Red Hat OpenStack Platform 8
Storage Guide

Understanding, using, and managing persistent storage in OpenStack

OpenStack Team

Red Hat OpenStack Platform 8 Storage Guide

Understanding, using, and managing persistent storage in OpenStack

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide details the different procedures for using and managing persistent storage in a Red Hat OpenStack Platform environment. It also includes procedures for configuring and managing the respective OpenStack service of each persistent storage type.

Table of Contents

PREFACE

Red Hat OpenStack Platform (Red Hat OpenStack Platform) provides the foundation to build a private or public Infrastructure-as-a-Service (IaaS) cloud on top of Red Hat Enterprise Linux. It offers a massively scalable, fault-tolerant platform for the development of cloud-enabled workloads.

This guide discusses procedures for creating and managing persistent storage. Within OpenStack, this storage is provided by three main services:

- Block Storage (**openstack-cinder**)
- Object Storage (**openstack-swift**)
- Shared File System Storage (**openstack-manila**), currently a Technology Preview

These services provide different types of persistent storage, each with its own set of advantages in different use cases. This guide discusses the suitability of each for general enterprise storage requirements.

You can manage cloud storage using either the OpenStack dashboard or the command-line clients. Most procedures can be carried out using either method; some of the more advanced procedures can only be executed on the command line. This guide provides procedures for the dashboard where possible.

Note

For the complete suite of documentation for Red Hat OpenStack Platform, see Red Hat OpenStack Platform Documentation.

CHAPTER 1. INTRODUCTION TO PERSISTENT STORAGE IN OPENSTACK

OpenStack recognizes two types of storage: *ephemeral* and *persistent*. Ephemeral storage is storage that is associated only to a specific Compute instance. Once that instance is terminated, so is its ephemeral storage. This type of storage is useful for basic runtime requirements, such as storing the instance's operating system.

Persistent storage, on the other hand, is designed to survive ("persist") independent of any running instance. This storage is used for any data that needs to be reused, either by different instances or beyond the life of a specific instance. OpenStack uses the following types of persistent storage:

Volumes

The OpenStack Block Storage service (**openstack-cinder**) allows users to access block storage devices through *volumes*. Users can attach volumes to instances in order to augment their ephemeral storage with general-purpose persistent storage. Volumes can be detached and re-attached to instances at will, and can only be accessed through the instance they are attached to.

Volumes also provide inherent redundancy and disaster recovery through backups and snapshots. In addition, you can also encrypt volumes for added security. For more information about volumes, see Chapter 2, *Block Storage and Volumes*.

Note

Instances can also be configured to use absolutely no ephemeral storage. In such cases, the Block Storage service can write images to a volume; in turn, the volume can be used as a bootable root volume for an instance.

Containers

The OpenStack Object Storage service (**openstack-swift**) provides a fully-distributed storage solution used to store any kind of static data or binary **object**, such as media files, large datasets, and disk images. The Object Storage service organizes these objects through **containers**.

While a volume's contents can only be accessed through instances, the objects inside a container can be accessed through the Object Storage REST API. As such, the Object Storage service can be used as a repository by nearly every service within the cloud. For example, the Data Processing service (**openstack-sahara**) can manage all of its binaries, data input, data output, and templates directly through the Object Storage service.

Shares

The OpenStack Shared File System service (**openstack-manila**) provides the means to easily provision remote, shareable file systems, or *shares*. Shares allow tenants within the cloud to openly share storage, and can be consumed by multiple instances simultaneously.

Important

The OpenStack Shared File System service is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Each storage type is designed to address specific storage requirements. Containers are designed for wide access, and as such feature the highest throughput, access, and fault tolerance among all storage types. Container usage is geared more towards services.

On the other hand, volumes are used primarily for instance consumption. They do not enjoy the same level of access and performance as containers, but they do have a larger feature set and have more native security features than containers. Shares are similar to volumes in this regard, except that they can be consumed by multiple instances.

The following sections discuss each storage type's architecture and feature set in detail, within the context of specific storage criteria.

1.1. SCALABILITY AND BACK END

In general, a clustered storage solution provides greater back end scalability. For example, when using Red Hat Ceph as a Block Storage back end, you can scale storage capacity and redundancy by adding more Ceph OSD (Object Storage Daemon) nodes. Both Block Storage and Object Storage services support Red Hat Ceph as a back end.

The Block Storage service can use multiple storage solutions as discrete back ends. At the back end level, you can scale capacity by adding more back ends and restarting the service. The Block Storage service also features a large list of supported back end solutions, some of which feature additional scalability features.

By default, the Object Storage service uses the file system on configured *storage nodes*, and can use as much space as is available. The Object Storage service supports the XFS and ext4 file systems, and both can be scaled up to consume as much available underlying block storage. You can also scale capacity by adding more storage devices to the storage node.

The Shared File System service provisions shares backed by storage from a separate *storage pool*. This pool (which is typically managed by a third-party back end service) provides the share with storage at the file system level. The OpenStack Shared File System service supports NetApp, which can be configured to use a storage pool of pre-created volumes which provisioned shares can use for storage. In this deployment, scaling involves adding more volumes to the pool.

1.2. ACCESSIBILITY AND ADMINISTRATION

Volumes are consumed only through instances, and can only be attached to and mounted within one instance at a time. Users can create snapshots of volumes, which can be used for cloning or restoring a volume to a previous state (see Section 1.4, "Redundancy and Disaster Recovery"). The Block Storage service also allows you to create *volume types*, which aggregate volume settings (for example, size and back end) that can be easily invoked by users when creating new volumes. These types can be further associated with *Quality-of-Service* specifications, which allow you to create different storage tiers for users.

Like volumes, shares are consumed through instances. However, shares can be directly mounted within an instance, and do not need to be attached through the dashboard or CLI. Shares can also

be mounted by multiple instances simultaneously. The Shared File System service also supports share snapshots and cloning; you can also create *share types* to aggregate settings (similar to volume types).

Objects in a container are accessible via API, and can be made accessible to instances and services within the cloud. This makes them ideal as object repositories for services; for example, the Image service (**openstack-glance**) can store its images in containers managed by the Object Storage service.

1.3. SECURITY

The Block Storage service provides basic data security through *volume encryption*. With this, you can configure a volume type to be encrypted through a static key; the key will then be used for encrypting all volumes created from the configured volume type. See Section 2.2.5, “Encrypt Volumes with Static Keys” for more details.

Object and container security, on the other hand, is configured at the service and node level. The Object Storage service provides no native encryption for containers and objects. Rather, the Object Storage service prioritizes accessibility within the cloud, and as such relies solely on the cloud’s network security in order to protect object data.

The Shared File System service can secure shares through access restriction, whether by instance IP, user/group, or TLS certificate. In addition, some Shared File System service deployments can feature a separate *share servers* to manage the relationship between share networks and shares; some share servers support (or even require) additional network security. For example, a CIFS share server requires the deployment of an LDAP, Active Directory, or Kerberos authentication service.

1.4. REDUNDANCY AND DISASTER RECOVERY

The Block Storage service features volume backup and restoration, providing basic disaster recovery for user storage. Backups allow you to protect volume contents. On top of this, the service also supports snapshots; aside from cloning, snapshots are also useful in restoring a volume to a previous state.

In a multi-backend environment, you can also migrate volumes between back ends. This is useful if you need to take a back end offline for maintenance. Backups are typically stored in a storage back end separate from their source volumes to help protect the data. This is not possible, however, with snapshots, as snapshots are dependent on their source volumes.

Finally, the Block Storage service also features *volume replication*. This allows you to configure volumes to replicate content between each other, thereby providing basic redundancy.

Note

Volume replication is only available through specific third-party back ends and their respective drivers.

The Object Storage service provides no built-in backup features. As such, all backups must be performed at the file system or node level. The service, however, features more robust redundancy and fault tolerance; even the most basic deployment of the Object Storage service replicates objects multiple times. You can use failover features like **dm-multipath** to enhance redundancy.

The Shared File System service provides no built-in backup features for shares, but it does allow you to create snapshots for cloning and restoration.

CHAPTER 2. BLOCK STORAGE AND VOLUMES

The Block Storage service (**openstack-cinder**) manages the administration, security, scheduling, and overall management of all volumes. Volumes are used as the primary form of persistent storage for Compute instances.

2.1. BACK ENDS

By default, the Block Storage service uses an LVM back end as a repository for volumes. While this back end is suitable for test environments, we advise that you deploy a more robust back end for an Enterprise environment.

When deploying Red Hat OpenStack Platform for the environment, we recommand using the director. Doing so helps ensure the proper configuration of each service, including the Block Storage service (and, by extension, its back end). The director also has several integrated back end configurations.

