
--source https://$account_name.blob.core.windows.net/container_name/image-disk.vhd
- Running
```

Verification

1. Create an instance either with the Microsoft Azure portal, or a command similar to the following:

```
$ az vm create --resource-group resource_group_name --location location --name vm_name
--image image-disk.vhd --admin-username azure-user --generate-ssh-keys
- Running
```

2. Use your private key via SSH to access the resulting instance. Log in as **azure-user**. This username was set on the previous step.

Additional Resources

- Composing an image for the **.vhd** format fails (Red Hat Knowledgebase)

2.3. CREATING AND AUTOMATICALLY UPLOADING VHD IMAGES TO MICROSOFT AZURE CLOUD

You can create **.vhd** images by using RHEL image builder that will be automatically uploaded to a Blob Storage of the Microsoft Azure Cloud service provider.

Prerequisites

- You have root access to the system.
- You have access to the RHEL image builder interface of the RHEL web console.
- You created a blueprint. See Creating a RHEL image builder blueprint in the web console interface.
- You have a Microsoft Storage Account created.
- You have a writable Blob Storage prepared.

Procedure

1. In the RHEL image builder dashboard, select the blueprint you want to use.
2. Click the **Images** tab.

3. Click **Create Image** to create your customized **.vhd** image.
   The **Create image** wizard opens.
   a. Select **Microsoft Azure (.vhd)** from the **Type** drop-down menu list.
   b. Check the **Upload to Azure** checkbox to upload your image to the Microsoft Azure Cloud.
   c. Enter the **Image Size** and click **Next**.
4. On the **Upload to Azure** page, enter the following information:
   a. On the Authentication page, enter:
      i. Your **Storage account** name. You can find it on the **Storage account** page, in the Microsoft Azure portal.
      ii. Your **Storage access key**. You can find it on the **Access Key** Storage page.
      iii. Click **Next**.
   b. On the **Authentication** page, enter:
      i. The image name.
      ii. The **Storage container**. It is the blob container to which you will upload the image. Find it under the **Blob service** section, in the Microsoft Azure portal.
      iii. Click **Next**.
5. On the **Review** page, click **Create**. The RHEL image builder and upload processes start.
   Access the image you pushed into **Microsoft Azure Cloud**.
6. Access the Microsoft Azure portal.
7. In the search bar, type "storage account" and click **Storage accounts** from the list.
8. On the search bar, type "Images" and select the first entry under **Services**. You are redirected to the **Image dashboard**.
9. On the navigation panel, click **Containers**.
10. Find the container you created. Inside the container is the **.vhd** file you created and pushed by using RHEL image builder.

Verification

1. Verify that you can create a VM image and launch it.
   a. In the search bar, type images account and click **Images** from the list.
   b. Click **+Create**.
   c. From the dropdown list, choose the resource group you used earlier.
   d. Enter a name for the image.
   e. For the **OS type**, select **Linux**.

f. For the **VM generation**, select **Gen 2**.

g. Under **Storage Blob**, click **Browse** and click through the storage accounts and container until you reach your VHD file.

h. Click **Select** at the end of the page.

i. Choose an Account Type, for example, **Standard SSD**.

j. Click **Review + Create** and then **Create**. Wait a few moments for the image creation.

2. To launch the VM, follow the steps:

a. Click **Go to resource**.

b. Click **+ Create VM** from the menu bar on the header.

c. Enter a name for your virtual machine.

d. Complete the **Size** and **Administrator account** sections.

e. Click **Review + Create** and then **Create**. You can see the deployment progress. After the deployment finishes, click the virtual machine name to retrieve the public IP address of the instance to connect by using SSH.

f. Open a terminal to create an SSH connection to connect to the VM.

Additional resources

- Microsoft Azure Storage Documentation
- Create a Microsoft Azure Storage account

CHAPTER 3. DEPLOYING A RHEL IMAGE AS A COMPUTE INSTANCE ON AZURE

To use a RHEL image on Microsoft Azure, convert the image to an Azure-compatible format and deploy a VM from the image to run as an Azure Compute VM. To create, customize, and deploy a RHEL Virtual Hard Disk (.vhd) as an Azure Disk Image format, you can use one of the following methods:

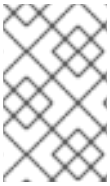
- Use the Red Hat image builder. For instructions, see Preparing and uploading VHD images to Microsoft Azure.
- Manually create and configure a VHD. This is a more complicated process but offers more granular customization options. For details, see the following sections.

Prerequisites

- You have created a Red Hat account.
- You have Microsoft Azure account.

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 a number of 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 Red Hat Image Builder. With Image Builder, you can create, upload, and deploy these custom images specific to your chosen CCSP.

Table 3.1. Image options

Image types	Subscriptions	Considerations	Sample scenario
Deploy a Red Hat Golden 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 Golden Image on the CCSP. For details on Golden 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 *pay-as-you-go* 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.

IMPORTANT

You cannot convert an on-demand instance to a custom RHEL instance. For migrating from an on-demand image to a custom RHEL bring your own subscription (BYOS) image:

- Create a new custom RHEL instance, then migrate data from your on-demand instance.
- When your data migration is completed, terminate the on-demand instance to avoid additional billing.

Next steps

- For the system requirement, check the required list of system packages.

Additional resources

- Red Hat Ecosystem Cataglogue
- Red Hat Cloud Access Reference Guide
- Using Red Hat Gold Images on Azure
- Azure Marketplace
- Billing options in the Azure Marketplace
- Red Hat Enterprise Linux Bring-Your-Own-Subscription Gold Images in Azure

3.2. REQUIRED SYSTEM PACKAGES

To create and configure a base image of RHEL, your host system must have the following packages installed.

Table 3.2. System packages

Package	Repository	Description
libvirt	rhel-10-for-x86_64-appstream-rpms	Open source API, daemon, and management tool for managing platform virtualization
virt-install	rhel-10-for-x86_64-appstream-rpms	A command-line utility for building VMs
libguestfs	rhel-10-for-x86_64-appstream-rpms	A library for accessing and modifying VM file systems
guestfs-tools	rhel-10-for-x86_64-appstream-rpms	System administration tools for VMs; includes the **virt-customize** utility

Next steps

- Follow the deployment steps in Deploying a RHEL instance by using a custom base image .

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

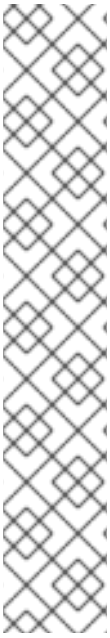
To manually configure a virtual machine (VM), first create a base (starter) image. Then, you can modify configuration settings and add the packages the VM requires to operate on the cloud. You can also make additional configuration changes for your specific application after you upload the image.

To prepare a cloud image of RHEL, follow the instructions in the sections below. To prepare a Hyper-V cloud image of RHEL, see the Prepare a Red Hat-based virtual machine from Hyper-V Manager .

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

Vertify the following VM configurations.

Settings are enabled either during the initial VM creation or provising VM image to Azure cloud.

- ssh - ssh must be enabled to provide remote access to your VMs
- dhcp - the primary virtual adapter should be configured for dhcp.
- Swap Space - Do not create a dedicated swap file or swap partition. You can configure swap space with the Windows Azure Linux Agent (WALinuxAgent).
- NIC - Choose virtio for the primary virtual network adapter.
- Encryption - For custom images, use Network Bound Disk Encryption (NBDE) for full disk encryption on Azure.

Prerequisites

- You have checked the required list of system packages.
- 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 VM:
   a. To install from the command line (CLI), ensure that you set the default memory, network interfaces, and CPUs as per 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 --auto-attach
```

Verification

- Verify that the **cloud-init** package is installed and enabled:

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

- Power down the VM.

