Red Hat OpenStack Platform 14

Block Storage Backup Guide

Understanding, using, and managing the Block Storage backup service in OpenStack

Last Updated: 2019-08-14

Red Hat OpenStack Platform 14 Block Storage Backup Guide

Understanding, using, and managing the Block Storage backup service in OpenStack

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This document describes how to deploy the OpenStack Block Storage Backup Service. The instructions herein are specific to an overcloud deployment. The OpenStack director can configure Red Hat Ceph Storage, NFS, and Object Storage (swift) as back ends. Google Cloud Storage can also be configured as a backup back end.

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 the Block Storage backup service.

You can manage some features of the backup service 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. OVERVIEW

The Block Storage service (cinder) provides a horizontally scalable backup service that can be used to back up cinder volumes using diverse storage back ends. This service can be used to create full or incremental backups and to restore these backups. The service is volume-array independent.

The Red Hat OpenStack Platform director is a toolset for installing and managing a complete OpenStack environment. The Red Hat OpenStack director orchestrates a functional, Enterprise-grade OpenStack deployment with minimal manual configuration. It helps address many of the issues inherent in manually configuring individual OpenStack components.

The end-result OpenStack deployment provided by the director is called the overcloud. The overcloud houses all the components that provide services to end users, including Block Storage. The Block Storage backup service is an optional service deployed on controller nodes.

This document provides guidance on how to deploy the overcloud's Block Storage backup service to use a specific back end. This guide walks you through planning, installing, configuring, and using the Block Storage backup service.

1.1. WHAT IS A BACKUP?

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 OpenStack Object Storage service (swift) 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-backup service 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 want to configure only a subset of volume backups to survive a catastrophic database loss, you can also export the backup's metadata. You can then re-import the metadata to the Block Storage database later, using the REST API or the cinder client, and restore the volume backup as normal.

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 the "Create, Use, or Delete Volume Snapshots " section of the *Storage Guide*.

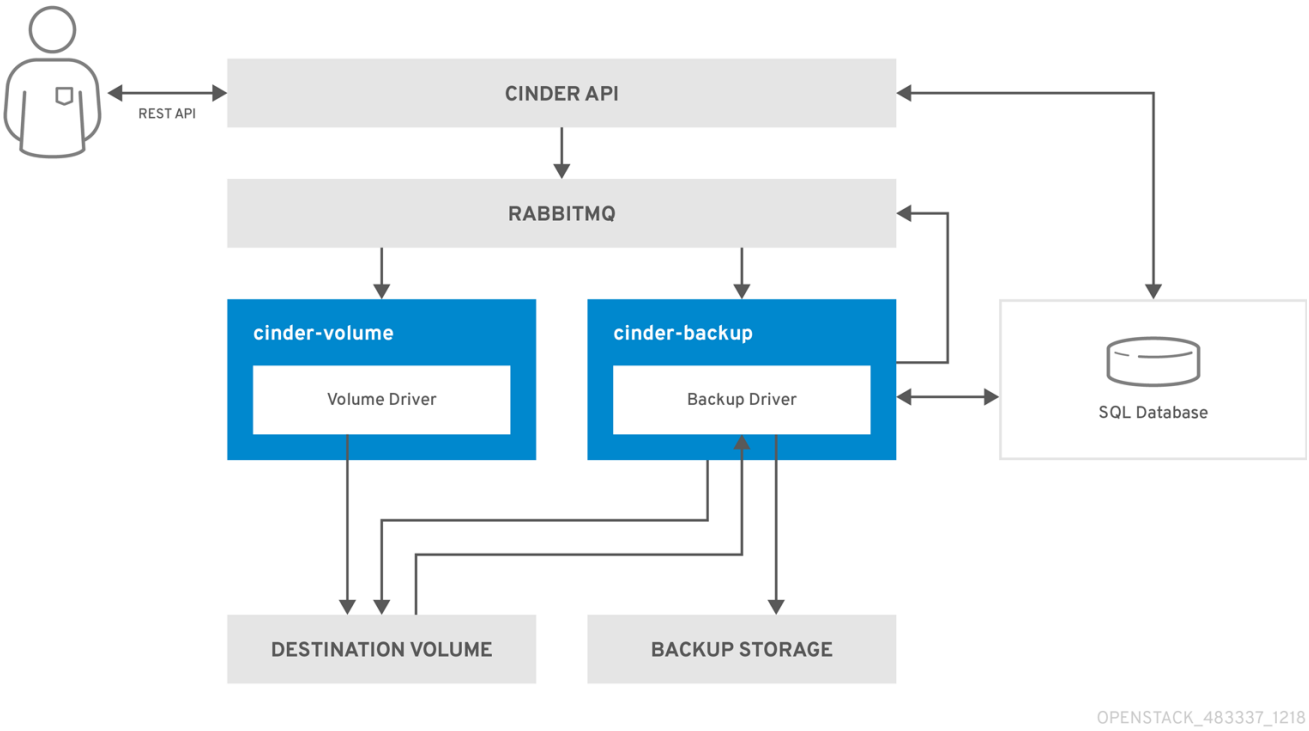
1.2. HOW DO BACKUPS AND RESTORES WORK?