Red Hat OpenStack Platform supports Red Hat Ceph and NFS as Block Storage back ends. For instructions on how to deploy either, see Director Installation and Usage. In particular:

- Ceph Storage (overview), Ceph Storage Node Requirements (requirements), and Red Hat Ceph Storage for the Overcloud (deployment instructions)
- Configuring NFS Storage

Third-Party Storage Providers

You can also configure the Block Storage service to use supported third-party storage appliances. The director includes the necessary components for easily deploying the following:

- Dell EqualLogic
- Dell Storage Center
- NetApp (for supported appliances)

Fujitsu ETERNUS is also supported as a back end, but is not yet integrated into the Director. For instructions on how to deploy a custom (non-integrated) back end, see Custom Block Storage Back End Deployment Guide.

For a complete list of supported back end appliances and drivers, see Component, Plug-In, and Driver Support in RHEL OpenStack Platform.

2.2. BLOCK STORAGE SERVICE ADMINISTRATION

The following procedures explain how to configure the Block Storage service to suit your needs. All of these procedures require administrator privileges.

2.2.1. Group Volume Settings with Volume Types

OpenStack allows you to create volume types, which allows you apply the type's associated settings. You can apply these settings during volume creation (Section 2.3.1, "Create a Volume") or even afterwards (Section 2.3.10, "Changing a Volume's Type (Volume Re-typing)"). For example, you can associate:

- Whether or not a volume is encrypted (Section 2.2.5.2, “Configure Volume Type Encryption”)
- Which back end a volume should use (Section 2.3.2, “Specify Back End for Volume Creation” and Section 2.4.2.1, “Migrating Between Back Ends”)
- Quality-of-Service (QoS) Specs

Settings are associated with volume types using key-value pairs called Extra Specs. When you specify a volume type during volume creation, the Block Storage scheduler applies these key/value pairs as settings. You can associate multiple key/value pairs to the same volume type.

Volume types provide the capability to provide different users with storage tiers. By associating specific performance, resilience, and other settings as key/value pairs to a volume type, you can map tier-specific settings to different volume types. You can then apply tier settings when creating a volume by specifying the corresponding volume type.

Note

Available and supported Extra Specs vary per volume driver. Consult your volume driver’s documentation for a list of valid Extra Specs.

2.2.1.1. List a Host Driver’s Capabilities

Available and supported Extra Specs vary per back end driver. Consult the driver’s documentation for a list of valid Extra Specs.

Alternatively, you can query the Block Storage host directly to determine which well-defined standard Extra Specs are supported by its driver. Start by logging in (through the command line) to the node hosting the Block Storage service. Then:

```
# cinder service-list
```

This command will return a list containing the host of each Block Storage service (**cinder-backup**, **cinder-scheduler**, and **cinder-volume**). For example:

```
+------------------+--------------------------+------+---------
|      Binary      |           Host           | Zone |  Status ...
+------------------+--------------------------+------+---------
|  cinder-backup   |   localhost.localdomain  | nova | enabled ...
| cinder-scheduler |   localhost.localdomain  | nova | enabled ...
|  cinder-volume   | localhost.localdomain@lvm | nova | enabled ...
+------------------+--------------------------+------+---------
```

To display the driver capabilities (and, in turn, determine the supported Extra Specs) of a Block Storage service, run:

```
# cinder get-capabilities VOLSVCHOST
```

Where *VOLSVCHOST* is the complete name of the **cinder-volume**'s host. For example:

```
# cinder get-capabilities localhost.localdomain@lvm
    +---------------------+----------------------------------------
-+
    |     Volume stats    |                      Value
```

```
|
    +--------------------+----------------------------------------
-+
    |     description    |                          None
|
    |     display_name   |                          None
|
    |    driver_version  |                         3.0.0
|
    |      namespace     |
OS::Storage::Capabilities::localhost.loc...
    |      pool_name     |                          None
|
    |   storage_protocol |                         iSCSI            |
    |     vendor_name    |                      Open Source         |
    |      visibility    |                          None
|
    | volume_backend_name |                          lvm             |
    +--------------------+----------------------------------------
-+
    +--------------------+----------------------------------------
-+
    | Backend properties |                        Value             |
    +--------------------+----------------------------------------
-+
    |    compression     |      {u'type': u'boolean',
u'description'...
    |        qos         |              {u'type': u'boolean', u'des
...
    |    replication     |      {u'type': u'boolean',
u'description'...
    | thin_provisioning  | {u'type': u'boolean', u'description': u'S...
    +--------------------+----------------------------------------
-+
```

The **Backend properties** column shows a list of Extra Spec Keys that you can set, while the **Value** column provides information on valid corresponding values.

2.2.1.2. Create and Configure a Volume Type

1. As an admin user in the dashboard, select **Admin > Volumes > Volume Types**.
2. Click **Create Volume Type**.
3. Enter the volume type name in the **Name** field.
4. Click **Create Volume Type**. The new type appears in the **Volume Types** table.
5. Select the volume type's **View Extra Specs** action.
6. Click **Create**, and specify the **Key** and **Value**. The key/value pair must be valid; otherwise, specifying the volume type during volume creation will result in an error.
7. Click **Create**. The associated setting (key/value pair) now appears in the **Extra Specs** table.

By default, all volume types are accessible to all OpenStack tenants. If you need to create volume types with restricted access, you will need to do so through the CLI. For instructions, see

Section 2.2.1.5, “Create and Configure Private Volume Types”.

Note

You can also associate a QOS Spec to the volume type. For details, refer to Section 2.2.4.2, “Associate a QOS Spec with a Volume Type”.

2.2.1.3. Edit a Volume Type

1. As an admin user in the dashboard, select **Admin > Volumes > Volume Types**.
2. In the **Volume Types** table, select the volume type’s **View Extra Specs** action.
3. On the **Extra Specs** table of this page, you can:
   - Add a new setting to the volume type. To do this, click **Create**, and specify the key/value pair of the new setting you want to associate to the volume type.
   - Edit an existing setting associated with the volume type. To do this, select the setting’s **Edit** action.
   - Delete existing settings associated with the volume type. To do this, select the extra specs' check box and click **Delete Extra Specs** in this and the next dialog screen.

2.2.1.4. Delete a Volume Type

To delete a volume type, select its corresponding check boxes from the **Volume Types** table and click **Delete Volume Types**.

2.2.1.5. Create and Configure Private Volume Types

By default, all volume types are visible to all tenants. You can override this during volume type creation and set it to **private**. To do so, you will need to set the type’s **`Is_Public`** flag to **`False`**.

Private volume types are useful for restricting access to certain volume settings. Typically, these are settings that should only be usable by specific tenants; examples include new back ends or ultra-high performance configurations that are being tested.

To create a private volume type, run:

```
# cinder --os-volume-api-version 2 type-create --is-public false VTYPE
```

+ Replace *VTYPE* with the name of the private volume type.

By default, private volume types are only accessible to their creators. However, admin users can find and view private volume types using the following command:

```
# cinder --os-volume-api-version 2 type-list --all
```

This command will list both public and private volume types, and will also include the name and ID of each one. You will need the volume type’s ID to provide access to it.

Access to a private volume type is granted at the tenant level. To grant a tenant access to a private volume type, run:

```
# cinder --os-volume-api-version 2 type-access-add --volume-type
VTYPEID --project-id TENANTID
```

Where:

- *VTYPEID* is the ID of the private volume type.
- *TENANTID* is the ID of the project/tenant you are granting access to *VTYPEID*.

To view which tenants have access to a private volume type, run:

```
# cinder --os-volume-api-version 2 type-access-list --volume-type VTYPE
```

To remove a tenant from the access list of a private volume type, run:

```
# cinder --os-volume-api-version 2 type-access-remove --volume-type
VTYPE --project-id TENANTID
```

Note

By default, only users with administrative privileges can create, view, or configure access for private volume types.

2.2.2. Create and Configure an Internal Tenant for the Block Storage Service

Some Block Storage features (for example, the Image-Volume cache) require the configuration of an *internal tenant*. The Block Storage service uses this tenant to manage block storage items that do not necessarily need to be exposed to normal users. Examples of such items are images cached for frequent volume cloning or temporary copies of volumes being migrated.