Next steps

- Install the Azure CLI to access Azure resources and services.

3.4. INSTALLING THE AZURE CLI

By using the Azure Command-Line Interface (CLI), you can connect to Azure Cloud and manage Azure resources directly from your host terminal.

Prerequisites

- You have completed the deployment of a RHEL image on Azure .
- You have an account with Microsoft Azure.
- You have installed Python 3.x.

Procedure

1. Import the Microsoft repository key:

```
$ sudo dnf --import https://packages.microsoft.com/keys/microsoft.asc
```

2. Create a local Azure CLI repository entry:

```
$ sudo sh -c 'echo -e "[azure-cli]\nname=Azure
CLI\nbaseurl=https://packages.microsoft.com/yumrepos/azure-
cli\nenabled=1\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
/etc/yum.repos.d/azure-cli.repo'
```

3. Update the **dnf** package index:

```
$ sudo dnf update
```

4. Install the Azure CLI:

```
$ sudo dnf install -y azure-cli
```

5. Run the Azure CLI:

```
$ az login
```

Next steps

- Install the Hyper-V device drivers to efficiently run VM on Azure.

Additional resources

- Install the Azure CLI on Linux
- Azure CLI command reference

3.5. INSTALLING HYPER-V DEVICE DRIVERS

Microsoft provides network and storage device drivers as a part of their Linux Integration Services (LIS) for Hyper-V package. Prior to provisioning of a VM image as an Azure VM, install Hyper-V device drivers on it.

Prerequisites

- You have installed the Azure CLI.

Procedure

1. Check if Hyper-V device drivers are installed:

```
# lsinitrd | grep hv

drwxr-xr-x   2 root     root            0 Aug 12 14:21 usr/lib/modules/3.10.0-
932.el10.x86_64/kernel/drivers/hv
-rw-r--r--   1 root     root        31272 Aug 11 08:45 usr/lib/modules/3.10.0-
932.el10.x86_64/kernel/drivers/hv/hv_vmbus.ko.xz
-rw-r--r--   1 root     root        25132 Aug 11 08:46 usr/lib/modules/3.10.0-
932.el10.x86_64/kernel/drivers/net/hyperv/hv_netvsc.ko.xz
-rw-r--r--   1 root     root         9796 Aug 11 08:45 usr/lib/modules/3.10.0-
932.el10.x86_64/kernel/drivers/scsi/hv_storvsc.ko.xz
```

In case all the drivers are not installed, complete the remaining steps.

NOTE

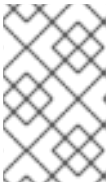
Though the **hv_vmbus** driver may exist in the environment, complete the following steps.

2. Create the **hv.conf** file in the **/etc/dracut.conf.d** directory.

```
# vi hv.conf
```

3. Add the following driver parameters to the **hv.conf** file:

```
add_drivers+=" hv_vmbus "
add_drivers+=" hv_netvsc "
add_drivers+=" hv_storvsc "
add_drivers+=" nvme "
```



NOTE

Make sure to have the spaces before and after the quotes, for example, **add_drivers+=" hv_vmbus "**. This ensures that unique drivers are loaded in the event that other Hyper-V drivers already exist in the environment.

4. Regenerate the **initramfs** image:

```
# dracut -f -v --regenerate-all
```

Verification

1. Reboot the machine.
2. Verify installation of drivers:

```
# lsinitrd | grep hv
```

Next steps

- Prepare your VM for deployment on Azure cloud.

3.6. PREPARING A VIRTUAL MACHINE FOR AZURE DEPLOYMENT

To ensure that the VM have compatibility and can operate in the Azure environment, perform the configuration changes before deploying a custom base image.

Prerequisites

- You have installed the Hyper-V device drivers.

Procedure

1. Log in and register the VM to enable the Red Hat Enterprise Linux repository:

```
# subscription-manager register --auto-attach
Installed Product Current Status:
Product Name: Red Hat Enterprise Linux for x86_64
Status: Subscribed
```

2. Install the **cloud-init** and **hyperv-daemons** packages:

```
# dnf install cloud-init hyperv-daemons -y
```

3. Create the **cloud-init** configuration files and edit them to provide integration with Azure services:

   a. To enable logging to the Hyper-V Data Exchange Service (KVP), edit the **/etc/cloud/cloud.cfg.d/10-azure-kvp.cfg** file and append the following lines:

```
reporting:
    logging:
        type: log
```

```
    telemetry:
      type: hyperv
```

b. To add the Azure datasource, edit the **/etc/cloud/cloud.cfg.d/91-azure_datasource.cfg** file and append the following lines:

```
datasource_list: [ Azure ]
datasource:
  Azure:
    apply_network_config: False
```

4. To block automatic loading of specific kernel modules, edit the **/etc/modprobe.d/blocklist.conf** file and append the following lines:

```
blacklist nouveau
blacklist lbm-nouveau
blacklist floppy
blacklist amdgpu
blacklist skx_edac
blacklist intel_cstate
```

5. Modify **udev** network device rules:

a. If present, remove the following persistent network device rules:

```
# rm -f /etc/udev/rules.d/70-persistent-net.rules
# rm -f /etc/udev/rules.d/75-persistent-net-generator.rules
# rm -f /etc/udev/rules.d/80-net-name-slot-rules
```

b. To ensure working of accelerated networking on Azure, edit the **/etc/udev/rules.d/68-azure-sriov-nm-unmanaged.rules** new network device rule and append the following line:

```
SUBSYSTEM=="net", DRIVERS=="hv_pci", ACTION=="add",
ENV{NM_UNMANAGED}="1"
```

6. Set the **sshd** service to start automatically:

```
# systemctl enable sshd
# systemctl is-enabled sshd
```

7. Modify kernel boot parameters:

a. Update the **GRUB_TIMEOUT** parameter value in the **/etc/default/grub** file:

```
GRUB_TIMEOUT=10
```

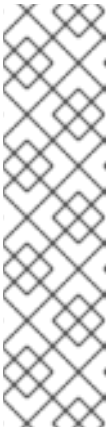
b. Remove the following option from the end of the **GRUB_CMDLINE_LINUX** line, if present:

```
rhgb quiet
```

c. Update the **/etc/default/grub** file with the following configuration details:

```
GRUB_CMDLINE_LINUX="loglevel=3 crashkernel=auto console=tty1 console=ttyS0
```

```
earlyprintk=ttyS0 rootdelay=300"
GRUB_TIMEOUT_STYLE=countdown
GRUB_TERMINAL="serial console"
GRUB_SERIAL_COMMAND="serial --speed=115200 --unit=0 --word=8 --parity=no --
stop=1"
```



NOTE

By adding the **elevator=none** option to the end of the **GRUB_CMDLINE_LINUX** line disables the I/O scheduler entirely. This option processes I/O requests as per the order of execution, without optimizing disk performance. With **elevator=none** on:

- HDD: Performance and throughput decreases, hence not suitable for running workloads.
- SSD: High performance and low latency, hence suitable for running workloads.

d. Regenerate the **grub.cfg** file:

- On a BIOS-based machine:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg --update-bls-cmdline
```

- On a UEFI-based machine:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg --update-bls-cmdline
```

WARNING

The path to rebuild **grub.cfg** is same for both BIOS and UEFI based machines. Original **grub.cfg** is present at BIOS path only. The UEFI path has a stub file that must not be modified or recreated using **grub2-mkconfig** command.

