Red Hat OpenStack Platform 16.2

Creating and Managing Images

Creating and Managing Images

Last Updated: 2021-10-21

Red Hat OpenStack Platform 16.2 Creating and Managing Images

Creating and Managing Images

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide provides procedures for creating and managing images, and procedures for configuring the Image service.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

CHAPTER 1. THE IMAGE SERVICE (GLANCE)

Manage images and storage in Red Hat OpenStack Platform (RHOSP).

A virtual machine image is a file that contains a virtual disk with a bootable operating system installed. Virtual machine images are supported in different formats. The following formats are available in RHOSP:

- **RAW** - Unstructured disk image format.
- **QCOW2** - Disk format supported by QEMU emulator. This format includes QCOW2v3 (sometimes referred to as QCOW3), which requires QEMU 1.1 or higher.
- **ISO** - Sector-by-sector copy of the data on a disk, stored in a binary file.
- **AKI** - Indicates an Amazon Kernel Image.
- **AMI** - Indicates an Amazon Machine Image.
- **ARI** - Indicates an Amazon RAMDisk Image.
- **VDI** - Disk format supported by VirtualBox virtual machine monitor and the QEMU emulator.
- **VHD** - Common disk format used by virtual machine monitors from VMware, VirtualBox, and others.
- **VMDK** - Disk format supported by many common virtual machine monitors.
- **PLOOP** - A disk format supported and used by Virtuozzo to run OS containers.
- **OVA** - Indicates that what is stored in the Image service (glance) is an OVA tar archive file.
- **DOCKER** - Indicates that what is stored in the Image service (glance) is a Docker tar archive of the container file system.

Although **ISO** is not normally considered a virtual machine image format, because ISOs contain bootable filesystems with an installed operating system, you use them in the same way as other virtual machine image files.

To download the official Red Hat Enterprise Linux cloud images, your account must have a valid Red Hat Enterprise Linux subscription:

- Red Hat Enterprise Linux 8 KVM Guest Image
- Red Hat Enterprise Linux 7 KVM Guest Image
- Red Hat Enterprise Linux 6 KVM Guest Image

If you are not logged in to the Customer Portal, a prompt opens where you must enter your Red Hat account credentials.

1.1. UNDERSTANDING THE IMAGE SERVICE

Red Hat OpenStack Platform (RHOSP) Image service (glance) features.

1.1.1. Supported Image service back ends

The following Image service (glance) back end scenarios are supported:

- RBD is the default back end when you use Ceph. For more information, see Configuring Ceph Storage in the *Advanced Overcloud Customization* guide.
- RBD multi-store. For more information, see Deploying the central site in the *Distributed compute node and storage deployment* guide.
- Object Storage (swift). For more information, see Using an External Object Storage Cluster in the *Advanced Overcloud Customization* guide.
- Block Storage (cinder). For more information, see Configuring cinder back end for the Image service in the *Advanced Overcloud Customization* guide.
  The Image service uses the Block Storage type and back end as the default.
- NFS. For more information, see Configuring NFS Storage in the *Advanced Overcloud Customization* guide.

  Important

  Although NFS is a supported Image service deployment option, more robust options are available.
  NFS is not native to the Image service. When you mount an NFS share on the Image service, the Image service does not manage the operation. The Image service writes data to the file system but is unaware that the back end is an NFS share.

  In this type of deployment, the Image service cannot retry a request if the share fails. This means that when a failure occurs on the back end, the store might enter read-only mode, or it might continue to write data to the local file system, in which case you risk data loss. To recover from this situation, you must ensure that the share is mounted and in sync, and then restart the Image service. For these reasons, Red Hat does not recommend NFS as an Image service back end.

  However, if you do choose to use NFS as an Image service back end, some of the following best practices can help to mitigate risks:

  - Use a reliable production-grade NFS back end.
  - Ensure that you have a strong and reliable connection between Controller nodes and the NFS back end, L2 is recommended.
  - Include monitoring and alerts for the mounted share.
  - Set underlying FS permissions.
    - Ensure that the user and the group that the glance-api process runs on do not have write permissions on the mount point at the local file system. This means that the process can detect possible mount failure and put the store into read-only mode during a write attempt.
    - The write permissions must be present in the shared file system that you use as a store.

1.1.2. Image signing and verification

Image signing and verification protects image integrity and authenticity by enabling deployers to sign images and save the signatures and public key certificates as image properties.

By taking advantage of this feature, you can:

- Sign an image using your private key and upload the image, the signature, and a reference to your public key certificate (the verification metadata). The Image service then verifies that the signature is valid.
- Create an image in the Compute service, have the Compute service sign the image, and upload the image and its verification metadata. The Image service again verifies that the signature is valid.
- Request a signed image in the Compute service. The Image service provides the image and its verification metadata, allowing the Compute service to validate the image before booting it.

For information on image signing and verification, see Validating Image Service (glance) images in the *Manage Secrets with OpenStack Key Manager* guide.

1.1.3. Image conversion

Image conversion converts images by calling the task API while importing an image.

As part of the import workflow, a plugin provides the image conversion. This plugin can be activated or deactivated based on the deployer configuration. Therefore, the deployer needs to specify the preferred format of images for the deployment.

Internally, the Image service receives the bits of the image in a particular format. These bits are stored in a temporary location. The plugin is then triggered to convert the image to the target format and moved to a final destination. When the task is finished, the temporary location is deleted. As a result, the format uploaded initially is not retained by the Image service.

For more information about image conversion, see Enabling image conversion.

NOTE

The conversion can be triggered only when importing an image. It does not run when uploading an image. For example:

```
$ glance image-create-via-import \
   --disk-format qcow2 \
   --container-format bare \
   --name NAME \
   --visibility public \
   --import-method web-download \
   --uri http://server/image.qcow2
```

1.1.4. Image introspection

Every image format comes with a set of metadata embedded inside the image itself. For example, a stream optimized **vmdk** contains the following parameters:

```
$ head -20 so-disk.vmdk

# Disk DescriptorFile
version=1
CID=d5a0bce5
parentCID=ffffffff
```

```
createType="streamOptimized"

# Extent description
RDONLY 209714 SPARSE "generated-stream.vmdk"

# The Disk Data Base
#DDB

ddb.adapterType = "buslogic"
ddb.geometry.cylinders = "102"
ddb.geometry.heads = "64"
ddb.geometry.sectors = "32"
ddb.virtualHWVersion = "4"
```

By introspecting this **vmdk**, you can easily know that the **disk_type** is **streamOptimized**, and the **adapter_type** is **buslogic**. These metadata parameters are useful for the consumer of the image. In Compute, the workflow to instantiate a **streamOptimized** disk is different from the one to instantiate a **flat** disk. This new feature allows metadata extraction. You can achieve image introspection by calling the task API while importing the image. An administrator can override metadata settings.

1.1.5. Interoperable image import

The interoperable image import workflow enables you to import images in two ways:

- Use the **web-download** (default) method to import images from a URI.
- Use the **glance-direct** method to import images from a local file system.

1.1.6. Improving scalability with Image service caching

Use the glance-api caching mechanism to store copies of images on Image service (glance) API servers and retrieve them automatically to improve scalability. With Image service caching, glance-api can run on multiple hosts. This means that it does not need to retrieve the same image from back end storage multiple times. Image service caching does not affect any Image service operations.

Configure Image service caching with the Red Hat OpenStack Platform director (tripleo) heat templates:

Procedure

1. In an environment file, set the value of the **GlanceCacheEnabled** parameter to **true**, which automatically sets the **flavor** value to **keystone+cachemanagement** in the **glance-api.conf** heat template:

```
parameter_defaults:
  GlanceCacheEnabled: true
```

2. Include the environment file in the **openstack overcloud deploy** command when you redeploy the overcloud.
3. Optional: Tune the **glance_cache_pruner** to an alternative frequency when you redeploy the overcloud. The following example shows a frequency of 5 minutes:

```
parameter_defaults:
  ControllerExtraConfig:
    glance::cache::pruner::minute: '*/5'
```

Adjust the frequency according to your needs to avoid file system full scenarios. Include the following elements when you choose an alternative frequency:

- The size of the files that you want to cache in your environment.
- The amount of available file system space.
- The frequency at which the environment caches images.

1.1.7. Image pre-caching

Red Hat OpenStack Platform (RHOSP) director can pre-cache images as part of the **glance-api** service.

1.1.7.1. Configuring the default interval for periodic image pre-caching

The default periodic interval for image pre-caching is 300 seconds. You can increase or decrease the default interval based on your requirements.

Procedure

1. Add a new interval with the **ExtraConfig** parameter in an environment file on the undercloud according to your requirements:

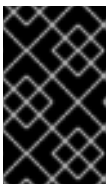
```
parameter_defaults:
  ControllerExtraConfig:
    glance::config::glance_api_config:
      DEFAULT/cache_prefetcher_interval:
        value: '<300>'
```

   Replace <300> with the number of seconds that you want as an interval to pre-cache images.