To configure an internal tenant, first create a generic tenant and user, both named **cinder-internal**. To do so, log in to the Controller node and run:

```
# keystone tenant-create --name cinder-internal --enabled true --
description "Block Storage Internal Tenant"
    +-------------+----------------------------------+
    |   Property  |              Value               |
    +-------------+----------------------------------+
    | description |  Block Storage Internal Tenant   |
    |   enabled   |               True               |
    |      id     | cb91e1fe446a45628bb2b139d7dccaef |
    |     name    |         cinder-internal          |
    +-------------+----------------------------------+
# keystone user-create --name cinder-internal
    +----------+----------------------------------+
    | Property |              Value               |
    +----------+----------------------------------+
    |  email   |                                  |
    | enabled  |               True               |
    |    id    | 84e9672c64f041d6bfa7a930f558d946 |
    |   name   |         cinder-internal          |
    | username |         cinder-internal          |
    +----------+----------------------------------+
```

Note that creating the tenant and user will display their respective IDs. Configure the Block Storage service to use both tenant and user as the internal tenant through their IDs. To do so, run the following on each Block Storage node:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
cinder_internal_tenant_project_id TENANTID
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
cinder_internal_tenant_user_id USERID
```

Replace *TENANTID* and *USERID* with the respective IDs of the tenant and user created through **keystone**. For example, using the IDs supplied above:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
cinder_internal_tenant_project_id cb91e1fe446a45628bb2b139d7dccaef
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
cinder_internal_tenant_user_id 84e9672c64f041d6bfa7a930f558d946
```

2.2.3. Configure and Enable the Image-Volume Cache

The Block Storage service features an optional *Image-Volume cache* which can be used when creating volumes from images. This cache is designed to improve the speed of volume creation from frequently-used images. For information on how to create volumes from images, see Section 2.3.1, “Create a Volume”.

When enabled, the Image-Volume cache stores a copy of an image the first time a volume is created from it. This stored image is cached locally to the Block Storage back end to help improve performance the next time the image is used to create a volume. The Image-Volume cache’s limit can be set to a size (in GB), number of images, or both.

The Image-Volume cache is supported by several back ends. If you are using a third-party back end, refer to its documentation for information on Image-Volume cache support.

Note

The Image-Volume cache requires that an *internal tenant* be configured for the Block Storage service. For instructions, see Section 2.2.2, “Create and Configure an Internal Tenant for the Block Storage Service”.

To enable and configure the Image-Volume cache on a Block Storage node, run the following commands:

```
# openstack-config --set /etc/cinder/cinder.conf BACKEND
image_volume_cache_enabled True
```

Replace *BACKEND* with the name of the target back end (specifically, its **volume_backend_name** value).

By default, the Image-Volume cache size is only limited by the back end. To configure a maximum size (*MAXSIZE*, in GB):

```
# openstack-config --set /etc/cinder/cinder.conf BACKEND
image_volume_cache_max_size_gb MAXSIZE
```

Alternatively, you can also set a maximum number of images (*MAXNUMBER*). To do so:

```
# openstack-config --set /etc/cinder/cinder.conf BACKEND
image_volume_cache_max_count MAXNUMBER
```

The Block Storage service database uses a time stamp to track when each cached image was last used to create an image. If either or both *MAXSIZE* and *MAXNUMBER* are set, the Block Storage service will delete cached images as needed to make way for new ones. Cached images with the oldest time stamp are deleted first whenever the Image-Volume cache limits are met.

After configuring the Image-Volume cache, restart the Block Storage service:

```
# openstack-service restart cinder
```

2.2.4. Use Quality-of-Service Specifications

You can map multiple performance settings to a single Quality-of-Service specification (QOS Specs). Doing so allows you to provide performance tiers for different user types.

Performance settings are mapped as key/value pairs to QOS Specs, similar to the way volume settings are associated to a volume type. However, QOS Specs are different from volume types in the following respects:

- QOS Specs are used to apply performance settings, which include limiting read/write operations to disks. Available and supported performance settings vary per storage driver.

  To determine which QOS Specs are supported by your back end, consult the documentation of your back end device's volume driver.

- Volume types are directly applied to volumes, whereas QOS Specs are not. Rather, QOS Specs are associated to volume types. During volume creation, specifying a volume type also applies the performance settings mapped to the volume type's associated QOS Specs.

2.2.4.1. Create and Configure a QOS Spec

As an administrator, you can create and configure a QOS Spec through the QOS Specs table. You can associate more than one key/value pair to the same QOS Spec.

1. As an admin user in the dashboard, select **Admin > Volumes > Volume Types**.
2. On the **QOS Specs** table, click **Create QOS Spec**.
3. Enter a name for the **QOS Spec**.
4. In the **Consumer** field, specify where the QOS policy should be enforced:

Table 2.1. Consumer Types

Type	Description
back-end	QOS policy will be applied to the Block Storage back end.

Type	Description
`front-end`	QOS policy will be applied to Compute.
`both`	QOS policy will be applied to both Block Storage and Compute.

5. Click **Create**. The new QOS Spec should now appear in the **QOS Specs** table.
6. In the **QOS Specs** table, select the new spec's **Manage Specs** action.
7. Click **Create**, and specify the **Key** and **Value**. The key/value pair must be valid; otherwise, specifying a volume type associated with this QOS Spec during volume creation will fail.
8. Click **Create**. The associated setting (key/value pair) now appears in the **Key-Value Pairs** table.

2.2.4.2. Associate a QOS Spec with a Volume Type

As an administrator, you can associate a QOS Spec to an existing volume type using the **Volume Types** table.

1. As an administrator in the dashboard, select **Admin > Volumes > Volume Types**.
2. In the **Volume Types** table, select the type's **Manage QOS Spec Association** action.
3. Select a QOS Spec from the **QOS Spec to be associated** list.
4. Click **Associate**. The selected QOS Spec now appears in the **Associated QOS Spec** column of the edited volume type.

2.2.4.3. Disassociate a QOS Spec from a Volume Type

1. As an administrator in the dashboard, select **Admin > Volumes > Volume Types**.
2. In the **Volume Types** table, select the type's **Manage QOS Spec Association** action.
3. Select **None** from the QOS Spec to be associated list.
4. Click **Associate**. The selected QOS Spec is no longer in the **Associated QOS Spec** column of the edited volume type.

2.2.5. Encrypt Volumes with Static Keys

Volume encryption helps provide basic data protection in case the volume back-end is either compromised or outright stolen. The contents of an encrypted volume can only be read with the use of a specific key; both Compute and Block Storage services must be configured to use the same key in order for instances to use encrypted volumes. You can also create a specific volume type that uses encryption and all volumes created using this volume type are encrypted.

This section describes how to configure an OpenStack deployment to use a single key for encrypting volumes.

Important

At present, volume encryption is only supported on volumes backed by block devices. Encryption of network-attached volumes (such as RBD) or file-based volumes (such as NFS) is still unsupported.

2.2.5.1. Configure a Static Key

The first step in implementing basic volume encryption is to set a *static key*. This key must be a hexadecimal string, which will be used by the Block Storage volume service (namely, **`openstack-cinder-volume`**) and all Compute services (**`openstack-nova-compute`**). To configure both services to use this key, set the key as the **`fixed_key`** value in the **`[keymgr]`** section of both service's respective configuration files.

1. From the command line, log in as **`root`** to the node hosting **`openstack-cinder-volume`**.
2. Set the static key:

   ```
   # openstack-config --set /etc/cinder/cinder.conf keymgr fixed_key
   HEX_KEY
   ```

   Replace *HEX_KEY* with a 16-digit alphanumeric hexadecimal key (for example, 04d6b077d60e323711b37813b3a68a71).

   You can use the **`openssl`** command to generate the key as follows:

   ```
   # openssl rand -hex 16
   04d6b077d60e323711b37813b3a68a71
   ```

   

   Note

   This value should be secure.

3. Restart the Block Storage volume service:

   ```
   # openstack-service restart cinder-volume
   ```

4. Log in to the node hosting **`openstack-nova-compute`**, and set the same static key:

   ```
   # openstack-config --set /etc/nova/nova.conf keymgr fixed_key
   HEX_KEY
   ```

   

   Note

   If you have multiple Compute nodes (multiple nodes hosting **`openstack-nova-compute`**), then you need to set the same static key in **`/etc/nova/nova.conf`** of each node.

5. Restart the Compute service:

```
# openstack-service restart nova-compute
```

Note

Likewise, if you set the static key on multiple Compute nodes, you need to restart the **openstack-nova-compute** service on each node as well.

At this point, both Compute and Block Storage volume services can now use the same static key to encrypt/decrypt volumes. That is, new instances will be able to use volumes encrypted with the static key (**HEX_KEY**).

2.2.5.2. Configure Volume Type Encryption

To create volumes encrypted with the static key from Section 2.2.5.1, “Configure a Static Key”, you need an *encrypted volume type*. Configuring a volume type as encrypted involves setting what provider class, cipher, and key size it should use. To do so, run:

```
# cinder encryption-type-create --cipher aes-xts-plain64 --key_size
BITSIZE --control_location front-end VOLTYPE
nova.volume.encryptors.luks.LuksEncryptor
```

Where:

- *BITSIZE* is the key size (for example, **512** for a 512-bit key).
- *VOLTYPE* is the name of the volume type you want to encrypt.