Volume backups and restores have similar workflows, illustrated below.

1.2.1. Volume backup workflow

When the Block Storage backup service performs a back up, it receives a request from the cinder API to backup a targeted volume. The request is completed and the backup content is stored on the back end.

The diagram below illustrates how the request interacts with the cinder services to perform the backup.

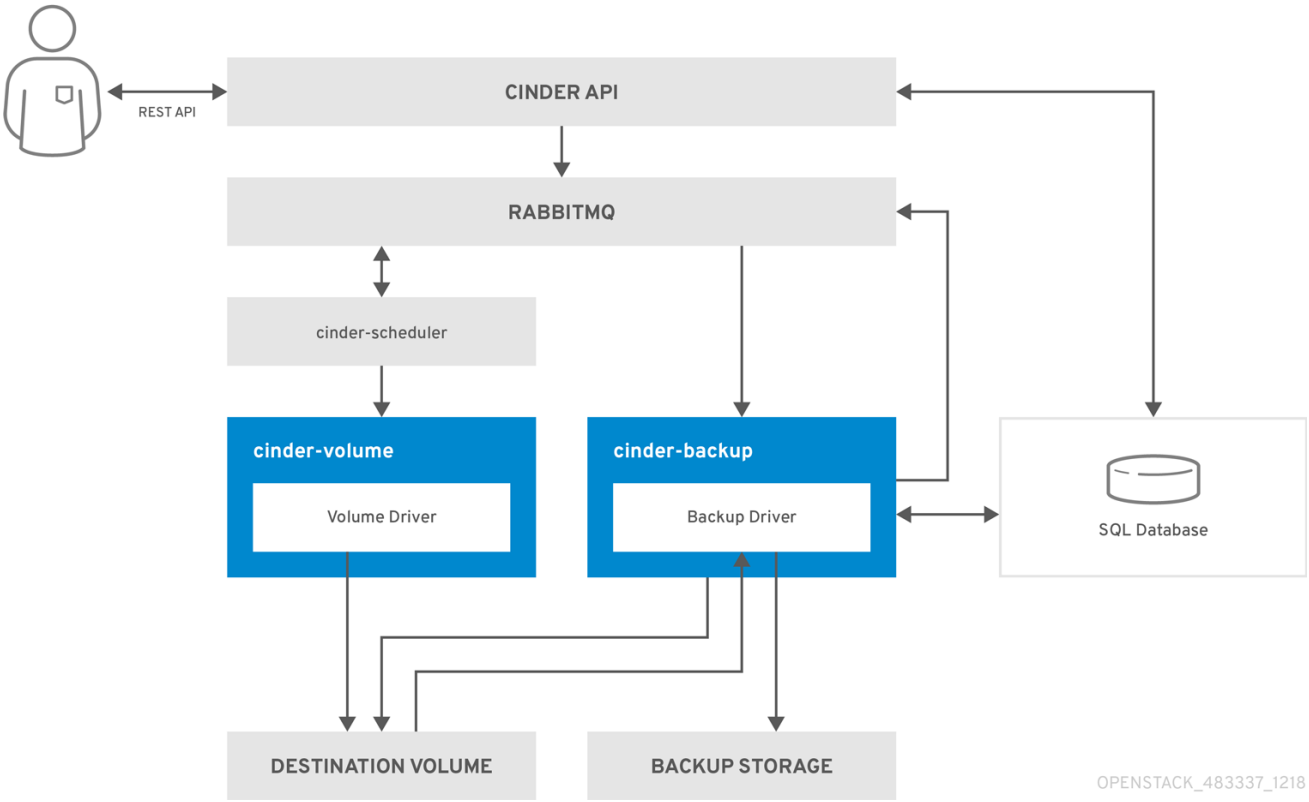


1. The client issues request to backup a Cinder volume by invoking the Cinder REST API (via the dashboard, client, etc.).
2. The **Cinder API** service receives the request (from **HAProxy**) and validates the request, user credentials, etc.
3. Creates the backup record in the SQL database.
4. Makes an asynchronous RPC call to the **cinder-backup** service via AMQP to backup the volume.
5. Returns current Backup record, with an ID, to the API caller.
6. RPC create message arrives on one of the backup services.
7. The **cinder-backup** service does a synchronous RPC call to get_backup_device.
8. The **cinder-volume** service ensures the right device is returned to the caller. Normally, it is the same volume, but if the volume is in use, the service will return instead a temporary cloned volume or a temporary snapshot, depending on the configuration.
9. The **cinder-backup** service issues another synchronous RPC to **cinder-volume** to expose the source device.
10. The **cinder-volume** service exports and maps the source device (volume or snapshot) returning the appropriate connection information.
11. The **cinder-backup** service attaches source volume using connection information.
12. The **cinder-backup** service calls the Backup Driver, with the device already attached, which begins the data transfer to the backup destination.

13. The volume is detached from the Backup host.
14. The **cinder-backup** service issues a synchronous RPC to **cinder-volume** to disconnect the source device.
15. The **cinder-volume** service unmaps and removes the export for the device.
16. If a temporary volume or temporary snapshot was created, **cinder-backup** calls **cinder-volume** to remove it.
17. The **cinder-volume** service removes the temporary volume.
18. Once the backup is completed, the Backup record is updated in the database.

1.2.2. Volume restore workflow

The following diagram illustrates the steps that occur when a user requests that a Block Storage backup be restored.



1. The client issues request to restore a Cinder backup by invoking the CinderREST API (via the dashboard, client, etc.).
2. The **Cinder API** receives the request (from **HAProxy**) and validates the request, user credentials, etc.
3. If the request didn't contain an existing volume as the destination, the API will make an asynchronous RPC call to create a new volume and polls the status of the volume until it becomes available.
4. The **cinder-scheduler** selects a volume service and makes the RPC call to create the volume.