2. After you adjust the interval in the environment file in **/home/stack/templates/**, log in as the **stack** user and deploy the configuration:

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/<env_file>.yaml
```

   Replace <env_file> with the name of the environment file that contains the **ExtraConfig** settings that you added.



IMPORTANT

If you passed any extra environment files when you created the overcloud, pass them again here by using the **-e** option to avoid making undesired changes to the overcloud.

For more information about the **openstack overcloud deploy** command, see Deployment command in the *Director Installation and Usage* guide.

1.1.7.2. Using a periodic job to pre-cache an image

Prerequisites

To use a periodic job to pre-cache an image, you must use the **glance-cache-manage** command connected directly to the node where the **glance_api** service is running. Do not use a proxy, which hides the node that answers a service request. Because the undercloud might not have access to the network where the **glance_api** service is running, run commands on the first overcloud node, which is called **controller-0** by default.

Complete the following prerequisite procedure to ensure that you run commands from the correct host, have the necessary credentials, and are also running the **glance-cache-manage** commands from inside the **glance-api** container.

Procedure

1. Log in to the undercloud as the stack user and identify the provisioning IP address of **controller-0**:

```
(undercloud) [stack@site-undercloud-0 ~]$ openstack server list -f value -c Name -c
Networks | grep controller
overcloud-controller-1 ctlplane=192.168.24.40
overcloud-controller-2 ctlplane=192.168.24.13
overcloud-controller-0 ctlplane=192.168.24.71
(undercloud) [stack@site-undercloud-0 ~]$
```

2. To authenticate to the overcloud, copy the credentials that are stored in **/home/stack/overcloudrc**, by default, to **controller-0**:

```
$ scp ~/overcloudrc heat-admin@192.168.24.71:/home/heat-admin/
```

3. Connect to **controller-0**:

```
$ ssh heat-admin@192.168.24.71
```

4. On **controller-0** as the **heat-admin** user, identify the IP address of the **glance_api service**. In the following example, the IP address is **172.25.1.105**:

```
(overcloud) [root@controller-0 ~]# grep -A 10 '^listen glance_api' /var/lib/config-data/puppet-
generated/haproxy/etc/haproxy/haproxy.cfg
listen glance_api
 server central-controller0-0.internalapi.redhat.local 172.25.1.105:9292 check fall 5 inter 2000
rise 2
```

5. Because the **glance-cache-manage** command is only available in the **glance_api** container, create a script to exec into that container where the environment variables to authenticate to the overcloud are already set. Create a script called **glance_pod.sh** in **/home/heat-admin** on **controller-0** with the following contents:

```
sudo podman exec -ti \
 -e NOVA_VERSION=$NOVA_VERSION \
 -e COMPUTE_API_VERSION=$COMPUTE_API_VERSION \
 -e OS_USERNAME=$OS_USERNAME \
 -e OS_PROJECT_NAME=$OS_PROJECT_NAME \
```

```
 -e OS_USER_DOMAIN_NAME=$OS_USER_DOMAIN_NAME \
 -e OS_PROJECT_DOMAIN_NAME=$OS_PROJECT_DOMAIN_NAME \
 -e OS_NO_CACHE=$OS_NO_CACHE \
 -e OS_CLOUDNAME=$OS_CLOUDNAME \
 -e no_proxy=$no_proxy \
 -e OS_AUTH_TYPE=$OS_AUTH_TYPE \
 -e OS_PASSWORD=$OS_PASSWORD \
 -e OS_AUTH_URL=$OS_AUTH_URL \
 -e OS_IDENTITY_API_VERSION=$OS_IDENTITY_API_VERSION \
 -e OS_COMPUTE_API_VERSION=$OS_COMPUTE_API_VERSION \
 -e OS_IMAGE_API_VERSION=$OS_IMAGE_API_VERSION \
 -e OS_VOLUME_API_VERSION=$OS_VOLUME_API_VERSION \
 -e OS_REGION_NAME=$OS_REGION_NAME \
glance_api /bin/bash
```

6. Source the **overcloudrc** file and run the **glance_pod.sh** script to exec into the **glance_api** container with the necessary environment variables to authenticate to the overcloud Controller node.

```
[heat-admin@controller-0 ~]$ source overcloudrc
(overcloudrc) [heat-admin@central-controller-0 ~]$ bash glance_pod.sh
()[glance@controller-0 /]$
```

7. Use a command such as **glance image-list** to verify that the container can run authenticated commands against the overcloud.

```
()[glance@controller-0 /]$ glance image-list
+--------------------------------------+---------------------------------+
| ID                                   | Name                            |
+--------------------------------------+---------------------------------+
| ad2f8daf-56f3-4e10-b5dc-d28d3a81f659 | cirros-0.4.0-x86_64-disk.img    |
+--------------------------------------+---------------------------------+
()[glance@controller-0 /]$
```

Procedure

1. As the admin user, queue an image to cache:

```
$ glance-cache-manage --host=<host_ip> queue-image <image_id>
```

- Replace <host_ip> with the IP address of the Controller node where the **glance-api** container is running.
- Replace <image_id> with the ID of the image that you want to queue.
  When you have queued the images that you want to pre-cache, the **cache_images** periodic job prefetches all queued images concurrently.

NOTE

Because the image cache is local to each node, if your Red Hat OpenStack Platform is deployed with HA (with 3, 5, or 7 Controllers) then you must specify the host address with the **--host** option when you run the **glance-cache-manage** command.

2. Run the following command to view the images in the image cache:

```
$ glance-cache-manage --host=<host_ip> list-cached
```

Replace <host_ip> with the IP address of the host in your environment.

Related information

You can use additional **glance-cache-manage** commands for the following purposes:

- **list-cached** to list all images that are currently cached.
- **list-queued** to list all images that are currently queued for caching.
- **queue-image** to queue an image for caching.
- **delete-cached-image** to purge an image from the cache.
- **delete-all-cached-images** to remove all images from the cache.
- **delete-queued-image** to delete an image from the cache queue.
- **delete-all-queued-images** to delete all images from the cache queue.

1.1.8. Using the Image service API to enable sparse image upload

With the Image service (glance) API, you can use sparse image upload to reduce network traffic and save storage space. This feature is particularly useful in distributed compute node (DCN) environments. With a sparse image file, the Image service does not write null byte sequences. The Image service writes data with a given offset. Storage back ends interpret these offsets as null bytes that do not actually consume storage space.

Limitations

- Sparse image upload is supported only with Ceph RBD.
- Sparse image upload is not supported for file systems.
- Sparseness is not maintained during the transfer between the client and the Image service API. The image is sparsed at the Image service API level.

Prerequisites

- Your Red Hat OpenStack Platform (RHOSP) deployment uses RBD for the Image service back end.

Procedure

1. Log in to the undercloud node as the **stack** user.
2. Source the **stackrc** credentials file:

```
$ source stackrc
```

3. Create an environment file with the following content:

```
parameter_defaults:
  GlanceSparseUploadEnabled: true
```

4. Add your new environment file to the stack with your other environment files and deploy the overcloud:

```
$ openstack overcloud deploy \
--templates \
…
-e <existing_overcloud_environment_files> \
-e <new_environment_file>.yaml \
…
```

For more information about uploading images, see Upload an image.

Verification

You can import an image and check its size to verify sparse image upload.

1. Download the image file locally:

```
$ wget <file_location>/<file_name>
```

Replace **<file_location>** with the location of the file. Replace **<file_name>** with the name of the file.

The following procedure uses example commands. Replace the values with those from your environment where appropriate.

```
$ wget https://cloud.centos.org/centos/6/images/CentOS-6-x86_64-GenericCloud-1508.qcow2
```

2. Check the disk size and the virtual size of the image that you want to upload:

```
qemu-img info <file_name>
```

The following procedure uses example commands. Replace the values with those from your environment where appropriate.

```
$ qemu-img info CentOS-6-x86_64-GenericCloud-1508.qcow2

image: CentOS-6-x86_64-GenericCloud-1508.qcow2
file format: qcow2
virtual size: 8 GiB (8589934592 bytes)
disk size: 1.09 GiB
cluster_size: 65536
Format specific information:
compat: 0.10
refcount bits: 1
```

3. Import the image:

```
$ glance image-create-via-import --disk-format qcow2 --container-format bare --name centos_1 --file <file_name>
```

4. Record the image ID. It is required in a subsequent step.

5. Verify that the image is imported and in an active state:

```
$ openstack image show <image_id>
```

6. From a Ceph Storage node, verify that the size of the image is less than the virtual size from the output of step 1:

```
$ sudo rbd -p images diff <image_id> | awk '{ SUM += $2 } END { print SUM/1024/1024/1024
" GB" }'

1.03906 GB
```

7. Optional: You can confirm that **rbd_thin_provisioning** is configured in the glance configuration file on the Controller nodes:

   a. Use SSH to access a Controller node:

```
$ ssh -A -t heat-admin@<controller_node_IP_address>
```

   b. Confirm that **rbd_thin_provisioning** equals **True** on that Controller node:

```
$ sudo podman exec -it glance_api sh -c 'grep ^rbd_thin_provisioning /etc/glance/glance-
api.conf'
```

1.1.9. Secure metadef APIs

In Red Hat OpenStack Platform (RHOSP), users can define key value pairs and tag metadata with metadata definition (metadef) APIs. Currently, there is no limit on the number of metadef namespaces, objects, properties, resources, or tags that users can create.

Metadef APIs can leak information to unauthorized users. A malicious user can exploit the lack of restrictions and fill the Image service (glance) database with unlimited resources, which can create a Denial of Service (DoS) style attack.

Image service policies control metadef APIs. However, the default policy setting for metadef APIs allows all users to create or read the metadef information. Because metadef resources are not isolated to the owner, metadef resources with potentially sensitive names, such as internal infrastructure details or customer names, can expose that information to malicious users.

1.1.9.1. Configuring a policy to restrict metadef APIs

To make the Image service (glance) more secure, restrict metadef modification APIs to admin-only access by default in your Red Hat OpenStack Platform (RHOSP) deployments.

Procedure

1. As a cloud administrator, create a separate heat template environment file, such as **lock-down-glance-metadef-api.yaml**, to contain policy overrides for the Image service metadef API:

```
...
parameter_defaults:
  GlanceApiPolicies: {
```

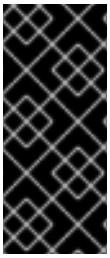
```
 glance-metadef_default: { key: 'metadef_default', value: '' },
 glance-metadef_admin: { key: 'metadef_admin', value: 'role:admin' },
 glance-get_metadef_namespace: { key: 'get_metadef_namespace', value:
'rule:metadef_default' },
 glance-get_metadef_namespaces: { key: 'get_metadef_namespaces', value:
'rule:metadef_default' },
 glance-modify_metadef_namespace: { key: 'modify_metadef_namespace', value:
'rule:metadef_admin' },
 glance-add_metadef_namespace: { key: 'add_metadef_namespace', value:
'rule:metadef_admin' },
 glance-delete_metadef_namespace: { key: 'delete_metadef_namespace', value:
'rule:metadef_admin' },
 glance-get_metadef_object: { key: 'get_metadef_object', value: 'rule:metadef_default' },
 glance-get_metadef_objects: { key: 'get_metadef_objects', value: 'rule:metadef_default' },
 glance-modify_metadef_object: { key: 'modify_metadef_object', value: 'rule:metadef_admin'
},
 glance-add_metadef_object: { key: 'add_metadef_object', value: 'rule:metadef_admin' },
 glance-delete_metadef_object: { key: 'delete_metadef_object', value: 'rule:metadef_admin' },
 glance-list_metadef_resource_types: { key: 'list_metadef_resource_types', value:
'rule:metadef_default' },
 glance-get_metadef_resource_type: { key: 'get_metadef_resource_type', value:
'rule:metadef_default' },
 glance-add_metadef_resource_type_association: { key:
'add_metadef_resource_type_association', value: 'rule:metadef_admin' },
 glance-remove_metadef_resource_type_association: { key:
'remove_metadef_resource_type_association', value: 'rule:metadef_admin' },
 glance-get_metadef_property: { key: 'get_metadef_property', value: 'rule:metadef_default' },
 glance-get_metadef_properties: { key: 'get_metadef_properties', value: 'rule:metadef_default'
},
 glance-modify_metadef_property: { key: 'modify_metadef_property', value:
'rule:metadef_admin' },
 glance-add_metadef_property: { key: 'add_metadef_property', value: 'rule:metadef_admin' },
 glance-remove_metadef_property: { key: 'remove_metadef_property', value:
'rule:metadef_admin' },
 glance-get_metadef_tag: { key: 'get_metadef_tag', value: 'rule:metadef_default' },
 glance-get_metadef_tags: { key: 'get_metadef_tags', value: 'rule:metadef_default' },
 glance-modify_metadef_tag: { key: 'modify_metadef_tag', value: 'rule:metadef_admin' },
 glance-add_metadef_tag: { key: 'add_metadef_tag', value: 'rule:metadef_admin' },
 glance-add_metadef_tags: { key: 'add_metadef_tags', value: 'rule:metadef_admin' },
 glance-delete_metadef_tag: { key: 'delete_metadef_tag', value: 'rule:metadef_admin' },
 glance-delete_metadef_tags: { key: 'delete_metadef_tags', value: 'rule:metadef_admin' }
  }

…
```

2. Include the environment file that contains the policy overrides in the deployment command with the **-e** option when you deploy the overcloud:

```
$ openstack overcloud deploy -e lock-down-glance-metadef-api.yaml
```

1.1.9.2. Enabling metadef APIs

If you previously restricted metadata definition (metadef) APIs or want to relax the new defaults, you can override metadef modification policies to allow users to update their respective resources.



IMPORTANT

Cloud administrators with users who depend on write access to the metadef APIs can make those APIs accessible to all users. In this type of configuration, however, there is the potential to unintentionally leak sensitive resource names, such as customer names and internal projects. Administrators must audit their systems to identify previously created resources that might be vulnerable even if only read access is enabled for all users.

Procedure

1. As a cloud administrator, log in to the undercloud and create a file for policy overrides. For example:

```
$ cat open-up-glance-api-metadef.yaml
```

2. Configure the policy override file to allow metadef API read-write access to all users:

```
GlanceApiPolicies: {
    glance-metadef_default: { key: 'metadef_default', value: '' },
    glance-get_metadef_namespace: { key: 'get_metadef_namespace', value:
'rule:metadef_default' },
    glance-get_metadef_namespaces: { key: 'get_metadef_namespaces', value:
'rule:metadef_default' },
    glance-modify_metadef_namespace: { key: 'modify_metadef_namespace', value:
'rule:metadef_default' },
    glance-add_metadef_namespace: { key: 'add_metadef_namespace', value:
'rule:metadef_default' },
    glance-delete_metadef_namespace: { key: 'delete_metadef_namespace', value:
'rule:metadef_default' },
    glance-get_metadef_object: { key: 'get_metadef_object', value: 'rule:metadef_default' },
    glance-get_metadef_objects: { key: 'get_metadef_objects', value: 'rule:metadef_default' },
    glance-modify_metadef_object: { key: 'modify_metadef_object', value:
'rule:metadef_default' },
    glance-add_metadef_object: { key: 'add_metadef_object', value: 'rule:metadef_default' },
    glance-delete_metadef_object: { key: 'delete_metadef_object', value: 'rule:metadef_default'
},
    glance-list_metadef_resource_types: { key: 'list_metadef_resource_types', value:
'rule:metadef_default' },
    glance-get_metadef_resource_type: { key: 'get_metadef_resource_type', value:
'rule:metadef_default' },
    glance-add_metadef_resource_type_association: { key:
'add_metadef_resource_type_association', value: 'rule:metadef_default' },
    glance-remove_metadef_resource_type_association: { key:
'remove_metadef_resource_type_association', value: 'rule:metadef_default' },
    glance-get_metadef_property: { key: 'get_metadef_property', value: 'rule:metadef_default'
},
    glance-get_metadef_properties: { key: 'get_metadef_properties', value:
'rule:metadef_default' },
    glance-modify_metadef_property: { key: 'modify_metadef_property', value:
'rule:metadef_default' },
    glance-add_metadef_property: { key: 'add_metadef_property', value: 'rule:metadef_default'
},
    glance-remove_metadef_property: { key: 'remove_metadef_property', value:
'rule:metadef_default' },
    glance-get_metadef_tag: { key: 'get_metadef_tag', value: 'rule:metadef_default' },
```

```
    glance-get_metadef_tags: { key: 'get_metadef_tags', value: 'rule:metadef_default' },
    glance-modify_metadef_tag: { key: 'modify_metadef_tag', value: 'rule:metadef_default' },
    glance-add_metadef_tag: { key: 'add_metadef_tag', value: 'rule:metadef_default' },
    glance-add_metadef_tags: { key: 'add_metadef_tags', value: 'rule:metadef_default' },
    glance-delete_metadef_tag: { key: 'delete_metadef_tag', value: 'rule:metadef_default' },
    glance-delete_metadef_tags: { key: 'delete_metadef_tags', value: 'rule:metadef_default' }
}
```

NOTE

You must configure all metadef policies to use **rule:metadeta_default**.

3. Include the new policy file in the deployment command with the **-e** option when you deploy the overcloud:

```
$ openstack overcloud deploy -e open-up-glance-api-metadef.yaml
```

1.2. MANAGE IMAGES

The Image service (glance) provides discovery, registration, and delivery services for disk and server images. It provides the ability to copy or snapshot a server image, and immediately store it. You can use stored images as a template to commission new servers quickly and more consistently than installing a server operating system and individually configuring services.

1.2.1. Creating an image

Manually create Red Hat OpenStack Platform (RHOSP) compatible images in the QCOW2 format by using Red Hat Enterprise Linux 7 ISO files, Red Hat Enterprise Linux 6 ISO files, or Windows ISO files.

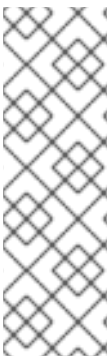
1.2.1.1. Use a KVM guest image with Red Hat OpenStack Platform

You can use a ready RHEL KVM guest QCOW2 image:

- Red Hat Enterprise Linux 8 KVM Guest Image
- Red Hat Enterprise Linux 7 KVM Guest Image
- Red Hat Enterprise Linux 6 KVM Guest Image

These images are configured with **cloud-init** and must take advantage of ec2-compatible metadata services for provisioning SSH keys to function correctly.

Ready Windows KVM guest QCOW2 images are not available.



NOTE

For the KVM guest images:

- The **root** account in the image is disabled, but **sudo** access is granted to a special user named **cloud-user**.
- There is no **root** password set for this image.

The **root** password is locked in **/etc/shadow** by placing **!!** in the second field.

For a RHOSP instance, generate an ssh keypair from the RHOSP dashboard or command line and use that key combination to perform an SSH public authentication to the instance as root.

When the instance is launched, this public key is injected to it. You can then authenticate by using the private key that you download when you create the keypair.

If you want to create custom Red Hat Enterprise Linux or Windows images, see Create a Red Hat Enterprise Linux 7 Image, Create a Red Hat Enterprise Linux 6 Image , or Create a Windows Image.

1.2.1.2. Create custom Red Hat Enterprise Linux or Windows images

Prerequisites

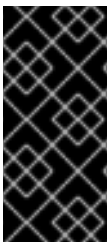
- Linux host machine to create an image. This can be any machine on which you can install and run the Linux packages.
- libvirt, virt-manager to install all packages necessary to create a guest operating system:

```
$ sudo dnf groupinstall -y @virtualization
```

- Libguestfs tools to install a set of tools to access and modify virtual machine images:

```
$ sudo dnf install -y libguestfs-tools-c
```

- A Red Hat Enterprise Linux 7 or 6 ISO file. For more information, see RHEL 7.2 Binary DVD or RHEL 6.8 Binary DVD or a Windows ISO file. If you do not have a Windows ISO file, see Microsoft TechNet Evaluation Center to download an evaluation image.
- Text editor, if you want to change the **kickstart** files (RHEL only).



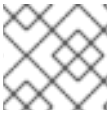
IMPORTANT

If you install the **libguestfs-tools** package on the undercloud, disable **iscsid.socket** to avoid port conflicts with the **tripleo_iscsid** service on the undercloud:

```
$ sudo systemctl disable --now iscsid.socket
```

1.2.1.2.1. Create a Red Hat Enterprise Linux 7 Image

Manually create a Red Hat OpenStack Platform (RHOSP) compatible image in the QCOW2 format by using a Red Hat Enterprise Linux 7 ISO file.



NOTE

You must run all commands with the **[root@host]#** on your host machine.

1. Start the installation by using **virt-install**:

```
[root@host]# qemu-img create -f qcow2 rhel7.qcow2 8G
[root@host]# virt-install --virt-type kvm --name rhel7 --ram 2048 \
--cdrom /tmp/rhel-server-7.2-x86_64-dvd.iso \
```

```
--disk rhel7.qcow2,format=qcow2 \
--network=bridge:virbr0 --graphics vnc,listen=0.0.0.0 \
--noautoconsole --os-type=linux --os-variant=rhel7
```

This launches an instance and starts the installation process.

NOTE

If the instance does not launch automatically, run the **virt-viewer** command to view the console:

```
[root@host]# virt-viewer rhel7
```

2. Configure the instance:

   a. At the initial Installer boot menu, select **Install Red Hat Enterprise Linux 7**.

   b. Choose the appropriate **Language** and **Keyboard** options.

   c. When prompted about which type of devices your installation uses, select **Auto-detected installation media**.

   d. When prompted about which type of installation destination, select **Local Standard Disks**. For other storage options, select **Automatically configure partitioning**.

   e. For software selection, select **Minimal Install**.

   f. For network and host name, select **eth0** for network and choose a host name for your device. The default host name is **localhost.localdomain**.

   g. Enter a password in the **Root Password** field and enter the same password again in the **Confirm** field.

   Result

   The installation process completes and the **Complete!** screen is displayed.

3. After the installation is complete, reboot the instance and log in as the root user.

4. Update the **/etc/sysconfig/network-scripts/ifcfg-eth0** file so that it contains only the following values:

```
TYPE=Ethernet
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
NM_CONTROLLED=no
```

5. Reboot the machine.

6. Register the machine with the Content Delivery Network.

```
# sudo subscription-manager register
# sudo subscription-manager attach --pool=Valid-Pool-Number-123456
# sudo subscription-manager repos --enable=rhel-7-server-rpms
```

7. Update the system:

```
# dnf -y update
```

8. Install the **cloud-init** packages:

```
# dnf install -y cloud-utils-growpart cloud-init
```

9. Edit the **/etc/cloud/cloud.cfg** configuration file and under **cloud_init_modules** add:

```
- resolv-conf
```

The **resolv-conf** option automatically configures the **resolv.conf** when an instance boots for the first time. This file contains information related to the instance such as **nameservers**, **domain** and other options.

10. Add the following line to **/etc/sysconfig/network** to avoid problems accessing the EC2 metadata service:

```
NOZEROCONF=yes
```

11. To ensure that the console messages appear in the **Log** tab on the dashboard and the **nova console-log** output, add the following boot option to the **/etc/default/grub** file:

```
GRUB_CMDLINE_LINUX_DEFAULT="console=tty0 console=ttyS0,115200n8"
```

12. Run the **grub2-mkconfig** command:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

The output is as follows:

```
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-229.7.2.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-229.7.2.el7.x86_64.img
Found linux image: /boot/vmlinuz-3.10.0-121.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-121.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-b82a3044fb384a3f9aeacf883474428b
Found initrd image: /boot/initramfs-0-rescue-b82a3044fb384a3f9aeacf883474428b.img
done
```

13. Deregister the instance so that the resulting image does not contain the subscription details for this instance:

```
# subscription-manager repos --disable=*
# subscription-manager unregister
# dnf clean all
```

14. Power off the instance:

```
# poweroff
```

15. Reset and clean the image by using the **virt-sysprep** command so that it can be used to create instances without issues:

```
[root@host]# virt-sysprep -d rhel7
```

16. Reduce the image size by converting any free space within the disk image back to free space within the host:

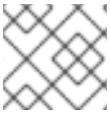
```
[root@host]# virt-sparsify --compress /tmp/rhel7.qcow2 rhel7-cloud.qcow2
```

This creates a new **rhel7-cloud.qcow2** file in the location from where the command is run.

The **rhel7-cloud.qcow2** image file is ready to be uploaded to the Image service. For more information about uploading this image to your RHOSP deployment by using the dashboard, see Upload an Image.

1.2.1.2.2. Create a Red Hat Enterprise Linux 6 Image

Manually create a Red Hat OpenStack Platform (RHOSP) compatible image in the QCOW2 format by using a Red Hat Enterprise Linux 6 ISO file.



NOTE

You must run all commands with the **[root@host]#** on your host machine.

1. Start the installation by using **virt-install**:

```
[root@host]# qemu-img create -f qcow2 rhel6.qcow2 4G
[root@host]# virt-install --connect=qemu:///system --network=bridge:virbr0 \
--name=rhel6 --os-type linux --os-variant rhel6 \
--disk path=rhel6.qcow2,format=qcow2,size=10,cache=none \
--ram 4096 --vcpus=2 --check-cpu --accelerate \
--hvm --cdrom=rhel-server-6.8-x86_64-dvd.iso
```

This launches an instance and starts the installation process.

NOTE

If the instance does not launch automatically, run the **virt-viewer** command to view the console:

```
[root@host]# virt-viewer rhel6
```

2. Configure the instances:

   a. At the initial Installer boot menu, select **Install or upgrade an existing system** and follow the installation prompts. Accept the defaults.
   The disk installer provides an option to test your installation media before installation. Select **OK** to run the test or **Skip** to proceed without testing.

   b. Choose the appropriate **Language** and **Keyboard** options.

   c. When prompted about which type of devices your installation uses, select **Basic Storage Devices**.

d. Choose a host name for your device. The default host name is **localhost.localdomain**.

e. Set the **timezone** and **root** password.

f. Based on the space on the disk, choose the type of installation you want from the options in the **Which type of installation would you like?** window.

g. Choose the **Basic Server** install, which installs an SSH server.

h. The installation process completes and the **Congratulations, your Red Hat Enterprise Linux installation is complete** screen is displayed.

3. Reboot the instance and log in as the **root** user.

4. Update the **/etc/sysconfig/network-scripts/ifcfg-eth0** file so that it contains only the following values:

```
TYPE=Ethernet
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
NM_CONTROLLED=no
```

5. Reboot the machine.

6. Register the machine with the Content Delivery Network:

```
# sudo subscription-manager register
# sudo subscription-manager attach --pool=Valid-Pool-Number-123456
# sudo subscription-manager repos --enable=rhel-6-server-rpms
```

7. Update the system:

```
# dnf -y update
```

8. Install the **cloud-init** packages:

```
# dnf install -y cloud-utils-growpart cloud-init
```

9. Edit the **/etc/cloud/cloud.cfg** configuration file and add the following content under **cloud_init_modules**.