This command sets the **nova.volume.encryptors.luks.LuksEncryptor** provider class and **aes-xts-plain64** cipher. As of this release, this is the only supported class/cipher configuration for volume encryption.

Once you have an encrypted volume type, you can invoke it to automatically create encrypted volumes. For more information on creating a volume type, see Section 2.2.1.2, “Create and Configure a Volume Type”. Specifically, select the encrypted volume type from the Type drop-down list in the **Create Volume** window (see to Section 2.3, “Basic Volume Usage and Configuration”).

You can view the metadata about the encryption by clicking on **Yes** in the **Encrypted** volume of a volume under **Project > Compute > Volumes**.

2.2.6. Configure How Volumes are Allocated to Multiple Back Ends

If the Block Storage service is configured to use multiple back ends, you can use configured volume types to specify where a volume should be created. For details, see Section 2.3.2, “Specify Back End for Volume Creation”.

The Block Storage service will automatically choose a back end if you do not specify one during volume creation. Block Storage sets the first defined back end as a default; this back end will be used until it runs out of space. At that point, Block Storage will set the second defined back end as a default, and so on.

If this is not suitable for your needs, you can use the filter scheduler to control how Block Storage should select back ends. This scheduler can use different filters to triage suitable back ends, such as:

AvailabilityZoneFilter

Filters out all back ends that do not meet the availability zone requirements of the requested volume

CapacityFilter

Selects only back ends with enough space to accommodate the volume

CapabilitiesFilter

Selects only back ends that can support any specified settings in the volume

To configure the filter scheduler:

1. Enable the **`FilterScheduler`**.

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
scheduler_driver
cinder.scheduler.filter_scheduler.FilterScheduler
```

2. Set which filters should be active:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
scheduler_default_filters
AvailabilityZoneFilter,CapacityFilter,CapabilitiesFilter
```

3. Configure how the scheduler should select a suitable back end. If you want the scheduler:
   - To always choose the back end with the most available free space, run:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
scheduler_default_weighers AllocatedCapacityWeigher
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
allocated_capacity_weight_multiplier -1.0
```

   - To choose randomly among all suitable back ends, run:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
scheduler_default_weighers ChanceWeigher
```

4. Restart the Block Storage scheduler to apply your settings:

```
# openstack-service restart openstack-cinder-scheduler
```

2.2.7. Backup Administration

The following sections discuss how to customize the Block Storage service's volume backup settings.

2.2.7.1. View and Modify a Tenant's Backup Quota

Unlike most tenant storage quotas (number of volumes, volume storage, snapshots, etc.), backup quotas cannot be modified through the dashboard yet.

Backup quotas can only be modified through the command-line interface; namely, through the **`cinder quota-update`** command.

To view the storage quotas of a specific tenant (*TENANTNAME*), run:

```
# cinder quota-show TENANTNAME
```

To update the maximum number of backups (*MAXNUM*) that can be created in a specific tenant, run:

```
# cinder quota-update --backups MAXNUM TENANTNAME
```

To update the maximum total size of all backups (*MAXGB*) within a specific tenant, run:

```
# cinder quota-update --backup-gigabytes MAXGB TENANTNAME
```

To view the storage quota usage of a specific tenant, run:

```
# cinder quota-usage TENANTNAME
```

2.2.7.2. Enable Volume Backup Management Through the Dashboard

You can now create, view, delete, and restore volume backups through the dashboard. To perform any of these functions, go to the **Project > Compute > Volumes > Volume Backups** tab.

However, the **Volume Backups** tab is not enabled by default. To enable it, configure the dashboard accordingly:

1. Open **`/etc/openstack-dashboard/local_settings`**.
2. Search for the following setting:

   ```
   OPENSTACK_CINDER_FEATURES = {
       'enable_backup': False,
   }
   ```

   Change this setting to:

   ```
   OPENSTACK_CINDER_FEATURES = {
       'enable_backup': True,
   }
   ```

3. Restart the dashboard by restarting the **`httpd`** service:

   ```
   # systemctl restart httpd.service
   ```

2.2.7.3. Set an NFS Share as a Backup Repository

By default, the Block Storage service uses the Object Storage service as a repository for backups. You can configure the Block Storage service to use an existing NFS share as a backup repository instead. To do so:

1. Log in to the node hosting the backup service (**openstack-cinder-backup**) as a user with administrative privileges.
2. Configure the Block Storage service to use the NFS backup driver (cinder.backup.drivers.nfs):

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
backup_driver cinder.backup.drivers.nfs
```

3. Set the details of the NFS share that you want to use as a backup repository:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
backup_share NFSHOST:PATH
```

   Where:

   - *NFSHOST* is the IP address or hostname of the NFS server.
   - *PATH* is the absolute path of the NFS share on *NFSHOST*.

4. If you want to set any optional mount settings for the NFS share, run:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
backup_mount_options NFSMOUNTOPTS
```

   Where *NFSMOUNTOPTS* is a comma-separated list of NFS mount options (for example, rw,sync). For more information on supported mount options, see the **man** pages for **nfs** and **mount**.

5. Restart the Block Storage backup service to apply your changes:

```
# systemctl restart openstack-cinder-backup.service
```

2.2.7.3.1. Set a Different Backup File Size

The backup service limits backup files sizes to a maximum **backup file size**. If you are backing up a volume that exceeds this size, the resulting backup will be split into multiple chunks. The default backup file size is 1.8GB.

To set a different backup file size, run:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
backup_file_size SIZE
```

Replace *SIZE* with the file size you want, in bytes. Restart the Block Storage backup service to apply your changes:

```
# systemctl restart openstack-cinder-backup.service
```

2.3. BASIC VOLUME USAGE AND CONFIGURATION

The following procedures describe how to perform basic end-user volume management. These procedures do not require administrative privileges.

2.3.1. Create a Volume

1. In the dashboard, select **Project > Compute > Volumes**.
2. Click **Create Volume**, and edit the following fields:

Field	Description
Volume name	Name of the volume.
Description	Optional, short description of the volume.
Type	Optional volume type (see Section 2.2.1, “Group Volume Settings with Volume Types”). If you have multiple Block Storage back ends, you can use this to select a specific back end. See Section 2.3.2, “Specify Back End for Volume Creation” for details.
Size (GB)	Volume size (in gigabytes).
Availability Zone	Availability zones (logical server groups), along with host aggregates, are a common method for segregating resources within OpenStack. Availability zones are defined during installation. For more information on availability zones and host aggregates, see **Manage Host Aggregates** in the **Instances and Images Guide** available at Red Hat OpenStack Platform.

3. Specify a **Volume Source**:

Source	Description
No source, empty volume	The volume will be empty, and will not contain a file system or partition table.
Snapshot	Use an existing snapshot as a volume source. If you select this option, a new Use snapshot as a source list appears; you can then choose a snapshot from the list. For more information about volume snapshots, refer to Section 2.3.8, “Create, Use, or Delete Volume Snapshots”.

Source	Description
Image	Use an existing image as a volume source. If you select this option, a new Use image as a source lists appears; you can then choose an image from the list.
Volume	Use an existing volume as a volume source. If you select this option, a new Use volume as a source list appears; you can then choose a volume from the list.

4. Click **Create Volume**. After the volume is created, its name appears in the **Volumes** table.

You can also change the volume's type later on. For details, see Section 2.3.10, "Changing a Volume's Type (Volume Re-typing)".

2.3.2. Specify Back End for Volume Creation

Whenever multiple Block Storage back ends are configured, you will also need to create a volume type for each back end. You can then use the type to specify which back end should be used for a created volume. For more information about volume types, see Section 2.2.1, "Group Volume Settings with Volume Types".

To specify a back end when creating a volume, select its corresponding volume type from the Type drop-down list (see Section 2.3.1, "Create a Volume").

If you do not specify a back end during volume creation, the Block Storage service will automatically choose one for you. By default, the service will choose the back end with the most available free space. You can also configure the Block Storage service to choose randomly among all available back ends instead; for more information, see Section 2.2.6, "Configure How Volumes are Allocated to Multiple Back Ends".

2.3.3. Edit a Volume's Name or Description

1. In the dashboard, select **Project > Compute > Volumes**.
2. Select the volume's **Edit Volume** button.
3. Edit the volume name or description as required.
4. Click **Edit Volume** to save your changes.

Note

To create an encrypted volume, you must first have a volume type configured specifically for volume encryption. In addition, both Compute and Block Storage services must be configured to use the same static key. For information on how to set up the requirements for volume encryption, refer to Section 2.2.5, “Encrypt Volumes with Static Keys”.

2.3.4. Delete a Volume

1. In the dashboard, select **Project > Compute > Volumes**.
2. In the **Volumes** table, select the volume to delete.
3. Click **Delete Volumes**.

Note

A volume cannot be deleted if it has existing snapshots. For instructions on how to delete snapshots, see Section 2.3.8, “Create, Use, or Delete Volume Snapshots”.