If your system uses a non-default location for **grub.cfg**, adjust the command accordingly.

8. Configure the Windows Azure Linux Agent (**WALinuxAgent**):

a. Install and enable the **WALinuxAgent** package:

```
# dnf install WALinuxAgent -y
# systemctl enable waagent
```

b. To prevent the use of a swap partition in provisioned VMs, edit the following lines in the **/etc/waagent.conf** file:

```
Provisioning.DeleteRootPassword=y
ResourceDisk.Format=n
ResourceDisk.EnableSwap=n
```

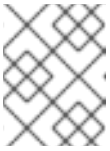
9. Prepare the VM for Azure provisioning:

   a. Unregister the VM from Red Hat Subscription Manager:

   ```
   # subscription-manager unregister
   ```

   b. Clean up the existing provisioning details:

   ```
   # waagent -force -deprovision
   ```

   NOTE

   This command generates warnings as Azure automatically handles the VM provisioning.

   c. Clear the shell history and shut down the VM:

   ```
   # export HISTSIZE=0
   # poweroff
   ```

Next steps

- To upload the RHEL image to Azure cloud, convert it to Azure disk image format .

3.7. CONVERTING A RHEL IMAGE TO AZURE DISK IMAGE

Microsoft Azure supports Azure disk image (.vhd) format. Hence, convert RHEL image to the **VHD** format. The image file must start at a position that is a multiple of 1 MB before it is converted to VHD. To convert the image from **qcow2** to a fixed **VHD** format, see the following procedure.

NOTE

The following commands use **qemu-img** version 2.12.0.

Prerequisites

- You have completed the steps of preparing VM for Azure deployment.

Procedure

1. Convert the image from **qcow2** to **raw** format.

   ```
   $ qemu-img convert -f qcow2 -O raw <image-example-name>.qcow2 <image-name>.raw
   ```

2. Edit the **align.sh** shell script:

   ```
   $ vi align.sh
   ```

```
#!/bin/bash
MB=$((1024 * 1024))
size=$(qemu-img info -f raw --output json "$1" | gawk 'match($0, /"virtual-size": ([0-9]+),/, val)
{print val[1]}')
rounded_size=$((($size/$MB + 1) * $MB))
if [ $(($size % $MB)) -eq  0 ]
then
 echo "Your image is already aligned. You do not need to resize."
 exit 1
fi
echo "rounded size = $rounded_size"
export rounded_size
```

3. Run the script:

```
$ sh align.sh <image-example-name>.raw
```

4. If the *Your image is already aligned. You do not need to resize.* message displays:

   a. Convert the file to a fixed **VHD** format:

   ```
   $ qemu-img convert -f raw -o subformat=fixed,force_size -O vpc <image-example-name>.raw <image-example-name>.vhd
   ```

   Once converted, the **VHD** file is ready to upload to Azure.

5. If a value displays mean the **raw** image is not aligned:

   a. Resize the **raw** file by using the rounded value as displayed above:

   ```
   $ qemu-img resize -f raw <image-example-name>.raw +1G
   ```

   b. Convert the **raw** image file to a **VHD** format.

   ```
   $ qemu-img convert -f raw -o subformat=fixed,force_size -O vpc <image-example-name>.raw <image-example-name>.vhd
   ```

   Once converted, the **VHD** file is ready to upload to Azure.

Next steps

- You can now configure Azure resources for your RHEL image .

3.8. CONFIGURING THE AZURE RESOURCES FOR A RHEL IMAGE

Azure resources are basic services of cloud based resource management such as compute, network, storage. You need to complete the Azure resources configuration before uploding the **VHD** file and create the Azure image.

Prerequisites

- You have completed the process of conversion of a RHEL image to Azure disk image .

Procedure

1. Authenticate your host with Azure credentials and log in:

```
$ az login
```

NOTE

If a browser is available for your environment, open the Azure sign-in page in the browser from the CLI. For details, see Sign in with Azure CLI.

2. Create a resource group in an Azure region:

```
$ az group create --name <resource-group> --location <azure-region>
```

Example:

```
[clouduser@localhost]$ az group create --name azrhelclirsgrp --location southcentralus
{
  "id": "/subscriptions//resourceGroups/azrhelclirsgrp",
  "location": "southcentralus",
  "managedBy": null,
  "name": "azrhelclirsgrp",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

3. Create a storage account with a valid SKU Types:

```
$ az storage account create -l <azure-region> -n <storage-account-name> -g <resource-group> --sku <sku_type>
```

Example:

```
$ az storage account create -l southcentralus -n azrhelclistact -g azrhelclirsgrp --sku Standard_LRS
{
  "accessTier": null,
  "creationTime": "2017-04-05T19:10:29.855470+00:00",
  "customDomain": null,
  "encryption": null,
  "id": "/subscriptions//resourceGroups/azrhelclirsgrp/providers/Microsoft.Storage/storageAccounts/azrhelclistact",
  "kind": "StorageV2",
  "lastGeoFailoverTime": null,
  "location": "southcentralus",
  "name": "azrhelclistact",
  "primaryEndpoints": {
    "blob": "https://azrhelclistact.blob.core.windows.net/",
    "file": "https://azrhelclistact.file.core.windows.net/",
    "queue": "https://azrhelclistact.queue.core.windows.net/",
```

```
    "table": "https://azrhelclistact.table.core.windows.net/"
},
"primaryLocation": "southcentralus",
"provisioningState": "Succeeded",
"resourceGroup": "azrhelclirsgrp",
"secondaryEndpoints": null,
"secondaryLocation": null,
"sku": {
  "name": "Standard_LRS",
  "tier": "Standard"
},
"statusOfPrimary": "available",
"statusOfSecondary": null,
"tags": {},
  "type": "Microsoft.Storage/storageAccounts"
}
```

4. Display the storage account details:

```
$ az storage account show-connection-string -n <storage-account-name> -g <resource-group>
```

Example:

```
$ az storage account show-connection-string -n azrhelclistact -g azrhelclirsgrp
{
  "connectionString":
"DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=azrhelclistact
AccountKey=NreGk...=="
}
```

5. Set the environment variable by exporting the existing connection string to connect system to the storage account:

```
$ export AZURE_STORAGE_CONNECTION_STRING="<storage-connection-string>"
```

Example:

```
$ export
AZURE_STORAGE_CONNECTION_STRING="DefaultEndpointsProtocol=https;EndpointSuffi
x=core.windows.net;AccountName=azrhelclistact;AccountKey=NreGk...=="
```

6. Create a storage container:

```
$ az storage container create -n <container-name>
```

Example:

```
$ az storage container create -n azrhelclistcont
{
  "created": true
}
```

7. Create a virtual network:

```
$ az network vnet create -g <resource group> --name <vnet-name> --subnet-name <subnet-name>
```

Example:

```
$ az network vnet create --resource-group azrhelclirsgrp --name azrhelclivnet1 --subnet-name azrhelclisubnet1
{
  "newVNet": {
    "addressSpace": {
      "addressPrefixes": [
      "10.0.0.0/16"
      ]
  },
  "dhcpOptions": {
    "dnsServers": []
  },
  "etag": "W/\"\"",
  "id":
"/subscriptions//resourceGroups/azrhelclirsgrp/providers/Microsoft.Network/virtualNetworks/azrhelclivnet1",
  "location": "southcentralus",
  "name": "azrhelclivnet1",
  "provisioningState": "Succeeded",
  "resourceGroup": "azrhelclirsgrp",
  "resourceGuid": "0f25efee-e2a6-4abe-a4e9-817061ee1e79",
  "subnets": [
    {
      "addressPrefix": "10.0.0.0/24",
      "etag": "W/\"\"",
      "id":
"/subscriptions//resourceGroups/azrhelclirsgrp/providers/Microsoft.Network/virtualNetworks/azrhelclivnet1/subnets/azrhelclisubnet1",
      "ipConfigurations": null,
      "name": "azrhelclisubnet1",
      "networkSecurityGroup": null,
      "provisioningState": "Succeeded",
      "resourceGroup": "azrhelclirsgrp",
      "resourceNavigationLinks": null,
      "routeTable": null
    }
  ],
  "tags": {},
  "type": "Microsoft.Network/virtualNetworks",
  "virtualNetworkPeerings": null
  }
}
```

Next steps

- You now upload the custom Azure disk image to Azure Blob storage .

Additional resources

- Azure Managed Disks Overview

3.9. UPLOADING A VHD IMAGE TO AZURE BLOB STORAGE

By using the Microsoft Azure Blob storage, you can manage the **VHD** file and create a custom Azure image.

WARNING

The exported storage connection string does not persist after a system reboot. If any of the commands in the following steps fail, export the connection string again. See Configuring the Azure resources for a RHEL image to obtain and export a connection string.

Prerequisites

- You have already configured Azure resources.

Procedure

1. Upload the **VHD** file to the storage container:

```
$ az storage blob upload \
    --account-name <storage-account-name> --container-name <container-name> \
    --type page --file <path-to-vhd> --name <image-name>.vhd
```

Example:

```
$ az storage blob upload \
--account-name azrhelclistact --container-name azrhelclistcont \
--type page --file ~/Downloads/rhel-image-10.vhd --name rhel-image-10.vhd

Percent complete: 100.0%
```

2. List the storage containers:

   a. To display in the tabular format, enter:

```
$ az storage container list --output table
```

   b. To display in the YAML format, enter:

```
$ az storage container list --output yaml
```

3. Use the URL for the uploaded **VHD** file from the 1st step:

```
$ az storage blob url -c <container-name> -n <image-name>.vhd <url-of-vhd-file>
```

Example:

```
$ az storage blob url -c azrhelclistcont -n rhel-image-10.vhd
"https://azrhelclistact.blob.core.windows.net/azrhelclistcont/rhel-image-10.vhd"
```

4. Create the Azure custom image:

```
$ az image create -n <image-name> -g <resource-group> -l <azure-region> --source <URL>
--os-type linux
```

NOTE

The default hypervisor generation of the VM is V1. You can optionally specify a V2 hypervisor generation by including the option **--hyper-v-generation V2**. Generation 2 VMs use a UEFI-based boot architecture. For details, see Support for generation 2 VMs on Azure. The command may return the error "Only blobs formatted as VHDs can be imported." This error may mean that the image was not aligned to the nearest 1 MB boundary before it was converted to **VHD**.

Example:

```
$ az image create -n rhel10 -g azrhelclirsgrp2 -l southcentralus --source
https://azrhelclistact.blob.core.windows.net/azrhelclistcont/rhel-image-10.vhd --os-type linux
```

Next steps

- You can launch and connect to a Azure VM .

3.10. LAUNCHING AND CONNECTING TO A RHEL VM IN AZURE

You need to create a managed disk Azure VM from the image.

Prerequisites

- You have completed the uploading of Azure VHD image to Azure Blob storage .

Procedure

1. Create the VM:

```
$ az vm create \
   -g <resource-group> -l <azure-region> -n <vm-name> \
   --vnet-name <vnet-name> --subnet <subnet-name> --size Standard_A2 \
   --os-disk-name <simple-name> --admin-username <administrator-name> \
   --generate-ssh-keys --image <path-to-image>
```

NOTE

The **--generate-ssh-keys** option creates a private and public key pair files in the ~/**.ssh** directory on your system. The public key is added to the **authorized_keys** file on the VM for the user specified by the **--admin-username** option. For details, see Types of SSH authentication methods.

Example:

```
$ az vm create \
-g azrhelclirsgrp2 -l southcentralus -n rhel-azure-vm-1 \
--vnet-name azrhelclivnet1 --subnet azrhelclisubnet1 --size Standard_A2 \
--os-disk-name vm-1-osdisk --admin-username clouduser \
--generate-ssh-keys --image rhel10

{
  "fqdns": "",
  "id":
"/subscriptions//resourceGroups/azrhelclirsgrp/providers/Microsoft.Compute/virtualMachines/rhe
l-azure-vm-1",
  "location": "southcentralus",
  "macAddress": "",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "<public-IP-address>",
  "resourceGroup": "azrhelclirsgrp2"
```

Note the **publicIpAddress** that is required to log in to the VM in the following step.

2. Start an SSH session and log in to the Azure VM:

```
[clouduser@localhost]$ ssh -i /home/clouduser/.ssh/id_rsa clouduser@<public-IP-address>.

The authenticity of host ',<public-IP-address>' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '<public-IP-address>' (ECDSA) to the list of known hosts.
```

3. If you see a user prompt, you have successfully deployed your Azure VM.

Launch the Microsoft Azure portal to manage VMs and check the audit logs and properties of assigned resources. You can also use the Azure CLI if you are managing multiple VMs. For details, enter **az --help** in the CLI or see the Azure CLI command reference.

3.11. TYPES OF SSH AUTHENTICATION METHODS

While it is an important security practice, in some cases, you can also connect to an Azure instance without the Azure-generated key pair. The following examples show two methods for a SSH authentication:

Example 1: Provision a new Azure VM with a password without generating a public key file.