```
- resolv-conf
```

The **resolv-conf** option automatically configures the **resolv.conf** configuration file when an instance boots for the first time. This file contains information related to the instance such as **nameservers**, **domain**, and other options.

10. To prevent network issues, create **/etc/udev/rules.d/75-persistent-net-generator.rules**:

```
# echo "#" > /etc/udev/rules.d/75-persistent-net-generator.rules
```

This prevents **/etc/udev/rules.d/70-persistent-net.rules** file from being created. If **/etc/udev/rules.d/70-persistent-net.rules** is created, networking might not function correctly

when you boot from snapshots, the network interface is created as **eth1** rather than **eth0** and the IP address is not assigned.

11. Add the following line to **/etc/sysconfig/network** to avoid problems accessing the EC2 metadata service:

```
NOZEROCONF=yes
```

12. To ensure that the console messages appear in the **Log** tab on the dashboard and the **nova console-log** output, add the following boot option to the **/etc/grub.conf** file:

```
console=tty0 console=ttyS0,115200n8
```

13. Deregister the virtual machine so that the resulting image does not contain the same subscription details for this instance:

```
# subscription-manager repos --disable=*
# subscription-manager unregister
# dnf clean all
```

14. Power off the instance:

```
# poweroff
```

15. Reset and clean the image by using the **virt-sysprep** command so that it can be used to create instances without issues:

```
[root@host]# virt-sysprep -d rhel6
```

16. Reduce image size by using the **virt-sparsify** command. This command converts any free space within the disk image back to free space within the host:

```
[root@host]# virt-sparsify --compress rhel6.qcow2 rhel6-cloud.qcow2
```

This creates a new **rhel6-cloud.qcow2** file in the location from where the command is run.

NOTE

You must manually resize the partitions of instances based on the image in accordance with the disk space in the flavor that is applied to the instance.

The **rhel6-cloud.qcow2** image file is ready to be uploaded to the Image service. For more information about uploading this image to your RHOSP deployment by using the dashboard, see Upload an Image

1.2.1.2.3. Create a Windows image

Manually create a Red Hat OpenStack Platform (RHOSP) compatible image in the QCOW2 format by using a Windows ISO file.

NOTE

You must run all commands with the **[root@host]#** on your host machine.

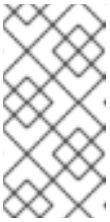
Procedure

1. Start the installation by using **virt-install**:

```
[root@host]# virt-install --name=<name> \
--disk size=<size> \
--cdrom=<path> \
--os-type=windows \
--network=bridge:virbr0 \
--graphics spice \
--ram=<ram>
```

Replace the following values of the **virt-install** parameters:

- <name> – the name that the Windows instance has.
- <size> – disk size in GB.
- <path> – the path to the Windows installation ISO file.
- <RAM> – the requested amount of RAM in MB.



NOTE

The **--os-type=windows** parameter ensures that the clock is configured correctly for the Windows guest, and enables its Hyper-V enlightenment features. You must also set **os_type=windows** in the image metadata before uploading the image to the Image service.

2. **virt-install** saves the guest image as **/var/lib/libvirt/images/<name>.qcow2** by default. If you want to keep the guest image elsewhere, change the parameter of the **--disk** option:

```
--disk path=<filename>,size=<size>
```

Replace <filename> with the name of the file that stores the instance image, and optionally its path. For example, **path=win8.qcow2,size=8** creates an 8 GB file named **win8.qcow2** in the current working directory.

TIP

If the guest does not launch automatically, run the **virt-viewer** command to view the console:

```
[root@host]# virt-viewer <name>
```

For more information about how to install Windows, see the relevant Microsoft documentation.

3. To allow the newly installed Windows system to use the virtualized hardware, you might need to install VirtIO drivers. To so do, first install the image, which you must attach as a CD-ROM drive to the Windows instance. To install the **virtio-win** package you must add the VirtIO ISO image to the instance, and install the VirtIO drivers. For more information, see Installing KVM paravirtualized drivers for Windows virtual machines in the *Configuring and managing virtualization* guide.

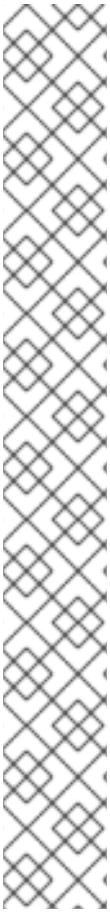
4. To complete the configuration, download and execute Cloudbase-Init on the Windows system.

At the end of the installation of Cloudbase-Init, select the **Run Sysprep** and **Shutdown** checkboxes. The **Sysprep** tool makes the guest unique by generating an OS ID, which is used by certain Microsoft services.

IMPORTANT

Red Hat does not provide technical support for Cloudbase-Init. If you encounter an issue, see contact Cloudbase Solutions .

When the Windows system shuts down, the **<name>.qcow2** image file is ready to be uploaded to the Image service. For more information about uploading this image to your RHOSP deployment by using the dashboard or the command line, see Upload an Image.



NOTE

libosinfo data

The Compute Service has deprecated support for using libosinfo data to set default device models. Instead, use the following image metadata properties to configure the optimal virtual hardware for an instance:

- **os_distro**
- **os_version**
- **hw_cdrom_bus**
- **hw_disk_bus**
- **hw_scsi_model**
- **hw_vif_model**
- **hw_video_model**
- **hypervisor_type**

For more information about these metadata properties, see Appendix A, *Image Configuration Parameters*.

1.2.2. Upload an image

1. In the dashboard, select **Project > Compute > Images**
2. Click **Create Image**.
3. Fill out the values, and click **Create Image**.

Table 1.1. Image options

Field	Notes
Name	Name for the image. The name must be unique within the project.

Field	Notes
Description	Brief description to identify the image.
Image Source	Image source: **Image Location** or **Image File**. Based on your selection, the next field is displayed.
Image Location or Image File	• Select **Image Location** option to specify the image location URL. • Select **Image File** option to upload an image from the local disk.
Format	Image format (for example, qcow2).
Architecture	Image architecture. For example, use i686 for a 32-bit architecture or x86_64 for a 64-bit architecture.
Minimum Disk (GB)	Minimum disk size required to boot the image. If this field is not specified, the default value is 0 (no minimum).
Minimum RAM (MB)	Minimum memory size required to boot the image. If this field is not specified, the default value is 0 (no minimum).
Public	If selected, makes the image public to all users with access to the project.
Protected	If selected, ensures only users with specific permissions can delete this image.

When the image has been successfully uploaded, its status is **active**, which indicates that the image is available for use. Note that the Image service can handle even large images that take a long time to upload – longer than the lifetime of the Identity service token which was used when the upload was initiated. This is due to the fact that the Image service first creates a trust with the Identity service so that a new token can be obtained and used when the upload is complete and the status of the image is to be updated.

NOTE

You can also use the **glance image-create** command with the **property** option to upload an image. More values are available on the command line. For a complete listing, see Image Configuration Parameters.

1.2.3. Update an image

1. In the dashboard, select **Project > Compute > Images**
2. Click **Edit Image** from the list.

NOTE

The **Edit Image** option is available only when you log in as an **admin** user. When you log in as a **demo** user, you have the option to **Launch an instance** or **Create Volume**.

3. Update the fields and click **Update Image** . You can update the following values - name, description, kernel ID, ramdisk ID, architecture, format, minimum disk, minimum RAM, public, protected.

4. Click the drop-down menu and select **Update Metadata** option.

5. Specify metadata by adding items from the left column to the right one. In the left column, there are metadata definitions from the Image Service Metadata Catalog. Select **Other** to add metadata with the key of your choice and click **Save** when finished.

NOTE

You can also use the **glance image-update** command with the **property** option to update an image. More values are available on the command line; for a complete listing, see Image Configuration Parameters.

1.2.4. Import an image

You can import images into the Image service (glance) using **web-download** to import an image from a URI and **glance-direct** to import an image from a local file system. The **web-download** method is enabled by default.

Import methods are configured by the cloud administrator. Run the **glance import-info** command to list available import options.

1.2.4.1. Import from a remote URI

You can use the **web-download** method to copy an image from a remote URI.

1. Create an image and specify the URI of the image to import.

```
glance image-create --uri <URI>
```

2. You can monitor the image availability by using the **glance image-show <image_id>** command. Replace **<image_id>** with the ID you provided during image creation.

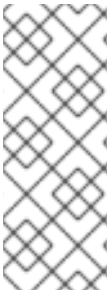
The Image service web download method uses a two-stage process to perform the import. First, it creates an image record. Second, it retrieves the image the specified URI. This method provides a more secure way to import images than the deprecated **copy-from** method used in Image API v1.

The URI is subject to optional denylist and allowlist filtering as described in the *Advanced Overcloud Customization* guide.

The Image Property Injection plugin may inject metadata properties to the image as described in the *Advanced Overcloud Customization* guide. These injected properties determine which compute nodes the image instances are launched on.

1.2.4.2. Import from a local volume

The **glance-direct** method creates an image record, which generates an image ID. After the image is uploaded to the service from a local volume, it is stored in a staging area and is made active after it passes any configured checks. The **glance-direct** method requires a shared staging area when used in a highly available (HA) configuration.



NOTE

Image uploads that use the **glance-direct** method fail in an HA environment if a common staging area is not present. In an HA active-active environment, API calls are distributed to the Image service controllers. The download API call can be sent to a different controller than the API call to upload the image. For more information about configuring the staging area, see Storage Configuration section in the *Advanced Overcloud Customization Guide*.

The glance-direct method uses three different calls to import an image:

- **glance image-create**
- **glance image-stage**
- **glance image-import**

You can use the **glance image-create-via-import** command to perform all three of these calls in one command. In the example below, replace uppercase words with the appropriate options.