2.3.5. Attach and Detach a Volume to an Instance

Instances can use a volume for persistent storage. A volume can only be attached to one instance at a time. For more information on instances, see **Manage Instances** in the **Instances and Images Guide** available at Red Hat OpenStack Platform.

2.3.5.1. Attach a Volume to an Instance

1. In the dashboard, select **Project > Compute > Volumes**.
2. Select the volume’s **Edit Attachments** action. If the volume is not attached to an instance, the **Attach To Instance** drop-down list is visible.
3. From the **Attach To Instance** list, select the instance to which you wish to attach the volume.
4. Click **Attach Volume**.

2.3.5.2. Detach a Volume From an Instance

1. In the dashboard, select **Project > Compute > Volumes**.
2. Select the volume’s **Manage Attachments** action. If the volume is attached to an instance, the instance’s name is displayed in the **Attachments** table.
3. Click **Detach Volume** in this and the next dialog screen.

2.3.6. Set a Volume to Read-Only

You can give multiple users shared access to a single volume without allowing them to edit its contents. To do so, set the volume to **read-only** using the following command:

```
# cinder readonly-mode-update VOLUME true
```

Replace **VOLUME** with the **ID** of the target volume.

To set a read-only volume back to read-write, run:

```
# cinder readonly-mode-update VOLUME false
```

2.3.7. Change a Volume's Owner

To change a volume's owner, you will have to perform a volume transfer. A volume transfer is initiated by the volume's owner, and the volume's change in ownership is complete after the transfer is accepted by the volume's new owner.

2.3.7.1. Transfer a Volume from the Command Line

1. Log in as the volume's current owner.
2. List the available volumes:

   ```
   # cinder list
   ```

3. Initiate the volume transfer:

   ```
   # cinder transfer-create VOLUME
   ```

   Where **VOLUME** is the name or **ID** of the volume you wish to transfer. For example,

   ```
   +------------+--------------------------------------+
   |  Property  |                Value                 |
   +------------+--------------------------------------+
   |  auth_key  |           f03bf51ce7ead189           |
   | created_at |      2014-12-08T03:46:31.884066      |
   |     id     | 3f5dc551-c675-4205-a13a-d30f88527490 |
   |    name    |                 None                 |
   | volume_id  | bcf7d015-4843-464c-880d-7376851ca728 |
   +------------+--------------------------------------+
   ```

   The **cinder transfer-create** command clears the ownership of the volume and creates an **id** and **auth_key** for the transfer. These values can be given to, and used by, another user to accept the transfer and become the new owner of the volume.

4. The new user can now claim ownership of the volume. To do so, the user should first log in from the command line and run:

   ```
   # cinder transfer-accept TRANSFERID TRANSFERKEY
   ```

   Where **TRANSFERID** and **TRANSFERKEY** are the **id** and **auth_key** values returned by the **cinder transfer-create** command, respectively. For example,

```
# cinder transfer-accept 3f5dc551-c675-4205-a13a-d30f88527490
f03bf51ce7ead189
```

Note

You can view all available volume transfers using:

```
# cinder transfer-list
```

2.3.7.2. Transfer a Volume Using the Dashboard

Create a volume transfer from the dashboard

1. As the volume owner in the dashboard, select **Projects > Volumes**.
2. In the **Actions** column of the volume to transfer, select **Create Transfer**.
3. In the **Create Transfer** dialog box, enter a name for the transfer and click **Create Volume Transfer**.

   The volume transfer is created and in the **Volume Transfer** screen you can capture the **`transfer ID`** and the **`authorization key`** to send to the recipient project.

   

   Note

   The authorization key is available only in the **Volume Transfer** screen. If you lose the authorization key, you must cancel the transfer and create another transfer to generate a new authorization key.

4. Close the **Volume Transfer** screen to return to the volume list.

   The volume status changes to **`awaiting-transfer`** until the recipient project accepts the transfer

Accept a volume transfer from the dashboard

1. As the recipient project owner in the dashboard, select **Projects > Volumes**.
2. Click **Accept Transfer**.
3. In the **Accept Volume Transfer** dialog box, enter the **`transfer ID`** and the **`authorization key`** that you received from the volume owner and click **Accept Volume Transfer**.

   The volume now appears in the volume list for the active project.

2.3.8. Create, Use, or Delete Volume Snapshots

You can preserve a volume's state at a specific point in time by creating a volume snapshot. You can then use the snapshot to clone new volumes.

Note

Volume backups are different from snapshots. Backups preserve the data contained in the volume, whereas snapshots preserve the state of a volume at a specific point in time. In addition, you cannot delete a volume if it has existing snapshots. Volume backups are used to prevent data loss, whereas snapshots are used to facilitate cloning.

For this reason, snapshot back ends are typically co-located with volume back ends in order to minimize latency during cloning. By contrast, a backup repository is usually located in a different location (eg. different node, physical storage, or even geographical location) in a typical enterprise deployment. This is to protect the backup repository from any damage that might occur to the volume back end.

For more information about volume backups, refer to Section 2.4.1, “Back Up and Restore a Volume”

To create a volume snapshot:

1. In the dashboard, select **Project > Compute > Volumes**.
2. Select the target volume’s **Create Snapshot** action.
3. Provide a **Snapshot Name** for the snapshot and click **Create a Volume Snapshot**. The **Volume Snapshots** tab displays all snapshots.

You can clone new volumes from a snapshot once it appears in the **Volume Snapshots** table. To do so, select the snapshot’s **Create Volume** action. For more information about volume creation, see Section 2.3.1, “Create a Volume”.

To delete a snapshot, select its **Delete Volume Snapshot** action.

If your OpenStack deployment uses a Red Hat Ceph back end, see Section 2.3.8.1, “Protected and Unprotected Snapshots in a Red Hat Ceph Back End” for more information on snapshot security and troubleshooting.

2.3.8.1. Protected and Unprotected Snapshots in a Red Hat Ceph Back End

When using Red Hat Ceph as a back end for your OpenStack deployment, you can set a snapshot to *protected* in the back end. Attempting to delete protected snapshots through OpenStack (as in, through the dashboard or the **`cinder snapshot-delete`** command) will fail.

When this occurs, set the snapshot to *unprotected* in the Red Hat Ceph back end first. Afterwards, you should be able to delete the snapshot through OpenStack as normal.

For related instructions, see Protecting a Snapshot and Unprotecting a Snapshot.

2.3.9. Upload a Volume to the Image Service

You can upload an existing volume as an image to the Image service directly. To do so:

1. In the dashboard, select **Project > Compute > Volumes**.
2. Select the target volume’s **Upload to Image** action.
3. Provide an **Image Name** for the volume and select a **Disk Format** from the list.

4. Click **Upload**. The QEMU disk image utility uploads a new image of the chosen format using the name you provided.

To view the uploaded image, select **Project > Compute > Images**. The new image appears in the **Images** table. For information on how to use and configure images, see **Manage Images** in the **Instances and Images Guide** available at Red Hat OpenStack Platform.

2.3.10. Changing a Volume's Type (Volume Re-typing)

Volume re-typing is the process of applying a volume type (and, in turn, its settings) to an already existing volume. For more information about volume types, see Section 2.2.1, "Group Volume Settings with Volume Types".

A volume can be re-typed whether or not it has an existing volume type. In either case, a re-type will only be successful if the Extra Specs of the volume type can be applied to the volume. Volume re-typing is useful for applying pre-defined settings or storage attributes to an existing volume, such as when you want to:

- Migrate the volume to a different back end (Section 2.4.2.1, "Migrating Between Back Ends").
- Change the volume's storage class/tier.

Users with no administrative privileges can only re-type volumes they own. To perform a volume re-type:

1. In the dashboard, select **Project > Compute > Volumes**.
2. In the **Actions** column of the volume to be migrated, select **Change Volume Type**.
3. In the **Change Volume Type** dialog, select the target volume type defining the new back end from the **Type** drop-down list.

   

   Note

   If you are migrating the volume to another back end, select **On Demand** from the **Migration Policy** drop-down list. For more information, see Section 2.4.2.1, "Migrating Between Back Ends".

4. Click **Change Volume Type** to start the migration.

2.4. ADVANCED VOLUME CONFIGURATION

The following procedures describe how to perform advanced volume management procedures.

2.4.1. Back Up and Restore a Volume

A volume backup is a persistent copy of a volume's contents. Volume backups are typically created as object stores, and are managed through the Object Storage service by default. You can, however, set up a different repository for your backups; OpenStack supports Red Hat Ceph and NFS as alternative back ends for backups.

When creating a volume backup, all of the backup's metadata is stored in the Block Storage service's database. The `cinder` utility uses this metadata when restoring a volume from the

backup. As such, when recovering from a catastrophic database loss, you must restore the Block Storage service's database first before restoring any volumes from backups. This also presumes that the Block Storage service database is being restored with all the original volume backup metadata intact.