```
$ az vm create \
   -g <resource-group> -l <azure-region> -n <vm-name> \
   --vnet-name <vnet-name> --subnet <subnet-name> --size Standard_A2 \
   --os-disk-name <simple-name> --authentication-type password \
   --admin-username <administrator-name> --admin-password <ssh-password> --image <path-to-
image>
```

```
$ ssh <admin-username>@<public-ip-address>
```

Example 2: Provision a new Azure VM with an existing public key file.

```
$ az vm create \
   -g <resource-group> -l <azure-region> -n <vm-name> \
   --vnet-name <vnet-name> --subnet <subnet-name> --size Standard_A2 \
   --os-disk-name <simple-name> --admin-username <administrator-name> \
   --ssh-key-value <path-to-existing-ssh-key> --image <path-to-image>
```

```
$ ssh -i <path-to-existing-ssh-key> <admin-username>@<public-ip-address>
```

3.12. ATTACHING RED HAT SUBSCRIPTIONS

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

Prerequisites

- You have an active Red Hat account.

Procedure

1. Register your system:

```
# subscription-manager register --auto-attach
```

2. Attach your subscriptions:
   - You can use an activation key to attach subscriptions. See Creating Red Hat Customer Portal Activation Keys for more information.
   - Alternatively, 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 Insights.

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

   For information on further configuration of Red Hat Insights, see Client Configuration Guide for Red Hat Insights.

Additional resources

- Creating Red Hat Customer Portal Activation Keys
- Attaching a host-based subscription to hypervisors
- Client Configuration Guide for Red Hat Insights
- Red Hat Cloud Access Reference Guide

3.13. CONFIGURING KDUMP FOR MICROSOFT AZURE INSTANCES

If a kernel crash occurs in a RHEL instance, you can use the **kdump** service to determine its cause. If **kdump** is configured correctly while your kernel instance terminates unexpectedly, **kdump** generates a

dump file, known as crash dump or a **vmcore** file. For debugging, you can then analyze the file to discover why the crash occurred.

For **kdump** to work on Microsoft Azure instances, you might need to adjust the **kdump** reserved memory and the **vmcore** target to fit VM sizes and RHEL versions.

Prerequisites

- You are using a VM from Microsoft Azure environment that supports **kdump**:
  - Standard_DS2_v2
  - Standard NV16as v4
  - Standard M416-208s v2
  - Standard M416ms v2
- You have the **root** permission.
- Your system meets the requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets ..

Procedure

1. Install **kdump** and other necessary packages:

```
# dnf install kexec-tools kdump-utils makedumpfile
```

2. Verify that the default location for crash dump files is set in the **kdump** configuration file and that the **/var/crash** file is available:

```
# grep -v "#" /etc/kdump.conf

path /var/crash
core_collector makedumpfile -l --message-level 7 -d 31
```

3. Based on the RHEL VM size and version, check if you need a **vmcore** target with more free space, such as **/mnt/crash**:

Table 3.3. Virtual machine sizes that have been tested with GEN2 VM on Azure

RHEL Version	Standard DS1 v2 (1 vCPU, 3.5GiB)	Standard NV16as v4 (16 vCPUs, 56 GiB)	Standard M416-208s v2 (208 vCPUs, 5700 GiB)	Standard M416ms v2 (416 vCPUs, 11400 GiB)
RHEL 9.4 - RHEL 10	Default	Default	Target	Target

- **Default** indicates that **kdump** works as expected with the default memory and the default **kdump** target. The default **kdump** target is the **/var/crash** file.

- **Target** indicates that **kdump** works as expected with the default memory. However, you might need to assign a target with more free space.

4. To assign a target with free space, such as **/mnt/crash**, edit the **/etc/kdump.conf** file and replace the default path:

```
$ sed s/"path /var/crash"/"path /mnt/crash"
```

   The option path **/mnt/crash** represents the path to the file system where **kdump** saves the crash dump file.

   For details, such as writing the crash dump file to a different partition, directly to a device or storing it to a remote machine, see Configuring the kdump target.

5. Increase the crash kernel size to the sufficient size for **kdump** to capture the **vmcore** by adding the relative boot parameter if the instance required:
   For example, for a Standard M416-208s v2 VM, the sufficient size is 512 MB, so the boot parameter would be **crashkernel=512M**.

   a. Open the GRUB configuration file and add **crashkernel=512M** to the boot parameter line:

```
# vi /etc/default/grub

GRUB_CMDLINE_LINUX="console=tty1 console=ttyS0 earlyprintk=ttyS0 rootdelay=300
crashkernel=512M"
```

   b. Update the GRUB configuration file:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg --update-bls-cmdline
```

6. Reboot the VM to allocate separate kernel crash memory to the VM.

Verification

- Ensure that **kdump** is active and running.

```
# systemctl status kdump
● kdump.service - Crash recovery kernel arming
   Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor prese>
   Active: active (exited) since Fri 2024-02-09 10:50:18 CET; 1h 20min ago
  Process: 1252 ExecStart=/usr/bin/kdumpctl start (code=exited, status=0/SUCCES>
 Main PID: 1252 (code=exited, status=0/SUCCESS)
    Tasks: 0 (limit: 16975)
   Memory: 512B
   CGroup: /system.slice/kdump.service
```

Additional resources

- How to troubleshoot kernel crashes, hangs, or reboots with kdump on Red Hat Enterprise Linux (Red Hat Knowledgebase)

CHAPTER 4. CONFIGURING A RED HAT HIGH AVAILABILITY CLUSTER ON MICROSOFT AZURE

By using the high availability (HA) cluster solution, you can create a cluster of RHEL nodes to automatically redistribute workloads if a node failure occurs. These HA clusters can also be deployed on public cloud platforms, such as Azure. Setting up HA clusters on Azure cloud is similar to configuring them in non-cloud environments.

To configure a Red Hat HA cluster on Azure that uses VHD instances as cluster nodes, see the following sections. You have a number of options for obtaining RHEL images for the cluster. For details, see Available RHEL image types for public cloud .

Prerequisites

- You have created a Red Hat account
- You have signed up and set up an Azure account.

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

A high-availability (HA) cluster is a set of computers (called *nodes*) that are linked together to run a specific workload. The purpose of HA clusters is to provide redundancy in case of a hardware or software failure. If a node in the HA cluster fails, the Pacemaker cluster resource manager distributes the workload to other nodes and no noticeable downtime occurs in the services that are running on the cluster.

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*: Additional nodes can be started when demand is high and stopped 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 provides 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 provides the following components for automation:

- **Pacemaker**, a cluster resource manager that provides both a command line utility (**pcs**) and a GUI (**pcsd**) to support multiple 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 like 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 to provide fault tolerance and system reliability.

4.2. INSTALLING THE HIGH AVAILABILITY PACKAGES AND AGENTS FOR AZURE

On each of the nodes, you need to install the High Availability packages and agents to be able to configure a Red Hat High Availability cluster on Azure.

Prerequisites

- You have completed deployment of RHEL image as an Azure compute instance .