5. Selected **cinder-volume** service creates the volume.

6. Once **cinder-api** detects that the volume is available, the backup record is created in the database.
7. Makes an asynchronous RPC call to the backup service via AMQP to restore the backup.
8. Returns the current volume ID, backup ID, and volume name to the API caller.
9. RPC create message arrives on one of the backup services.
10. The **cinder-backup** service makes a synchronous RPC call to **cinder-volume** to expose the destination volume.
11. The **cinder-volume** service exports and maps the Destination Volume returning the appropriate connection information.
12. The **cinder-backup** service attaches source volume using connection information.
13. The **cinder-backup** service calls the driver with the device already attached which begins the data restoration to the volume destination.
14. The volume is detached from the Backup host.
15. The **cinder-backup** service issues a synchronous RPC to **cinder-volume** to disconnect the source device.
16. The **cinder-volume** service unmaps and removes the export for the device.
17. Once the backup is completed, the Backup record is updated in the database.

1.3. CLOUD STORAGE VERSUS LOCAL STORAGE

The Google Cloud Storage driver is the only cloud driver supported by the Block Storage backup service. By default, the Google Cloud Storage driver uses the least expensive storage solution, nearline, which is meant for this type of backup.

Updating configuration settings may improve performance. For example, if you are creating backups from Europe and you leave the backup default region (US), performance may be slower because you are backing up to a region that is a farther away.

NOTE

Google Cloud Storage requires special configuration that is explained in section Appendix A, *Google Cloud Storage configuration*.

The table below compares pros and cons for cloud storage and local storage based upon the situation.

Situation	Cloud	Local
Offsite backup	Cloud storage is in another company's data center and therefore automatically offsite. Access to data from many locations. Remote copy for disaster recovery.	Requires additional planning and expense.

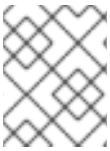
Situation	Cloud	Local
Hardware control	Relies on the availability of another service and their expertise.	Complete control over storage hardware. Requires management and expertise.
Cost considerations	Different pricing policies or tiers depending upon the services you use from the vendor.	Known cost to add additional hardware as needed.
Network speed and data access	Overall data access is slower and requires Internet access. Speed and latency depend upon multiple factors.	Immediate and fast access to data. No Internet access required.

CHAPTER 2. BLOCK STORAGE BACKUP SERVICE DEPLOYMENT

The Block Storage backup service is optional. It is not installed by default and must be added to the overcloud deployment.