If you wish to configure only a subset of volume backups to survive a catastrophic database loss, you can also export the backup's metadata. In doing so, you can then re-import the metadata to the Block Storage database later on, and restore the volume backup as normal.

Note

Volume backups are different from snapshots. Backups preserve the data contained in the volume, whereas snapshots preserve the state of a volume at a specific point in time. In addition, you cannot delete a volume if it has existing snapshots. Volume backups are used to prevent data loss, whereas snapshots are used to facilitate cloning.

For this reason, snapshot back ends are typically co-located with volume back ends in order to minimize latency during cloning. By contrast, a backup repository is usually located in a different location (eg. different node, physical storage, or even geographical location) in a typical enterprise deployment. This is to protect the backup repository from any damage that might occur to the volume back end.

For more information about volume snapshots, refer to Section 2.3.8, "Create, Use, or Delete Volume Snapshots".

2.4.1.1. Create a Full Volume Backup

To back up a volume, use the **`cinder backup-create`** command. By default, this command will create a full backup of the volume. If the volume has existing backups, you can choose to create an **incremental** backup instead (see Section 2.4.1.2, "Create an Incremental Volume Backup" for details.)

You can create backups of volumes you have access to. This means that users with administrative privileges can back up any volume, regardless of owner. For more information, see Section 2.4.1.1.1, "Create a Volume Backup as an Admin".

1. View the **`ID`** or **`Display Name`** of the volume you wish to back up:

   ```
   # cinder list
   ```

2. Back up the volume:

   ```
   # cinder backup-create VOLUME
   ```

   Replace *VOLUME* with the **`ID`** or **`Display Name`** of the volume you want to back up. For example:

   ```
   +-----------+--------------------------------------+
   |  Property |                Value                 |
   +-----------+--------------------------------------+
   |     id    | e9d15fc7-eeae-4ca4-aa72-d52536dc551d |
   |    name   |                 None                 |
   | volume_id | 5f75430a-abff-4cc7-b74e-f808234fa6c5 |
   +-----------+--------------------------------------+
   ```

Note

The **volume_id** of the resulting backup is identical to the **ID** of the source volume.

3. Verify that the volume backup creation is complete:

```
# cinder backup-list
```

The volume backup creation is complete when the **Status** of the backup entry is **available**.

At this point, you can also export and store the volume backup's metadata. This allows you to restore the volume backup, even if the Block Storage database suffers a catastrophic loss. To do so, run:

```
# cinder --os-volume-api-version 2 backup-export BACKUPID
```

Where *BACKUPID* is the ID or name of the volume backup. For example,

```
+----------------+-----------------------------------------+
|    Property    |                Value                    |
+----------------+-----------------------------------------+
| backup_service |     cinder.backup.drivers.swift         |
|   backup_url   | eyJzdGF0dXMiOiAiYXZhaWxhYmxlIiwgIm9iam...|
|                | ...4NS02ZmY4MzBhZWYwNWUiLCAic2l6ZSI6IDF9 |
+----------------+-----------------------------------------+
```

The volume backup metadata consists of the **backup_service** and **backup_url** values.

2.4.1.1.1. Create a Volume Backup as an Admin

Users with administrative privileges (such as the default **admin** account) can back up any volume managed by OpenStack. When an admin backs up a volume owned by a non-admin user, the backup is hidden from the volume owner by default.

As an admin, you can still back up a volume **and** make the backup available to a specific tenant. To do so, run:

```
# cinder --os-auth-url KEYSTONEURL --os-tenant-name TENANTNAME --os-
username USERNAME --os-password PASSWD backup-create VOLUME
```

Where:

- *TENANTNAME* is the name of the tenant where you want to make the backup available.
- *USERNAME* and *PASSWD* are the username/password credentials of a user within *TENANTNAME*.
- *VOLUME* is the name or ID of the volume you want to back up.
- *KEYSTONEURL* is the URL endpoint of the Identity service (typically http://*IP*:5000/v2, where *IP* is the IP address of the Identity service host).

When performing this operation, the resulting backup's size will count against the quota of *TENANTNAME* rather than the admin's tenant.

2.4.1.2. Create an Incremental Volume Backup

By default, the **`cinder backup-create`** command will create a full backup of a volume. However, if the volume has existing backups, you can choose to create an **incremental** backup.

An incremental backup captures any changes to the volume since the last backup (full or incremental). Performing numerous, regular, full back ups of a volume can become resource-intensive as the volume's size increases over time. In this regard, incremental backups allow you to capture periodic changes to volumes while minimizing resource usage.

To create an incremental volume backup, use the **`--incremental`** option. As in:

```
# cinder backup-create VOLUME --incremental
```

Replace *VOLUME* with the **`ID`** or **`Display Name`** of the volume you want to back up. Incremental backups are fully supported on NFS and Object Storage backup repositories.

Note

You cannot delete a full backup if it already has an incremental backup. In addition, if a full backup has multiple incremental backups, you can only delete the latest one.

Warning

When using Red Hat Ceph Storage as a back end for both Block Storage (cinder) volumes and backups, any attempt to perform an incremental backup will result in a full backup instead, without any warning. This is a known issue (BZ#1463061).

2.4.1.3. Restore a Volume After a Block Storage Database Loss

Typically, a Block Storage database loss prevents you from restoring a volume backup. This is because the Block Storage database contains metadata required by the volume backup service (openstack-cinder-backup). This metadata consists of **`backup_service`** and **`backup_url`** values, which you can export after creating the volume backup (as shown in Section 2.4.1.1, "Create a Full Volume Backup").

If you exported and stored this metadata, then you can import it to a new Block Storage database (thereby allowing you to restore the volume backup).

1. As a user with administrative privileges, run:

   ```
   # cinder --os-volume-api-version 2 backup-import backup_service backup_url
   ```

   Where *backup_service* and *backup_url* are from the metadata you exported. For example, using the exported metadata from Section 2.4.1.1, "Create a Full Volume Backup":

```
# cinder --os-volume-api-version 2 backup-import
cinder.backup.drivers.swift eyJzdGF0dXMi...c2l6ZSI6IDF9
+----------+--------------------------------------+
| Property |                Value                 |
+----------+--------------------------------------+
|    id    | 77951e2f-4aff-4365-8c64-f833802eaa43 |
|   name   |                 None                 |
+----------+--------------------------------------+
```

2. After the metadata is imported into the Block Storage service database, you can restore the volume as normal (see Section 2.4.1.4, “Restore a Volume from a Backup”).

2.4.1.4. Restore a Volume from a Backup

1. Find the **ID** of the volume backup you wish to use:

   ```
   # cinder backup-list
   ```

   The **Volume ID** should match the ID of the volume you wish to restore.

2. Restore the volume backup:

   ```
   # cinder backup-restore BACKUP_ID
   ```

   Where *BACKUP_ID* is the ID of the volume backup you wish to use.

3. If you no longer need the backup, delete it:

   ```
   # cinder backup-delete BACKUP_ID
   ```

2.4.2. Migrate a Volume

The Block Storage service allows you to migrate volumes between hosts or back ends. You can migrate a volume currently in-use (attached to an instance), but not volumes that have snapshots.

When migrating a volume between hosts, both hosts must reside on the same back end. To do so:

1. In the dashboard, select **Admin > Volumes**.
2. In the **Actions** column of the volume to be migrated, select **Migrate Volume**.
3. In the **Migrate Volume** dialog, select the target host from the **Destination Host** drop-down list.

   

   Note

   If you wish to bypass any driver optimizations for the host migration, select the **Force Host Copy** checkbox.

4. Click **Migrate** to start the migration.

2.4.2.1. Migrating Between Back Ends

Migrating a volume between back ends, on the other hand, involves **volume re-typing**. This means that in order to migrate to a new back end:

1. The new back end must be specified as an **Extra Spec** in a separate *target volume type*.
2. All other Extra Specs defined in the target volume type must be compatible with the volume's original volume type.

See Section 2.2.1, "Group Volume Settings with Volume Types" and Section 2.3.2, "Specify Back End for Volume Creation" for more details.

When defining the back end as an Extra Spec, use **volume_backend_name** as the **Key**. Its corresponding value will be the back end's name, as defined in the Block Storage configuration file (/etc/cinder/cinder.conf). In this file, each back end is defined in its own section, and its corresponding name is set in the **volume_backend_name** parameter.

Once you have a back end defined in a target volume type, you can migrate a volume to that back end through **re-typing**. This involves re-applying the target volume type to a volume, thereby applying the new back end settings. See Section 2.3.10, "Changing a Volume's Type (Volume Re-typing)" for instructions.