Procedure

1. To get the public IP address for an Azure VM, open the VM properties in the Azure Portal or enter:

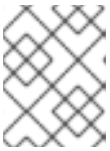
```
$ az vm list -g <resource-group> -d --output table
```

   Example:

```
$ az vm list -g azrhelclirsgrp -d --output table

Name    ResourceGroup          PowerState      PublicIps       Location
------  ---------------------  --------------  -------------   --------------
node01  azrhelclirsgrp         VM running      192.98.152.251  southcentralus
```

2. Register the VM with Red Hat:

```
$ sudo -i
# subscription-manager register --auto-attach
```



NOTE

If the **--auto-attach** command fails, manually register the VM to your subscription.

3. Disable all repositories:

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

4. Enable the RHEL Server HA repositories:

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

5. Update all packages:

```
# dnf update -y
```

6. Install the Red Hat HA Add-On packages, along with the Azure fencing agent:

```
# dnf install pcs pacemaker fence-agents-azure-arm
```

7. The **hacluster** user was created during the pcs and pacemaker installation in the previous step. Create a password for the **hacluster** and use it for all cluster 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

Created symlink from /etc/systemd/system/multi-user.target.wants/pcsd.service to
/usr/lib/systemd/system/pcsd.service.
```

Verification

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

```
# systemctl status pcsd.service
...
Active: active (running) since Fri 2018-02-23 11:00:58 EST; 1min 23s ago
...
```

Additional resources

- Support Policies for RHEL High Availability Clusters - Microsoft Azure Virtual Machines as Cluster Members

4.3. CREATING AN AZURE ACTIVE DIRECTORY APPLICATION

Complete the following procedure to create an Azure Active Directory (AD) application. The Azure AD application authorizes and automates access for HA operations for all nodes in the cluster.

Prerequisites

- The Azure Command Line Interface (CLI) is installed on your system.
- You are an Administrator or Owner for the Microsoft Azure subscription to create an Azure AD application.

Procedure

1. From any node in the HA cluster, log in to your Azure account:

```
$ az login
```

2. Create a **json** configuration file for a custom role for the Azure fence agent. Use the following configuration, but replace *<subscription-id>* with your subscription IDs:

```
{
    "Name": "Linux Fence Agent Role",
    "description": "Allows to power-off and start virtual machines",
    "assignableScopes": [
        "/subscriptions/<subscription-id>"
    ],
    "actions": [
        "Microsoft.Compute/*/read",
        "Microsoft.Compute/virtualMachines/powerOff/action",
        "Microsoft.Compute/virtualMachines/start/action"
    ],
    "notActions": [],
    "dataActions": [],
    "notDataActions": []
}
```

3. Define the custom role for the Azure fence agent. Use the **json** file created in the previous step:

```
$ az role definition create --role-definition azure-fence-role.json

{
  "assignableScopes": [
    "/subscriptions/<my-subscription-id>"
  ],
  "description": "Allows to power-off and start virtual machines",
  "id": "/subscriptions/<my-subscription-
id>/providers/Microsoft.Authorization/roleDefinitions/<role-id>",
  "name": "<role-id>",
  "permissions": [
    {
      "actions": [
        "Microsoft.Compute/*/read",
        "Microsoft.Compute/virtualMachines/powerOff/action",
        "Microsoft.Compute/virtualMachines/start/action"
      ],
      "dataActions": [],
      "notActions": [],
      "notDataActions": []
    }
  ],
  "roleName": "Linux Fence Agent Role",
  "roleType": "CustomRole",
  "type": "Microsoft.Authorization/roleDefinitions"
}
```

4. In the Azure web console interface, select **Virtual Machine** → Click **Identity** in the left-side menu.

5. Select **On** → Click **Save** → click **Yes** to confirm.

6. Click **Azure role assignments → Add role assignment**
7. Select the **Scope** required for the role, for example **Resource Group**.
8. Select the required **Resource Group**.
9. Optional: Change the **Subscription** if necessary.
10. Select the **Linux Fence Agent Role** role.
11. Click **Save**.

Verification

- Display nodes visible in Azure AD:

```
# fence_azure_arm --msi -o list
node1,
node2,
[...]
```

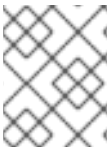
If this command outputs all nodes on your cluster, the AD application has been configured successfully.

Additional resources

- View the access a user has to Azure resources
- Create a custom role for the fence agent
- Assign Azure roles by using Azure CLI

4.4. CREATING A HIGH AVAILABILITY CLUSTER

You create a Red Hat High Availability Add-On cluster with the following procedure. This example procedure creates a cluster that consists of the 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.

Procedure

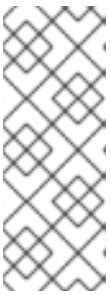
1. Authenticate the **pcs** user **hacluster** for each node in the cluster on the node from which you will be running **pcs**.
   The following command authenticates user **hacluster** on **z1.example.com** for both of the nodes in a two-node cluster that will consist of **z1.example.com** and **z2.example.com**.

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

2. 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
```

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



NOTE

For your particular environment, you may choose to leave the cluster services disabled by skipping this step. This allows you to ensure that if a node goes down, any issues with your cluster or your resources are resolved before the node rejoins the cluster. If you leave the cluster services disabled, you will 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
```

4. Display the status of the cluster you created with the **pcs cluster status** command. Because there may 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

...
```

4.5. CONFIGURING FENCING IN A RED HAT HIGH AVAILABILITY CLUSTER

If communication with a single node in the cluster fails, then other nodes in the cluster must be able to restrict or release access to resources that the failed cluster node may have access to. This cannot be accomplished by contacting the cluster node itself as the cluster node may not be responsive. Instead, you must provide an external method, which is called fencing with a fence agent. A fence device is an external device that can be used by the cluster to restrict access to shared resources by an errant node, or to issue a hard reboot on the cluster node.

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.

4.5.1. Displaying available fence agents and their options

The following commands can be used to 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.

4.5.2. Creating a fence device

The format for the command to create a fence device is as follows. For a listing of the available fence device creation options, see the **pcs stonith -h** display.

```
pcs stonith create stonith_id stonith_device_type [stonith_device_options] [op 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.

For information about the **pcmk_host_list** and **pcmk_host_map** parameters, see General properties of fencing devices.

After configuring a fence device, it is imperative that you test the device to ensure that it is working correctly. For information about testing a fence device, see Testing a fence device.

4.5.3. General properties of fencing devices

There are many general properties you can set for fencing devices, as well as various cluster properties that determine fencing behavior.

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	* **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.
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.

Field	Type	Default	Description
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.
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.

Field	Type	Default	Description
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
stonith-enabled	true	Indicates that failed nodes and nodes with resources that cannot be stopped should be fenced. Protecting your data requires that you set this **true**. If **true**, or unset, the cluster will refuse to start resources unless one or more STONITH resources have been configured also. Red Hat only supports clusters with this value set to **true**.
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	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. 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.
priority-fencing-delay	0 (disabled)	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.

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

4.5.4. Fencing delays

When cluster communication is lost in a two-node cluster, one node may detect this first and fence the other node. If both nodes detect this at the same time, however, each node may be able to initiate fencing of the other, leaving both nodes powered down or reset. By setting a fencing delay, you can decrease the likelihood of both cluster nodes fencing each other. You can set delays in a cluster with more than two nodes, but this is generally not of any benefit because only a partition with quorum will initiate fencing.

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.

4.5.5. Testing a fence device

Fencing is a fundamental part of the Red Hat Cluster infrastructure and it is important to validate or test that fencing is working properly.

NOTE

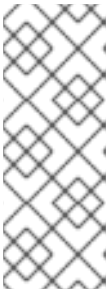
When a Pacemaker cluster node or Pacemaker remote node is fenced a hard kill should occur and not a graceful shutdown of the operating system. If a graceful shutdown occurs when your system fences a node, disable ACPI soft-off in the **/etc/systemd/logind.conf** file so that your system ignores any power-button-pressed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

Procedure

Use the following procedure to test a fence device.

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.

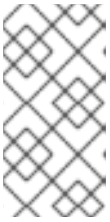
```
# fence_ipmilan -a ipaddress -l username -p password -o reboot
```

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
```