```
glance image-create-via-import --container-format FORMAT --disk-format DISKFORMAT --name
NAME --file /PATH/TO/IMAGE
```

After the image moves from the staging area to the back end location, the image is listed. However, it might take some time for the image to become active.

You can monitor the image availability by using the **glance image-show <image_id>** command. Replace **<image_id** with the ID you provided during image creation.

1.2.5. Delete an image

1. In the dashboard, select **Project > Compute > Images**
2. Select the image you want to delete and click **Delete Images**.

1.2.6. Hide or unhide an image

You can hide public images from normal listings presented to users. For instance, you can hide obsolete CentOS 7 images and show only the latest version to simplify the user experience. Users can discover and use hidden images.

To hide an image:

```
glance image-update <image_id> --hidden 'true'
```

To create a hidden image, add the **--hidden** argument to the **glance image-create** command.

To unhide an image:

```
glance image-update <image_id> --hidden 'false'
```

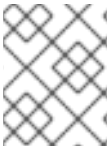
1.2.7. Show hidden images

To list hidden images:

```
glance image-list --hidden 'true'
```

1.2.8. Enabling image conversion

With the **GlanceImageImportPlugins** parameter enabled, you can upload a QCOW2 image, and the Image service can convert it to RAW.



NOTE

Image conversion is automatically enabled when you use Red Hat Ceph Storage RBD to store images and boot Nova instances.

To enable image conversion, create an environment file that contains the following parameter value and include the new environment file with the **-e** option in the **openstack overcloud deploy** command:

+

```
parameter_defaults:
  GlanceImageImportPlugins:'image_conversion'
```

1.2.9. Converting an image to RAW format

Red Hat Ceph Storage can store, but does not support using, QCOW2 images to host virtual machine (VM) disks.

When you upload a QCOW2 image and create a VM from it, the compute node downloads the image, converts the image to RAW, and uploads it back into Ceph, which can then use it. This process affects the time it takes to create VMs, especially during parallel VM creation.

For example, when you create multiple VMs simultaneously, uploading the converted image to the Ceph cluster might impact already running workloads. The upload process can starve those workloads of IOPS and impede storage responsiveness.

To boot VMs in Ceph more efficiently (ephemeral back end or boot from volume), the glance image format must be RAW.

Procedure

1. Converting an image to RAW might yield an image that is larger in size than the original QCOW2 image file. Run the following command before the conversion to determine the final RAW image size:

   ```
   qemu-img info <image>.qcow2
   ```

2. Convert an image from QCOW2 to RAW format:

```
qemu-img convert -p -f qcow2 -O raw <original qcow2 image>.qcow2 <new raw image>.raw
```

1.2.9.1. Configuring disk formats in the Image service (glance)

You can the configure the Image service (glance) to enable or reject disk formats by using the **GlanceDiskFormats** parameter.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. Include the **GlanceDiskFormats** parameter in an environment file, for example, **glance_disk_formats.yaml**:

```
parameter_defaults:
  GlanceDiskFormats:
    - <disk_format>
```

   - For example, use the following configuration to enable only RAW and ISO disk formats:

```
parameter_defaults:
  GlanceDiskFormats:
  - raw
  - iso
```

   - Use the following example configuration to reject QCOW2 disk images:

```
parameter_defaults:
  GlanceDiskFormats:
  - raw
  - iso
  - aki
  - ari
  - ami
```

4. Include the environment file that contains your new configuration in the **openstack overcloud deploy** command with any other environment files that are relevant to your environment:

```
$ openstack overcloud deploy --templates \
  -e <overcloud_environment_files> \
  -e <new_environment_file> \
  …
```

   - Replace **<overcloud_environment_files>** with the list of environment files that are part of your deployment.
   - Replace **<new_environment_file>** with the environment file that contains your new configuration.

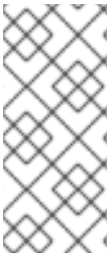
For more information about the disk formats available in RHOSP, see Image service.

1.2.10. Storing an image in RAW format

With the **GlanceImageImportPlugins** parameter enabled, run the following command to store a previously created image in RAW format:

```
$ glance image-create-via-import \
  --disk-format qcow2 \
  --container-format bare \
  --name NAME \
  --visibility public \
  --import-method web-download \
  --uri http://server/image.qcow2
```

- For **--name**, replace **NAME** with the name of the image; this is the name that will appear in **glance image-list**.
- For **--uri**, replace **http://server/image.qcow2** with the location and file name of the QCOW2 image.



NOTE

This command example creates the image record and imports it by using the **web-download** method. The glance-api downloads the image from the **--uri** location during the import process. If **web-download** is not available, **glanceclient** cannot automatically download the image data. Run the **glance import-info** command to list the available image import methods.

CHAPTER 2. IMAGE SERVICE WITH MULTIPLE STORES

The Red Hat OpenStack Platform Image service (glance) supports using multiple stores with distributed edge architecture so that you can have an image pool at every edge site. You can copy images between the central site, which is also known as the hub site, and the edge sites.

The image metadata contains the location of each copy. For example, an image present on two edge sites is exposed as a single UUID with three locations: the central site plus the two edge sites. This means you can have copies of image data that share a single UUID on many stores. For more information about locations, see Understanding the location of images.

With an RBD image pool at every edge site, you can boot VMs quickly by using Ceph RBD copy-on-write (COW) and snapshot layering technology. This means that you can boot VMs from volumes and have live migration. For more information about layering with Ceph RBD, see Ceph block device layering in the *Block Device Guide*.

2.1. REQUIREMENTS OF STORAGE EDGE ARCHITECTURE

- A copy of each image must exist in the Image service at the central location.
- Prior to creating an instance at an edge site, you must have a local copy of the image at that edge site.
- Images uploaded to an edge site must be copied to the central location before they can be copied to other edge sites.
- You must use raw images when deploying a DCN architecture with Ceph storage.
- RBD must be the storage driver for the Image, Compute and Block Storage services.
- For each site, you must assign the same value to the **NovaComputeAvailabilityZone** and **CinderStorageAvailabilityZone** parameters.

2.2. IMPORT AN IMAGE TO MULTIPLE STORES

Use the interoperable image import workflow to import image data into multiple Ceph Storage clusters. You can import images into the Image service that are available on the local file system or through a web server.

If you import an image from a web server, the image can be imported into multiple stores at once. If the image is not available on a web server, you can import the image from a local file system into the central store and then copy it to additional stores. For more information, see Copy an existing image to multiple stores.

IMPORTANT

Always store an image copy on the central site, even if there are no instances using the image at the central location. For more information about importing images into the Image service, see the Distributed compute node and storage deployment guide.

2.2.1. Manage image import failures

You can manage failures of the image import operation by using the **--allow-failure** parameter:

- If the value of the **--allow-failure** parameter to **true**, the image status becomes **active** after the first store successfully imports the data. This is the default setting. You can view a list of stores that failed to import the image data by using the **os_glance_failed_import** image property.
- If you set the value of the **--allow-failure** parameter to **false**, the image status only becomes **active** after all specified stores successfully import the data. Failure of any store to import the image data results in an image status of **failed**. The image is not imported into any of the specified stores.

2.2.2. Importing image data to multiple stores

Because the default setting of the **--allow-failure** parameter is **true**, you do not need to include the parameter in the command if it is acceptable for some stores to fail to import the image data.

NOTE

This procedure does not require all stores to successfully import the image data.

Procedure

1. Import image data to multiple, specified stores:

```
$ glance image-create-via-import \
--container-format bare \
--name IMAGE-NAME \
--import-method web-download \
--uri URI \
--stores STORE1,STORE2,STORE3
```

   - Replace *IMAGE-NAME* with the name of the image you want to import.
   - Replace *URI* with the URI of the image.
   - Replace *STORE1*, *STORE2*, and *STORE3* with the names of the stores to which you want to import the image data.
   - Alternatively, replace **--stores** with **--all-stores true** to upload the image to all the stores.

NOTE

The **glance image-create-via-import** command, which automatically converts the QCOW2 image to RAW format, works only with the **web-download** method. The **glance-direct** method is available, but it works only in deployments with a configured shared file system. For more information, see Storing an image in RAW format .

2.2.3. Importing image data to multiple stores without failure

This procedure requires all stores to successfully import the image data.

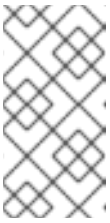
Procedure

1. Import image data to multiple, specified stores:

```
$ glance image-create-via-import \
```

```
--container-format bare \
--name IMAGE-NAME \
--import-method web-download \
--uri URI \
--stores STORE1,STORE2
```

- Replace *IMAGE-NAME* with the name of the image you want to import.
- Replace *URI* with the URI of the image.
- Replace *STORE1*, *STORE2*, and *STORE3* with the names of stores to which you want to copy the image data.
- Alternatively, replace **--stores** with **--all-stores true** to upload the image to all the stores.



NOTE

With the **--allow-failure** parameter set to **false**, the Image service does not ignore stores that fail to import the image data. You can view the list of failed stores with the image property **os_glance_failed_import**. For more information see Checking the progress of image import operation .

2. Verify that the image data was added to specific stores:

```
$ glance image-show IMAGE-ID | grep stores
```

Replace *IMAGE-ID* with the ID of the original existing image.

The output displays a comma-delimited list of stores.

2.2.4. Importing image data to a single store

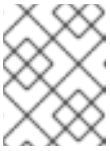
You can import image data to a single store.

Procedure

1. Import image data to a single store:

```
$ glance image-create-via-import \
--container-format bare \
--name IMAGE-NAME \
--import-method web-download \
--uri URI \
--store STORE
```

- Replace *IMAGE-NAME* with the name of the image you want to import.
- Replace *URI* with the URI of the image.
- Replace *STORE* with the name of the store to which you want to copy the image data.



NOTE

If you do not include the options of **--stores**, **--all-stores**, or **--store** in the command, the Image service creates the image in the central store.

2. Verify that the image data was added to specific store:

```
$ glance image-show IMAGE-ID | grep stores
```

Replace *IMAGE-ID* with the ID of the original existing image.

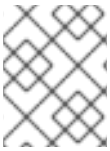
The output displays a comma-delimited list of stores.

2.2.5. Checking the progress of the image import operation

The interoperable image import workflow sequentially imports image data into stores. The size of the image, the number of stores, and the network speed between the central site and the edge sites impact how long it takes for the image import operation to complete.

You can follow the progress of the image import by looking at two image properties, which appear in notifications sent during the image import operation:

- The **os_glance_importing_to_stores** property lists the stores that have not imported the image data. At the beginning of the import, all requested stores show up in the list. Each time a store successfully imports the image data, the Image service removes the store from the list.
- The **os_glance_failed_import** property lists the stores that fail to import the image data. This list is empty at the beginning of the image import operation.



NOTE

In the following procedure, the environment has three Ceph Storage clusters: the **central** store and two stores at the edge, **dcn0** and **dcn1**.

Procedure

1. Verify that the image data was added to specific stores:

```
$ glance image-show IMAGE-ID
```

Replace *IMAGE-ID* with the ID of the original existing image.

The output displays a comma-delimited list of stores similar to the following example snippet:

```
| os_glance_failed_import       |
| os_glance_importing_to_stores | central,dcn0,dcn1
| status                        | importing
```

2. Monitor the status of the image import operation. When you precede a command with **watch**, the command output refreshes every two seconds.

```
$ watch glance image-show IMAGE-ID
```

Replace *IMAGE-ID* with the ID of the original existing image.

The status of the operation changes as the image import operation progresses:

```
| os_glance_failed_import      |
| os_glance_importing_to_stores | dcn0,dcn1
| status                       | importing
```

Output that shows that an image failed to import resembles the following example:

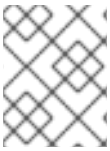
```
| os_glance_failed_import      | dcn0
| os_glance_importing_to_stores | dcn1
| status                       | importing
```

After the operation completes, the status changes to active:

```
| os_glance_failed_import      | dcn0
| os_glance_importing_to_stores |
| status                       | active
```

2.3. COPY AN EXISTING IMAGE TO MULTIPLE STORES

This feature enables you to copy existing images using Red Hat OpenStack Image service (glance) image data into multiple Ceph Storage stores at the edge by using the interoperable image import workflow.



NOTE

The image must be present at the central site before you copy it to any edge sites. Only the image owner or administrator can copy existing images to newly added stores.

You can copy existing image data either by setting **--all-stores** to **true** or by specifying specific stores to receive the image data.

- The default setting for the **--all-stores** option is **false**. If **--all-stores** is **false**, you must specify which stores receive the image data by using **--stores STORE1,STORE2**. If the image data is already present in any of the specified stores, the request fails.
- If you set **all-stores** to **true**, and the image data already exists in some of the stores, then those stores are excluded from the list.

After you specify which stores receive the image data, the Image service copies data from the central site to a staging area. Then the Image service imports the image data by using the interoperable image import workflow. For more information, see Importing an image to multiple stores .

IMPORTANT

Red Hat recommends that administrators carefully avoid closely timed image copy requests. Two closely timed copy-image operations for the same image causes race conditions and unexpected results. Existing image data remains as it is, but copying data to new stores fails.

2.3.1. Copying an image to all stores

Use the following procedure to copy image data to all available stores.

Procedure

1. Copy image data to all available stores:

```
$ glance image-import IMAGE-ID \
--all-stores true \
--import-method copy-image
```

   Replace *IMAGE-ID* with the name of the image you want to copy.

2. Confirm that the image data successfully replicated to all available stores:

```
$ glance image-list --include-stores
```

   For information about how to check the status of the image import operation, see Checking the progress of the image import operation.

2.3.2. Copying an image to specific stores

Use the following procedure to copy image data to specific stores.

Procedure

1. Copy image data to specific stores:

```
$ glance image-import IMAGE-ID \
--stores STORE1,STORE2 \
--import-method copy-image
```

   - Replace *IMAGE-ID* with the name of the image you want to copy.
   - Replace *STORE1* and *STORE2* with the names of the stores to which you want to copy the image data.

2. Confirm that the image data successfully replicated to the specified stores:

```
$ glance image-list --include-stores
```

   For information about how to check the status of the image import operation, see Checking the progress of the image import operation.

2.4. DELETING AN IMAGE FROM A SPECIFIC STORE

This feature enables you to delete an existing image copy on a specific store using Red Hat OpenStack Image service (glance).

Procedure

Delete an image from a specific store:

```
$ glance stores-delete --store _STORE_ID_ _IMAGE_ID_
```

- Replace _STORE_ID with the name of the store on which the image copy should be deleted.

- Replace *IMAGE_ID* with the ID of the image you want to delete.

WARNING

Using **glance image-delete** will permanently delete the image across all the sites. All image copies will be deleted, as well as the image instance and metadata.

2.5. UNDERSTANDING THE LOCATIONS OF IMAGES

Although an image can be present on multiple sites, there is only a single UUID for a given image. The image metadata contains the locations of each copy. For example, an image present on two edge sites is exposed as a single UUID with three locations: the central site plus the two edge sites.

Procedure

1. Show the sites on which a copy of the image exists:

```
$ glance image-show ID | grep "stores"

| stores |  default_backend,dcn1,dcn2
```

   In the example, the image is present on the central site, the **default_backend**, and on the two edge sites **dcn1** and **dcn2**.

2. Alternatively, you can run the **glance image-list** command with the **--include-stores** option to see the sites where the images exist:

```
$ glance image-list --include-stores

| ID                                   | Name    | Stores

| 2bd882e7-1da0-4078-97fe-f1bb81f61b00 | cirros | default_backend,dcn1,dcn2
```

3. List the image locations properties to show the details of each location:

```
$ openstack image show ID -c properties

| properties |

(--- cut ---)
locations='[{'url': 'rbd://79b70c32-df46-4741-93c0-8118ae2ae284/images/2bd882e7-1da0-
4078-97fe-f1bb81f61b00/snap', 'metadata': {'store': 'default_backend'}}, {'url': 'rbd://63df2767-
8ddb-4e06-8186-8c155334f487/images/2bd882e7-1da0-4078-97fe-f1bb81f61b00/snap',
'metadata': {'store': 'dcn1'}}, {'url': 'rbd://1b324138-2ef9-4ef9-bd9e-
aa7e6d6ead78/images/2bd882e7-1da0-4078-97fe-f1bb81f61b00/snap', 'metadata': {'store':
'dcn2'}}]',
(--- cut --)
```

   The image properties show the different Ceph RBD URIs for the location of each image.

In the example, the central image location URI is:

```
rbd://79b70c32-df46-4741-93c0-8118ae2ae284/images/2bd882e7-1da0-4078-97fe-f1bb81f61b00/snap', 'metadata': {'store': 'default_backend'}}
```

The URI is composed of the following data:

- **79b70c32-df46-4741-93c0-8118ae2ae284** corresponds to the central Ceph FSID. Each Ceph cluster has a unique FSID.
- The default value for all sites is **images**, which corresponds to the Ceph pool on which the images are stored.
- **2bd882e7-1da0-4078-97fe-f1bb81f61b00** corresponds to the image UUID. The UUID is the same for a given image regardless of its location.
- The metadata shows the glance store to which this location maps. In this example, it maps to the **default_backend**, which is the central hub site.

APPENDIX A. IMAGE CONFIGURATION PARAMETERS

The following keys can be used with the **property** option for both the **glance image-update** and **glance image-create** commands.