To deploy the backup service, you need:

- An existing OpenStack installation or be setting up a new one
- An available storage source with a compatible backup driver: Object Storage (swift; default), Ceph, NFS, or Google Cloud storage.



NOTE

Google Cloud Storage requires additional configuration explained in Appendix A, *Google Cloud Storage configuration*.

The examples in this section assume that you are deploying the backend service in a standard OpenStack environment that uses Pacemaker (default installation).

2.1. CONFIGURING BACKEND OPTIONS

The backup service is enabled by including the **cinder-backup.yaml** environment file, which resides in the **/usr/share/openstack-tripleo-heat-templates/environments/** directory.

The default settings in this file set up a swift back end for the Block Storage backup service with Pacemaker.

Procedure

The next step is to create a custom environment file, for example **cinder-backup-settings.yaml**, that contains the parameter settings for the backup service and configuration options for the driver.

1. Create a copy of the **cinder-backup.yaml** file and store it in the same location as other custom templates.

```
cp /usr/share/openstack-tripleo-heat-templates/environments/cinder-backup.yaml
/home/stack/templates/cinder-backup-settings.yaml
```

2. Modify the appropriate options for the backup back end that you are using (see instructions in the sections below).

3. Save the changes to the file.

2.1.1. Object storage (swift)

Swift is the default value for the **CinderBackupBackend** option. If you are using swift, no additional changes are needed.

Example

```
resource_registry:
  OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
```

```
templates/docker/services/pacemaker/cinder-backup.yaml
  # For non-pcmk managed implementation
  # OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/cinder-backup.yaml

parameter_defaults:
  CinderBackupBackend: swift
```

Setting	Options	Values
CinderBackupBackend	**swift** (default)	Swift is the default selection in the **cinder-backup.yaml** template.

2.1.2. Red Hat Ceph Storage

If you are using Red Hat Ceph Storage as a backup back end, then you have the option of changing the RBD pool name used for the backup. The default value is **backups**.

Example

```
resource_registry:
  OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/pacemaker/cinder-backup.yaml
  # For non-pcmk managed implementation
  # OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/cinder-backup.yaml

parameter_defaults:
  CinderBackupBackend: ceph
  CinderBackupRbdPoolName: backups
```

Setting	Options	Values
CinderBackupBackend	**ceph**	Required. Change the value to **ceph.**
CinderBackupRbdPoolName	**backups** (default name)	Optional. No other settings need to be change unless you are using a custom RBD pool name.

2.1.3. NFS

To use NFS as a back end for the backup service, you need to provide the NFS share to be mounted.

Example

```
resource_registry:
  OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/pacemaker/cinder-backup.yaml
```

```
  # For non-pcmk managed implementation
  # OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/cinder-backup.yaml

parameter_defaults:
  CinderBackupBackend: nfs
  CinderBackupNfsShare: '192.168.122.1:/export/cinder/backups'
  CinderBackupNfsMountOptions: ''
```

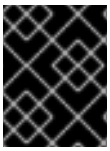
Setting	Options	Values
CinderBackupBackend	nfs	Required. Set **nfs** as the value.
CinderBackupNfsShare		Required. Enter the NFS share to be mounted. Default value is empty.
CinderNfsMountOptions		Optional. Backup NFS Mount options can be left blank. If you need to specify mount options, include them here.

Additional configuration options are described in Appendix A, *Google Cloud Storage configuration* .

2.2. DEPLOYING THE BLOCK STORAGE BACKUP SERVICE

After you create the environment file in **/home/stack/templates/**, log in as the stack user and deploy the configuration by running:

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/cinder-backup-settings.yaml
```



IMPORTANT

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

For more information, see the Including Environment Files in Overcloud Creation in the *Director Installation and Usage Guide* and the Environment Files section of the *Advanced Overcloud Customization Guide*.

CHAPTER 3. USING THE BLOCK STORAGE BACKUP SERVICE

This chapter explains how to use the Block Storage backup service to perform full or incremental backups, and how to restore a backup to a volume. Basic troubleshooting tips are also provided.

3.1. FULL BACKUPS

3.1.1. Creating a full volume backup

To back up a volume, use the **cinder backup-create** command. By default, this command creates 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.)

NOTE

If you are using a Ceph volume backed up to a Ceph cluster, then the second time you do a backup, the backup service automatically performs incremental backups.

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 3.1.2, "Creating a volume backup as an admin".

Procedure

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_
```