To do so:

1. In the dashboard, select **Project > Compute > Volumes**.
2. In the **Actions** column of the volume to be migrated, select **Change Volume Type**.
3. In the **Change Volume Type** dialog, select the target volume type defining the new back end from the **Type** drop-down list.
4. Select **On Demand** from the **Migration Policy** drop-down list.
5. Click **Change Volume Type** to start the migration.

CHAPTER 3. OBJECT STORAGE AND CONTAINERS

OpenStack Object Storage (**openstack-swift**) stores its objects (data) in containers, which are similar to directories in a file system although they cannot be nested. Containers provide an easy way for users to store any kind of unstructured data; for example, objects might include photos, text files, or images. Stored objects are neither encrypted nor compressed.

3.1. OBJECT STORAGE SERVICE ADMINISTRATION

The following procedures explain how to further customize the Object Storage service to suit your needs.

3.1.1. Erasure Coding for Object Storage Service

Erasure coding (EC) is a method of data protection in which the data is broken into fragments, expanded and encoded with redundant data pieces and stored across a set of different locations or storage media. It uses a smaller volume of storage to attain the required durability than traditional replication. When compared to replication factor of 3, savings of 50% may be attained with careful deployment. However, depending on the workload, erasure coding may incur a performance penalty.

With the Red Hat OpenStack Platform 8 release, erasure coding support is available as a technology preview for Object Storage service. For more information on the support scope for features marked as technology previews, refer to https://access.redhat.com/support/offerings/techpreview/.

Erasure coding is supported for Object Storage service as a Storage Policy. A Storage Policy allows segmenting the cluster for various purposes through the creation of multiple object rings. Red Hat recommends you split off devices used by erasure coding and replication Storage Policies. This way behavior of the cluster is easier to analyze.

The direction you choose depends on why the erasure coding policy is being deployed. Some of the main considerations are:

- Layout of existing infrastructure.
- Cost of adding dedicated erasure coding nodes (or just dedicated erasure coding devices).
- Intended usage model(s).

3.1.1.1. Configure Erasure Coding

To use an erasure coding policy, define an erasure coding policy in **swift.conf** file and create, configure the associated object ring. An example of how an erasure coding policy can be setup is shown below:

```
[storage-policy:2]
name = ec104
policy_type = erasure_coding
ec_type = jerasure_rs_vand
ec_num_data_fragments = 10
ec_num_parity_fragments = 4
ec_object_segment_size = 1048576
```

The following table describes the terms in the storage policy:

Term	Description
name	This is a standard storage policy parameter.
policy_type	Set this to erasure_coding to indicate that this is an erasure coding policy.
ec_type	Set this value according to the available options in the selected PyECLib back-end. This specifies the erasure coding scheme that is to be used. For example, the option shown here selects Vandermonde Reed-Solomon encoding while an option of flat_xor_hd_3 would select Flat-XOR based HD combination codes. See the PyECLib page for full details.
ec_num_data_fragments	The total number of fragments that will be comprised of data.
ec_num_parity_fragments	The total number of fragments that will be comprised of parity.
ec_object_segment_size	The amount of data that will be buffered up before feeding a segment into the encoder/decoder. The default value is 1048576.

When PyECLib encodes an object, it breaks it into N fragments. It is important during configuration to know how many of those fragments are data and how many are parity. So in the example above, PyECLib will break an object in 14 different fragments, 10 of them will be made up of actual object data and 4 of them will be made of parity data (calculations depending on ec_type). With such a configuration, the system can sustain 4 disk failures before the data is lost. Other commonly used configurations are 4+2 (with 4 data fragments and 2 partiy fragments) or 8+3 (with 8 data fragments and 3 parity fragments).

Note

It is important to note that once you have deployed a policy and have created objects with that policy, these configurations options cannot be changed. In case a change in the configuration is desired, you must create a new policy and migrate the data to a new container. However, once defined, policy indices cannot be discarded. If policies are to be retired, they may be disabled, but not be removed. There is essentially no performance penalty for having old policies around, but a minor administrative overhead.

3.1.1.2. Configure an Object Storage Ring

Object Storage uses a data structure called the **Ring** to distribute a partition space across the cluster. This partition space is core to the replication system in Object Storage service. It allows the Object Storage service to quickly and easily synchronize each partition across the cluster. When

any component in Swift needs to interact with data, a quick lookup is done locally in the Ring to determine the possible partitions for each replica.

Object Storage service already has three rings to store different types of data. There is one for account information, another for containers (so that it's convenient to organize objects under an account) and another for the object replicas. To support erasure codes, there will be an additional ring that is created to store erasure code chunks.

To create a typical replication ring, for example, you can use the following command:

```
swift-ring-builder object-1.builder create 10 3 1
```

where 3 is the number of replicas.

In order to create an erasure coding object ring, you need to use the number of fragments in place of the number of replicas, for example:

```
swift-ring-builder object-1.builder create 10 14 1
```

where 14 is for a 10+4 configuration with 10 data fragments and 4 parity fragments.

Consider the performance impacts when deciding which devices to use in the erasure coding policy's object ring. We recommend that you run some performance benchmarking in a test environment for the configuration before deployment. After you have configured your erasure coding policy in the **swift.conf** and created your object ring, your application is ready to start using erasure coding by creating a container with the specified policy name and interacting as usual.

3.1.2. Set Object Storage as a Back End for the Image Service

The OpenStack Image service, by default, saves images and instance snapshots to the local filesystem in **`/var/lib/glance/images/`**. Alternatively, you can configure the Image service to save images and snapshots to the Object Storage service (when available).

To do so, perform the following procedure:

1. Log into the node running the Image service (the controller node also running Identity) as root and source your OpenStack credentials (this is typically a file named **`openrc`**).

   ```
   # source ~/openrc
   ```

2. Verify that the Image service is part of the tenant **`service`** with role **`admin`**.

   ```
   # keystone user-role-list --user glance --tenant service
   ```

   One of the roles returned should be **`admin`**.

3. Open the **`/etc/glance/glance.conf`** file and comment out the following lines:

   ```
   ##### DEFAULT OPTIONS #####
   #default_store = file
   #filesystem_store_datadir = /var/lib/glance/images/
   ```

4. In the same file, add the following lines to the **`DEFAULT OPTIONS`** section.

```
default_store = swift
swift_store_auth_address = http://KEYSTONEIP:35357/v2.0/
swift_store_user = service:glance
swift_store_key = ADMINPW
swift_store_create_container_on_put = True
```

Where:

- **KEYSTONEIP** is the IP address of the Identity service, and
- **ADMINPW** is the value of admin password attribute in the **/etc/glance/glance-api.conf** file.

5. Apply the changes by restarting the Image service:

```
# systemctl restart openstack-glance-api
# systemctl restart openstack-glance-registry
```

From this point onwards, images uploaded to the Image service (whether through the Dashboard or **glance**) should now be saved to an Object Storage container named **glance**. This container exists in the service account.

To verify whether newly-created images are saved to the Image service, run:

```
# ls /var/lib/glance/images
```

Once the Dashboard or the **glance image-list** reports the image is active, you can verify whether it is in Object Storage by running the following command:

```
# swift --os-auth-url http://KEYSTONEIP:5000/v2.0 --os-tenant-name
service --os-username glance --os-password ADMINPW list glance
```

3.2. BASIC CONTAINER MANAGEMENT

To help with organization, pseudo-folders are logical devices that can contain objects (and can be nested). For example, you might create an *Images* folder in which to store pictures and a *Media* folder in which to store videos.

You can create one or more containers in each project, and one or more objects or pseudo-folders in each container.

3.2.1. Create a Container

1. In the dashboard, select **Project > Object Store > Containers**.
2. Click **Create Container**.
3. Specify the **Container Name**, and select one of the following in the **Container Access** field.

Type	Description
`Private`	Limits access to a user in the current project.
`Public`	Permits API access to anyone with the public URL. However, in the dashboard, project users cannot see public containers and data from other projects.

4. Click **Create Container**.

3.2.2. Create Pseudo Folder for Container

1. In the dashboard, select **Project > Object Store > Containers**.
2. Click the name of the container to which you want to add the pseudo-folder.
3. Click **Create Pseudo-folder**.
4. Specify the name in the **Pseudo-folder Name** field, and click **Create**.

3.2.3. Delete a Container

1. In the dashboard, select **Project > Object Store > Containers**.
2. Browse for the container in the **Containers** section, and ensure all objects have been deleted (see Section 3.2.6, “Delete an Object”).
3. Select **Delete Container** in the container’s arrow menu.
4. Click **Delete Container** to confirm the container’s removal.

3.2.4. Upload an Object

If you do not upload an actual file, the object is still created (as placeholder) and can later be used to upload the file.