```
$ glance image-update IMG-UUID --property architecture=x86_64
```

Table A.1. Property keys

Specific to	Key	Description	Supported values
All	**architecture**	The CPU architecture that must be supported by the hypervisor. For example, **x86_64**, **arm**, or **ppc64**. Run **uname -m** to get the architecture of a machine.	• **alpha** - DEC 64-bit RISC • **armv7l** - ARM Cortex-A7 MPCore • **cris** - Ethernet, Token Ring, AXis-Code Reduced Instruction Set • **i686** - Intel sixth-generation x86 (P6 micro architecture) • **ia64** - Itanium • **lm32** - Lattice Micro32 • **m68k** - Motorola 68000 • **microblaze** - Xilinx 32-bit FPGA (Big Endian) • **microblazeel** - Xilinx 32-bit FPGA (Little Endian) • **mips** - MIPS 32-bit RISC (Big Endian) • **mipsel** - MIPS 32-bit RISC (Little Endian) • **mips64** - MIPS 64-bit RISC (Big Endian) • **mips64el** - MIPS 64-bit RISC (Little Endian) • **openrisc** - OpenCores RISC • **parisc** - HP Precision Architecture RISC • **parisc64** - HP Precision Architecture 64-bit RISC • **ppc** - PowerPC 32-bit • **ppc64** - PowerPC 64-bit

Specific to	Key	Description	Supported values
			• **ppcemb** - PowerPC ed 32-bit) • **s390** - IBM Enterprise Systems Architecture/390 • **s390x** - S/390 64-bit • **sh4** - SuperH SH-4 (Little Endian) • **sh4eb** - SuperH SH-4 (Big Endian) • **sparc** - Scalable Processor Architecture, 32-bit • **sparc64** - Scalable Processor Architecture, 64-bit • **unicore32** - Microprocessor Research and Development Center RISC Unicore32 • **x86_64** - 64-bit extension of IA-32 • **xtensa** - Tensilica Xtensa configurable microprocessor core • **xtensaeb** - Tensilica Xtensa configurable microprocessor core (Big Endian)
All	**hypervisor_type**	The hypervisor type.	**kvm**, **vmware**
All	**instance_uuid**	For snapshot images, this is the UUID of the server used to create this image.	Valid server UUID
All	**kernel_id**	The ID of an image stored in the Image Service that should be used as the kernel when booting an AMI-style image.	Valid image ID
All	**os_distro**	The common name of the operating system distribution in lowercase.	• **arch** - Arch Linux. Do not use **archlinux** or **org.archlinux**. • **centos** - Community Enterprise Operating System. Do not use **org.centos** or **CentOS**.

Specific to	Key	Description	Supported values
			• **debian** - Debian. Do not use **Debian** or **org.debian**. • **fedora** - Fedora. Do not use **Fedora**, **org.fedora**, or **org.fedoraproject**. • **freebsd** - FreeBSD. Do not use **org.freebsd**, **freeBSD**, or **FreeBSD**. • **gentoo** - Gentoo Linux. Do not use **Gentoo** or **org.gentoo**. • **mandrake** - Mandrakelinux (MandrakeSoft) distribution. Do not use **mandrakelinux** or **MandrakeLinux**. • **mandriva** - Mandriva Linux. Do not use **mandrivalinux**. • **mes** - Mandriva Enterprise Server. Do not use **mandrivaent** or **mandrivaES**. • **msdos** - Microsoft Disc Operating System. Do not use **ms-dos**. • **netbsd** - NetBSD. Do not use **NetBSD** or **org.netbsd**. • **netware** - Novell NetWare. Do not use **novell** or **NetWare**. • **openbsd** - OpenBSD. Do not use **OpenBSD** or **org.openbsd**. • **opensolaris** - OpenSolaris. Do not use **OpenSolaris** or **org.opensolaris**. • **opensuse** - openSUSE. Do not use **suse**, **SuSE**, or **org.opensuse**. • **rhel** - Red Hat Enterprise Linux. Do not use **redhat**, **RedHat**, or **com.redhat**. • **sled** - SUSE Linux Enterprise Desktop. Do not use **com.suse**.

Specific to	Key	Description	Supported values
			• **ubuntu** - Ubuntu. Do not use **com.ubuntu**, **org.ubuntu**, or **canonical**. • **windows** - Microsoft Windows. Do not use **com.microsoft.server**.
All	**os_version**	The operating system version as specified by the distributor.	Version number (for example, "11.10")
All	**ramdisk_id**	The ID of image stored in the Image Service that should be used as the ramdisk when booting an AMI-style image.	Valid image ID
All	**vm_mode**	The virtual machine mode. This represents the host/guest ABI (application binary interface) used for the virtual machine.	**hvm**-Fully virtualized. This is the mode used by QEMU and KVM.
libvirt API driver	**hw_disk_bus**	Specifies the type of disk controller to attach disk devices to.	**scsi**, **virtio**, **ide**, or **usb**. Note that if using **iscsi**, the **hw_scsi_model** needs to be set to **virtio-scsi**.
libvirt API driver	**hw_cdrom_bus**	Specifies the type of disk controller to attach CD-ROM devices to.	**scsi**, **virtio**, **ide**, or **usb**. If you specify **iscsi**, you must set the **hw_scsi_model** parameter to **virtio-scsi**.
libvirt API driver	**hw_numa_nodes**	Number of NUMA nodes to expose to the instance (does not override flavor definition).	Integer.
libvirt API driver	**hw_numa_cpus.0**	Mapping of vCPUs N-M to NUMA node 0 (does not override flavor definition).	Comma-separated list of integers.
libvirt API driver	**hw_numa_cpus.1**	Mapping of vCPUs N-M to NUMA node 1 (does not override flavor definition).	Comma-separated list of integers.
libvirt API driver	**hw_numa_mem.0**	Mapping N MB of RAM to NUMA node 0 (does not override flavor definition).	Integer
libvirt API driver	**hw_numa_mem.1**	Mapping N MB of RAM to NUMA node 1 (does not override flavor definition).	Integer

Specific to	Key	Description	Supported values
libvirt API driver	**hw_qemu_guest_agent**	Guest agent support. If set to **yes**, and if **qemu-ga** is also installed, file systems can be quiesced (frozen) and snapshots created automatically.	**yes** / **no**
libvirt API driver	**hw_rng_model**	Adds a random number generator (RNG) device to instances launched with this image. The instance flavor enables the RNG device by default. To disable the RNG device, the cloud administrator must set **hw_rng:allowed** to **False** on the flavor. The default entropy source is **/dev/random**. To specify a hardware RNG device, set **rng_dev_path** to **/dev/hwrng** in your Compute environment file.	**virtio**, or other supported device.
libvirt API driver	**hw_scsi_model**	Enables the use of VirtIO SCSI (virtio-scsi) to provide block device access for compute instances; by default, instances use VirtIO Block (virtio-blk). VirtIO SCSI is a para-virtualized SCSI controller device that provides improved scalability and performance, and supports advanced SCSI hardware.	**virtio-scsi**

Specific to	Key	Description	Supported values
libvirt API driver	**hw_video_model**	The video device driver for the display device to use in virtual machine instances.	Set to one of the following values to specify the supported driver to use: • **virtio** - **Recommended** Driver for the virtual machine display device, supported by most architectures. The VirtIO GPU driver is included in RHEL-7 and later, and Linux kernel versions 4.4 and later. If an instance kernel has the VirtIO GPU driver, then the instance can use all the VirtIO GPU features. If an instance kernel does not have the VirtIO GPU driver, the VirtIO GPU device gracefully falls back to VGA compatibility mode, which provides a working display for the instance. • **qxl** - **Deprecated** Driver for Spice or noVNC environments that is no longer maintained. • **cirrus** - Legacy driver. • **vga** - Use this driver for IBM Power environments. • **gop** - Not supported for QEMU/KVM environments. • **xen** - Not supported for KVM environments. • **vmvga** - Legacy driver, do not use. • **none** - Use this value to disable emulated graphics or video in virtual GPU (vGPU) instances where the driver is configured separately.
libvirt API driver	**hw_video_ram**	Maximum RAM for the video image. Used only if a **hw_video:ram_max_mb** value has been set in the flavor's **extra_specs** and that value is higher than the value set in **hw_video_ram**.	Integer in MB (for example, *64*)

Specific to	Key	Description	Supported values
libvirt API driver	**hw_watchdog_action**	Enables a virtual hardware watchdog device that carries out the specified action if the server hangs. The watchdog uses the i6300esb device (emulating a PCI Intel 6300ESB). If **hw_watchdog_action** is not specified, the watchdog is disabled.	• disabled-The device is not attached. Allows the user to disable the watchdog for the image, even if it has been enabled using the image's flavor. The default value for this parameter is disabled. • reset-Forcefully reset the guest. • poweroff-Forcefully power off the guest. • pause-Pause the guest. • none-Only enable the watchdog; do nothing if the server hangs.
libvirt API driver	**os_command_line**	The kernel command line to be used by the libvirt driver, instead of the default. For Linux Containers (LXC), the value is used as arguments for initialization. This key is valid only for Amazon kernel, ramdisk, or machine images (aki, ari, or ami).	
libvirt API driver and VMware API driver	**hw_vif_model**	Specifies the model of virtual network interface device to use.	The valid options depend on the configured hypervisor. • KVM and QEMU: e1000, ne2k_pci, pcnet, rtl8139, and virtio. • VMware: e1000, e1000e, VirtualE1000, VirtualE1000e, VirtualPCNet32, VirtualSriovEthernetCard, and VirtualVmxnet. • Xen: e1000, netfront, ne2k_pci, pcnet, and rtl8139.
VMware API driver	**vmware_adaptertype**	The virtual SCSI or IDE controller used by the hypervisor.	**lsiLogic**, **busLogic**, or **ide**

Specific to	Key	Description	Supported values
VMware API driver	**vmware_ostype**	A VMware GuestID which describes the operating system installed in the image. This value is passed to the hypervisor when creating a virtual machine. If not specified, the key defaults to **otherGuest**.	For more information, see Images with VMware vSphere.
VMware API driver	**vmware_image_version**	Currently unused.	**1**
XenAPI driver	**auto_disk_config**	If true, the root partition on the disk is automatically resized before the instance boots. This value is only taken into account by the Compute service when using a Xen-based hypervisor with the XenAPI driver. The Compute service will only attempt to resize if there is a single partition on the image, and only if the partition is in **ext3** or **ext4** format.	**true** / **false**
libvirt API driver and XenAPI driver	**os_type**	The operating system installed on the image. The XenAPI driver contains logic that takes different actions depending on the value of the **os_type** parameter of the image. For example, for **os_type=windows** images, it creates a FAT32-based swap partition instead of a Linux swap partition, and it limits the injected host name to less than 16 characters.	**linux** or **windows**