3. 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 |
+-----------+--------------------------------------+
```

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

4. Verify that the volume backup creation is complete:

```
# cinder backup-list
```

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

3.1.2. Creating 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 counts against the quota of *TENANTNAME* rather than the admin's tenant.

3.1.3. Exporting a volume backup's metadata

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 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.

3.1.4. Backing up an in-use volume

You can also use the **cinder backup-create** command to create a backup of an in-use volume by adding the **--force** option.

NOTE

The **--force** option relies on Block Storage back end snapshot support and should be supported by most drivers. You can verify snapshot support by checking the documentation for the back end you are using.

By using the **--force** option, you acknowledge that you are not quiescing the drive before performing the backup. Using this method creates a crash-consistent (but not application-consistent) backup. In other words, the backup will not have an awareness of which applications were running when the backup was performed. However, the data will be intact.

Procedure

To create a backup of an in-use volume, run:

```
# cinder backup-create _VOLUME_ --incremental --force
```

3.1.5. Backing up a snapshot

When creating a backup from a snapshot, you can do a full backup or an incremental backup (by including the **--incremental** option). You must identify the volume ID associated with the snapshot.

Procedure

1. Locate the snapshot ID of the snapshot to backup using **cinder snapshot list**.

   ```
   # cinder snapshot-list --volume-id _VOLUME_ID_
   ```

2. If the snapshot is named, then you can use this example to locate the **ID**:

   ```
   # cinder snapshot-show _SNAPSHOT_NAME_
   ```

3. Create the backup of a snapshot:

   ```
   # cinder backup-create _VOLUME_ --snapshot-id=_SNAPSHOT_ID_
   ```

3.2. INCREMENTAL BACKUPS

The Block Storage backup service provides the option of performing incremental backups.

3.2.1. Performance considerations

Some backup features like incremental and data compression may impact performance. Incremental backups have a performance impact because all of the data in a volume must be read and checksummed for both the full and each incremental backup.

Data compression can be used with non-Ceph backends. Enabling data compression requires additional CPU power but uses less network bandwidth and storage space overall.

Multipathing configuration also impacts performance. If multiple volumes are attached without multipathing enabled, you might not be able to connect and might not be using the full network capabilities, which impacts performance.

You can use the advanced configuration options (see Appendix B, *Advanced Block Storage backup configuration options*) to enable or disable compression, define the number of processes, and add additional CPU resources.

3.2.1.1. Backup from a snapshot impact

Some back ends support creating a backup from a snapshot. A driver that supports this feature can directly attach a snapshot, which is faster than cloning the snapshot into a volume to be able to attach to it. In general, this feature can affect performance because of the extra step of creating the volume from a snapshot.

3.2.2. Performing incremental backups

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

Incremental backups are fully supported on NFS, Object Storage (swift), and Red Hat Ceph Storage backup repositories. Ceph backups currently ignore the **--incremental** option: Ceph backups always try to perform incremental backups when the source is a Ceph volume.

NOTE

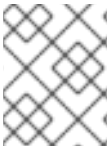
Incremental Ceph backups can not be performed for non-Ceph volumes.

An incremental backup captures any changes to the volume since the last backup (full or incremental). Performing numerous, regular, full backups 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.



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.

3.3. CANCELING A BACKUP

To issue a backup cancellation on a backup, an admin must request a force delete on the backup.

```
# cinder backup-delete --force BACKUP ID
```

Even if the backup is immediately deleted, and therefore no longer appears in the listings, the cancellation may take a little bit longer. Check the status of the source resource to verify when the status stops being "backing-up".

NOTE

Before OpenStack version 12, the "backing-up" status would always be stored in the volume, even when backing up a snapshot, so when backing up a snapshot any delete operation on the snapshot that followed a cancellation could result in an error if the snapshot was still mapped. Since version 13, ongoing restoring operation can be canceled on any of the supported backup drivers.

3.4. VIEWING AND MODIFYING A TENANT'S BACKUP QUOTA

Normally, you can use the dashboard to modify tenant storage quotas, for example, the number of volumes, volume storage, snapshots, or other operational limits that a tenant can have. However, the functionality to modify backup quotas with the dashboard is not yet available.

You must use the command-line interface to modify backup quotas with the **cinder quota-update** command.

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

```
# cinder quota-show TENANT_ID
```

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

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

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

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

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