1. In the dashboard, select **Project > Object Store > Containers**.
2. Click the name of the container in which the uploaded object will be placed; if a pseudo-folder already exists in the container, you can click its name.
3. Browse for your file, and click **Upload Object**.
4. Specify a name in the **Object Name** field:
   - Pseudo-folders can be specified in the name using a **/** character (for example, *Images/myImage.jpg*). If the specified folder does not already exist, it is created when the object is uploaded.

- A name that is not unique to the location (that is, the object already exists) overwrites the object's contents.

5. Click **Upload Object**.

3.2.5. Copy an Object

1. In the dashboard, select **Project > Object Store > Containers**.
2. Click the name of the object's container or folder (to display the object).
3. Click **Upload Object**.
4. Browse for the file to be copied, and select **Copy** in its arrow menu.
5. Specify the following:

Field	Description
Destination container	Target container for the new object.
Path	Pseudo-folder in the destination container; if the folder does not already exist, it is created.
Destination object name	New object's name. If you use a name that is not unique to the location (that is, the object already exists), it overwrites the object's previous contents.

6. Click **Copy Object**.

3.2.6. Delete an Object

1. In the dashboard, select **Project > Object Store > Containers**.
2. Browse for the object, and select **Delete Object** in its arrow menu.
3. Click **Delete Object** to confirm the object's removal.

CHAPTER 4. FILE SHARES

Important

The OpenStack Shared File System service is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

OpenStack's Shared File System service (**openstack-manila**) provides the means to easily provision shared file systems that can be consumed by multiple instances. In the past, OpenStack users needed to manually deploy shared file systems before mounting them on instances. The Shared File System service, on the other hand, allows users to easily provision shares from a pre-configured storage pool, ready to be mounted securely. This pool, in turn, can be independently managed and scaled to meet demand.

The OpenStack Shared File System service also allows administrators to define settings for different types of shares (namely, share type), in the same way that the OpenStack Block Storage service uses *volume types*. In addition, the Shared File System service also provides the means to manage access, security, and snapshots for provisioned shares.

At present, the Shared File System service can only be deployed manually. For instructions on how to do so, see Install the Shared File System Service (Technology Preview).

4.1. CREATE AND MANAGE SHARES

This section assumes that you manually deployed the Shared File System service as described in Install the Shared File System Service (Technology Preview) and OpenStack Shared File System Service (Manila). As such, at this point you should be using the NetApp driver (**manila.share.drivers.netapp.common.NetAppDriver**) for shares.

With this driver, you should be able to perform the following operations:

- Create and delete a share.
- Allow (read/write) or deny access to a share.

Before creating a share, you must first create a *share type*. Typically, this step is part of the Shared File System service deployment, as described in Create a Share Type for the Defined Back End.

The following procedures assume that your NetApp back end:

- Can be invoked through a share type named netapp, and
- Supports the NFS share protocol.

4.2. CREATE A SHARE

To create a share, log in to the Shared File System service host and run:

```
# manila create --share-type SHARETYPE --name SHARENAME PROTO GB
```

Where:

- *SHARETYPE* applies settings associated with the specified share type.
- *SHARENAME* is the name of the share.
- *PROTO* is the share protocol you want to use.
- *GB* is the size of the share, in GB.

For example, to create a 1 GB NFS share named share-00 using the netapp back end, run:

```
# manila create --share-type netapp --name share-00 nfs 10
 +-------------------+--------------------------------------+
 | Property          | Value                                |
 +-------------------+--------------------------------------+
 | status            | creating                             |
 | description       | None                                 |
 | availability_zone | nova                                 |
 | share_network_id  | None                                 |
 | export_locations  | []                                   |
 | share_server_id   | None                                 |
 | host              | None                                 |
 | snapshot_id       | None                                 |
 | is_public         | False                                |
 | id                | d760eee8-1d91-48c4-8f9a-ad07072e17a2 |
 | size              | 10                                   |
 | name              | share-01                             |
 | share_type        | 8245657b-ab9e-4db1-8224-451c32d6b5ea |
 | created_at        | 2015-09-29T16:27:54.092272           |
 | export_location   | None                                 |
 | share_proto       | NFS                                  |
 | project_id        | a19dc7ec562c4ed48cea58d22eb0d3c7     |
 | metadata          | {}                                   |
 +-------------------+--------------------------------------+
```

4.3. LIST SHARES AND EXPORT INFORMATION

To verify that the shares were created successfully:

```
# manila list
 +--------------------------------------+----------+-----+---------
--+
 | ID                                   | Name     | ... | Status
...
 +--------------------------------------+----------+-----+---------
--+
 | d760eee8-1d91-48c4-8f9a-ad07072e17a2 | share-01 | ... | available
...
 +--------------------------------------+----------+-----+---------
--+
```

The **manila list** command will also display the **export location** of the share:

```
 +-------------------------------------------------------------+
 | Export location                                            ...
 +-------------------------------------------------------------+
```

```
 | 10.70.37.46:/manila-nfs-volume-01/share-d760eee8-1d91-...
 +-----------------------------------------------------------+
```

This information will be used later when mounting the share (Section 4.5, “Mount a Share on an Instance”).

4.4. GRANT SHARE ACCESS

Before you can mount a share on an instance, grant the instance access to the share first:

```
# manila access-allow SHAREID IDENT IDENTKEY
```

Where:

- *SHAREID* is the ID of the share created in Section 4.2, “Create a Share”.
- *IDENT* is the method that the File Share Service should use to authenticate a share user or instance.
- The *IDENTKEY* varies depending on what identifying method you choose as *IDENT*:
  - cert: this method is used for authenticating an instance through TLS certificates.
  - user: use this to authenticate by user or group name.
  - ip: use this to authenticate an instance through its IP address.

For example, to grant read-write access to an instance (identified by the IP 10.70.36.85), run:

```
# manila access-allow d760eee8-1d91-48c4-8f9a-ad07072e17a2 ip
10.70.36.85
 +--------------+--------------------------------------+
 | Property     | Value                                |
 +--------------+--------------------------------------+
 | share_id     | d760eee8-1d91-48c4-8f9a-ad07072e17a2 |
 | deleted      | False                                |
 | created_at   | 2015-09-29T16:35:33.862114           |
 | updated_at   | None                                 |
 | access_type  | ip                                   |
 | access_to    | 10.70.36.85                          |
 | access_level | rw                                   |
 | state        | new                                  |
 | deleted_at   | None                                 |
 | id           | b4e990d7-e9d1-4801-bcbe-a860fc1401d1 |
 +--------------+--------------------------------------+
```

Note that access to the share has its own ID (*ACCESSID*), **b4e990d7-e9d1-4801-bcbe-a860fc1401d1**.

To verify that the access configuration was successful:

```
# manila access-list d760eee8-1d91-48c4-8f9a-ad07072e17a2
 +--------------------------+-----------+-----------+------------
-+
 | id                       |access type|access to  | access level
...
```

```
 +--------------------------+-----------+-----------+-------------
-+
 |b4e990d7-e9d1-4801-bcbe-...|ip         |10.70.36.85| rw
...
 +--------------------------+-----------+-----------+-------------
-+
```

4.5. MOUNT A SHARE ON AN INSTANCE

After configuring the share to authenticate an instance, you can then mount the share. For example, to mount the share from Section 4.2, “Create a Share” to /mnt on the instance from Section 4.4, “Grant Share Access”, log in to the instance and mount as normal:

```
# ssh root@10.70.36.85
# mount -t nfs -o vers=3 10.70.37.46:/manila-nfs-volume-01/share-
d760eee8-1d91-48c4-8f9a-ad07072e17a2 /mnt
```

See Section 4.3, “List Shares and Export Information” to learn how to view a share’s export information.

Upon mounting the volume from inside the instance, check if you can write to the share at its mount point.

4.6. REVOKE ACCESS TO A SHARE

To revoke previously-granted access to a share, you need to delete the access to the share:

```
# manila access-deny SHAREID ACCESSID
```

For example, to revoke the access granted earlier in Section 4.4, “Grant Share Access”:

```
# manila access-list d760eee8-1d91-48c4-8f9a-ad07072e17a2
 +--------------------------+-----------+-----------+-------------
-+
 | id                       |access type|access to  | access level
...
 +--------------------------+-----------+-----------+-------------
-+
 |b4e990d7-e9d1-4801-bcbe-...|ip         |10.70.36.85| rw
...
 +--------------------------+-----------+-----------+-------------
-+
# manila access-deny d760eee8-1d91-48c4-8f9a-ad07072e17a2 b4e990d7-
e9d1-4801-bcbe-a860fc1401d1
```

At this point, the instance will no longer be able to use the mounted share.

4.7. DELETE A SHARE

To delete a share:

```
# manila delete SHAREID
```

For example:

```
# manila delete d760eee8-1d91-48c4-8f9a-ad07072e17a2
```