4.5.6. 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
```

For information about testing fence devices, see Testing a fence device.

4.5.6.1. Removing a fence level

The following command removes the fence level for the specified node and devices. If no nodes or devices are specified then the fence level you specify is removed from all nodes.

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

4.5.6.2. Clearing fence levels

The following command clears 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.

```
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
```

4.5.6.3. Verifying nodes and devices in fence levels

The following command verifies that all fence devices and nodes specified in fence levels exist.

```
pcs stonith level verify
```

4.5.6.4. 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. For example, 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
```

For information about node attributes, see https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/10/html-single/configuring_and_managing_high_availability_clusters/index#node-attributes

4.5.7. 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
```

4.5.8. 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.

4.5.8.1. 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]
```

4.5.8.2. 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
```

4.5.8.3. 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.

4.5.8.4. 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
```

4.5.8.5. 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
```

4.5.8.6. 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
```

4.5.8.7. 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.

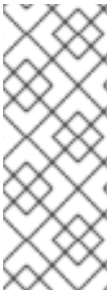
The following example prevents fence device **node1-ipmi** from running on **node1**.

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

4.5.9. 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.



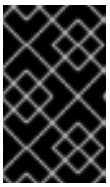
NOTE

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 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.

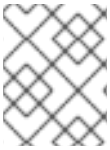


IMPORTANT

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.

4.5.9.1. Disabling ACPI Soft-Off with the BIOS

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

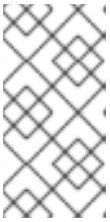


NOTE

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 Utiliy** 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          |                   |
| x Hot Key Power ON           Ctrl-F1        |                   |
|                                 |                   |
|                                 |                   |
+---------------------------------------------|-------------------+
```

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

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

To disable power-key handing in the **/etc/systemd/logind.conf** file, use the following procedure.

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
```

3. Verify that the node turns off immediately when fenced. For information about testing a fence device, see Testing a fence device.

4.5.9.3. 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

Use the following procedure to disable ACPI in the GRUB 2 file:

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.
3. Verify that the node turns off immediately when fenced. For information about testing a fence device, see Testing a fence device.

4.6. CREATING AN AZURE INTERNAL LOAD BALANCER

The Azure internal load balancer removes cluster nodes that do not respond to health probe requests. The following procedure has each step that references a specific Microsoft procedure and includes the settings for customizing the load balancer for HA.

Prerequisites

- You have logged in the Azure control panel.

Procedure

1. Create a Basic load balancer. Select **Internal load balancer**, the **Basic SKU**, and **Dynamic** for the type of IP address assignment.
2. Create a back-end address pool. Associate the backend pool to the availability set created while creating Azure resources in HA. Do not set any target network IP configurations.
3. Create a health probe. For the health probe, select **TCP** and enter port **61000**. You can use TCP port number that does not interfere with another service. For certain HA product applications (for example, SAP HANA and SQL Server), you may need to work with Microsoft to identify the correct port to use.
4. Create a load balancer rule. To create the load balancing rule, the default values are prepopulated. Ensure to set **Floating IP (direct server return)** to **Enabled**.

Next steps

Next steps

- You can configure the load balancer resource agent for Azure .

4.7. CONFIGURING THE LOAD BALANCER RESOURCE AGENT

After creating the health probe, you must configure the **load balancer** resource agent. This resource agent runs a service that responds health probe requests from the Azure load balancer and removes cluster nodes that do not respond requests.

Prerequisites

- You have completed the process of creating an Azure load balancer.

Procedure

1. Install the **nmap-ncat** resource agents on all nodes:

```
# dnf install nmap-ncat resource-agents-cloud
```

2. Perform the following steps on a single node:

   a. Create the **pcs** resources and group by using the **FrontendIP** load balancer for the **IPaddr2** address:

```
# pcs resource create resource-name IPaddr2 ip="10.0.0.7" --group cluster-resources-group
```

   b. Configure the **load balancer** resource agent:

```
# pcs resource create resource-loadbalancer-name azure-lb port=port-number --group cluster-resources-group
```

Verification

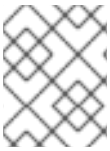
- Run **pcs status** to see the results.

```
[root@node01 clouduser]# pcs status

Cluster name: clusterfence01
Stack: corosync
Current DC: node02 (version 1.1.16-12.el7_4.7-94ff4df) - partition with quorum
Last updated: Tue Jan 30 12:42:35 2018
Last change: Tue Jan 30 12:26:42 2018 by root via cibadmin on node01

3 nodes configured
3 resources configured

Online: [ node01 node02 node03 ]

Full list of resources:

clusterfence (stonith:fence_azure_arm):      Started node01
Resource Group: g_azure
```

```
  vip_azure  (ocf::heartbeat:IPaddr2):       Started node02
  lb_azure   (ocf::heartbeat:azure-lb):      Started node02

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

4.8. CONFIGURING SHARED BLOCK STORAGE

To configure shared block storage for a Red Hat High Availability cluster with Microsoft Azure Shared Disks, use the following procedure. Note that this procedure is optional, and the steps below assume three Azure VMs (a three-node cluster) with a 1 TB shared disk.



NOTE

This is a stand-alone sample procedure for configuring block storage. The procedure assumes that you have not yet created your cluster.

Prerequisites

- You must have the Azure CLI on your host system and created SSH key(s).
- You must have created the cluster environment in Azure. To do so, follow the resources in this list:
  - Resource group
  - Virtual network
  - Network security group(s)
  - Network security group rules
  - Subnet(s)
  - Load balancer (optional)
  - Storage account
  - Proximity placement group
  - Availability set

Procedure

1. Create a shared block volume:

```
$ az disk create -g <resource_group> -n <shared_block_volume_name> --size-gb
<disk_size> --max-shares <number_vms> -l <location>
```

For example, the following command creates a shared block volume named **shared-block-volume.vhd** in the resource group **sharedblock** within the Azure Availability Zone **westcentralus**.

```
$ az disk create -g sharedblock-rg -n shared-block-volume.vhd --size-gb 1024 --max-shares
3 -l westcentralus

{
  "creationData": {
    "createOption": "Empty",
    "galleryImageReference": null,
    "imageReference": null,
    "sourceResourceId": null,
    "sourceUniqueId": null,
    "sourceUri": null,
    "storageAccountId": null,
    "uploadSizeBytes": null
  },
  "diskAccessId": null,
  "diskIopsReadOnly": null,
  "diskIopsReadWrite": 5000,
  "diskMbpsReadOnly": null,
  "diskMbpsReadWrite": 200,
  "diskSizeBytes": 1099511627776,
  "diskSizeGb": 1024,
  "diskState": "Unattached",
  "encryption": {
    "diskEncryptionSetId": null,
    "type": "EncryptionAtRestWithPlatformKey"
  },
  "encryptionSettingsCollection": null,
  "hyperVgeneration": "V1",
  "id": "/subscriptions/12345678910-12345678910/resourceGroups/sharedblock-
rg/providers/Microsoft.Compute/disks/shared-block-volume.vhd",
  "location": "westcentralus",
  "managedBy": null,
  "managedByExtended": null,
  "maxShares": 3,
  "name": "shared-block-volume.vhd",
  "networkAccessPolicy": "AllowAll",
  "osType": null,
  "provisioningState": "Succeeded",
  "resourceGroup": "sharedblock-rg",
  "shareInfo": null,
  "sku": {
    "name": "Premium_LRS",
    "tier": "Premium"
  },
  "tags": {},
  "timeCreated": "2020-08-27T15:36:56.263382+00:00",
  "type": "Microsoft.Compute/disks",
  "uniqueId": "cd8b0a25-6fbe-4779-9312-8d9cbb89b6f2",
  "zones": null
}
```

2. Verify the shared block volume:

```
$ az disk show -g <resource_group> -n <shared_block_volume_name>
```

For example, the following command shows details for the shared block volume **shared-block-volume.vhd** within the resource group **sharedblock-rg**.

```
$ az disk show -g sharedblock-rg -n shared-block-volume.vhd

{
  "creationData": {
    "createOption": "Empty",
    "galleryImageReference": null,
    "imageReference": null,
    "sourceResourceId": null,
    "sourceUniqueId": null,
    "sourceUri": null,
    "storageAccountId": null,
    "uploadSizeBytes": null
  },
  "diskAccessId": null,
  "diskIopsReadOnly": null,
  "diskIopsReadWrite": 5000,
  "diskMbpsReadOnly": null,
  "diskMbpsReadWrite": 200,
  "diskSizeBytes": 1099511627776,
  "diskSizeGb": 1024,
  "diskState": "Unattached",
  "encryption": {
    "diskEncryptionSetId": null,
    "type": "EncryptionAtRestWithPlatformKey"
  },
  "encryptionSettingsCollection": null,
  "hyperVgeneration": "V1",
  "id": "/subscriptions/12345678910-12345678910/resourceGroups/sharedblock-
rg/providers/Microsoft.Compute/disks/shared-block-volume.vhd",
  "location": "westcentralus",
  "managedBy": null,
  "managedByExtended": null,
  "maxShares": 3,
  "name": "shared-block-volume.vhd",
  "networkAccessPolicy": "AllowAll",
  "osType": null,
  "provisioningState": "Succeeded",
  "resourceGroup": "sharedblock-rg",
  "shareInfo": null,
  "sku": {
    "name": "Premium_LRS",
    "tier": "Premium"
  },
  "tags": {},
  "timeCreated": "2020-08-27T15:36:56.263382+00:00",
  "type": "Microsoft.Compute/disks",
  "uniqueId": "cd8b0a25-6fbe-4779-9312-8d9cbb89b6f2",
  "zones": null
}
```

3. Create three network interfaces. Run the following command three times by using a different **<nic_name>** for each node:

```
$ az network nic create \
    -g <resource_group> -n <nic_name> --subnet <subnet_name> \
    --vnet-name <virtual_network> --location <location> \
    --network-security-group <network_security_group> --private-ip-address-version IPv4
```

For example, the following command creates a network interface with the name **shareblock-nodea-vm-nic-protected**.

```
$ az network nic create \
    -g sharedblock-rg -n sharedblock-nodea-vm-nic-protected --subnet sharedblock-subnet-protected \
    --vnet-name sharedblock-vn --location westcentralus \
    --network-security-group sharedblock-nsg --private-ip-address-version IPv4
```

4. Create three VMs and attach the shared block volume. Option values are the same for each VM except that each VM has its own **<vm_name>**, **<new_vm_disk_name>**, and **<nic_name>**:

```
$ az vm create \
    -n <vm_name> -g <resource_group> --attach-data-disks <shared_block_volume_name> \
    --data-disk-caching None --os-disk-caching ReadWrite --os-disk-name <new-vm-disk-name> \
    --os-disk-size-gb <disk_size> --location <location> --size <virtual_machine_size> \
    --image <image_name> --admin-username <vm_username> --authentication-type ssh \
    --ssh-key-values <ssh_key> --nics <nic_name> --availability-set <availability_set> --ppg <proximity_placement_group>
```

For example, the following command creates a VM named **sharedblock-nodea-vm**:

```
$ az vm create \
-n sharedblock-nodea-vm -g sharedblock-rg --attach-data-disks shared-block-volume.vhd \
--data-disk-caching None --os-disk-caching ReadWrite --os-disk-name sharedblock-nodea-vm.vhd \
--os-disk-size-gb 64 --location westcentralus --size Standard_D2s_v3 \
--image /subscriptions/12345678910-12345678910/resourceGroups/sample-azureimagesgroupwestcentralus/providers/Microsoft.Compute/images/sample-azure-rhel-10.3.0-20200713.n.0.x86_64 --admin-username sharedblock-user --authentication-type ssh \
--ssh-key-values @sharedblock-key.pub --nics sharedblock-nodea-vm-nic-protected --availability-set sharedblock-as --ppg sharedblock-ppg

{
  "fqdns": "",
  "id": "/subscriptions/12345678910-12345678910/resourceGroups/sharedblock-rg/providers/Microsoft.Compute/virtualMachines/sharedblock-nodea-vm",
  "location": "westcentralus",
  "macAddress": "00-22-48-5D-EE-FB",
  "powerState": "VM running",
  "privateIpAddress": "198.51.100.3",
  "publicIpAddress": "",
  "resourceGroup": "sharedblock-rg",
  "zones": ""
}
```

Verification

1. For each VM in your cluster, verify that the block device is available. To do so, connect to the vm by using the **ssh** command with your VM's IP address:

```
# ssh <ip_address> "hostname ; lsblk -d | grep ' 1T '"
```

For example, the following command lists details including the host name and block device for the VM IP **198.51.100.3**:

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

nodea
sdb    8:16   0    1T  0 disk
```

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

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

For example, the following command lists details including the host name and shared disk volume ID for the instance IP address **198.51.100.3**:

```
# 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=3600224808dd8eb102f6ffc5822c41d89
```

After you have verified that the shared disk is attached to each VM, you can configure resilient storage for the cluster.

Additional resources

- Configuring a GFS2 file system in a cluster
- Configuring GFS2 file systems