```
# cinder quota-usage TENANT_ID
```

3.5. RESTORING FROM BACKUPS

3.5.1. Restoring a volume from a backup

These steps create a new volume from a backup.

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

   ```
   # cinder backup-list
   ```

2. The Volume ID should match the ID of the volume you want to restore.

3. Restore the volume backup:

   ```
   # cinder backup-restore _BACKUP_ID_
   ```

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

If you no longer need the backup, delete it:

```
# cinder backup-delete _BACKUP_ID_
```

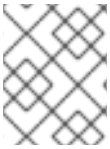
If you need to restore a backed up volume to a volume of a particular type, you can use **--volume** to restore a backup to a specific volume:

```
# cinder backup-restore _BACKUP_ID --volume VOLUME_ID_
```

3.5.2. Restoring 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 3.1.1, “Creating 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).



NOTE

For incremental backups, you must import all exported data before you can restore one of the incremental backups

Procedure

1. As a user with administrative privileges, run:

```
# cinder 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 3.1.1, “Creating a full volume backup” :

```
# cinder 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 3.5.1, “Restoring a volume from a backup”).

3.5.3. Canceling a restoration

To issue a backup restoration cancellation, alter its status to anything other than restoring. You can use the “error” state to avoid any confusion on whether the restore was successful or not or the value can be changed to “available.”

```
$ openstack volume backup set --state error BACKUP_ID
```

The backup cancelation is an asynchronous action, as the backup driver needs to detect the status change before cancelling the backup. A change to “available” in the destination volume status means that the cancellation has been completed.

NOTE

This feature is not currently available on RBD backups.

WARNING

After a restore operation has started, if it is then cancelled, the destination volume is useless, as there is no way of knowing how much data, or if any, was actually restored.

3.6. TROUBLESHOOTING

Many issues can be diagnosed by verifying services are available and looking in log files for error messages.

Two scenarios account for many issues with the backup service:

- When the **cinder-backup** service starts up, it connects to its configured backend and uses this as a target for backups. Problems with this connection can cause service to be down.
- When backups are requested, the backup service connects to the volume service and attaches the requested volume. Connection problems here won't be seen until backup time.

In either case, messages describing the error should be present in the logs.

For general information, log locations, and troubleshooting suggestions, refer to the Logging, Monitoring and Troubleshooting Guide. Log files and services are listed in the Log Files for OpenStack Services section.

3.6.1. Verifying services

One of the first steps in troubleshooting is to verify that the necessary services are available and to check the logs for additional clues. Section 1.2, "How do backups and restores work?" illustrates the key services and their interaction.

After you verify the status of the services, check the **cinder-backup.log** file. The Block Storage Backup service log is located in **/var/log/containers/cinder]/cinder-backup.log**.

Procedure

1. Use **cinder show** on the volume to see if it is stored by the host.

```
# cinder show
```

2. Use **cinder service-list** to view running services.

```
# cinder service-list
+------------------+-------------------+------+---------+-------+----------------------------+-------------
---+
| Binary           | Host              | Zone | Status  | State | Updated_at                 | Disabled
Reason |
+------------------+-------------------+------+---------+-------+----------------------------+-------------
---+
| cinder-backup    | hostgroup         | nova | enabled | up    | 2017-05-15T02:42:25.000000 | -
```

```
|
| cinder-scheduler | hostgroup          | nova | enabled | up    | 2017-05-15T02:42:25.000000 | -
|
| cinder-volume    | hostgroup@sas-pool | nova | enabled | down  | 2017-05-
14T03:04:01.000000 | -              |
| cinder-volume    | hostgroup@ssd-pool | nova | enabled | down  | 2017-05-
14T03:04:01.000000 | -              |
+------------------+--------------------+------+---------+-------+----------------------------+-------------
---+
```

3. Verify that the expected services are available.

3.6.2. Troubleshooting tips

Backups themselves are asynchronous. A relatively small number of static checks are performed upon receiving an API request. These checks include an invalid volume reference (missing) or a volume that is **in-use** or attached to an instance. The in-use case requires you to use the **--force** option.

NOTE

Using the **--force** option means that I/O will not be quiesced and the resulting volume image may be corrupt.

If the API accepts the request, the backup will take place in the background. Usually the CLI will return immediately even if the backup has failed or is about to fail. The status of a backup can be queried using cinder's backup API. If an error does occur, review the logs for the cause.

3.6.3. Pacemaker

The Block Storage backup service is deployed with Pacemaker by default. By configuring virtual IP addresses, containers, services, and other features as resources in a cluster, Pacemaker makes sure that the defined set of OpenStack cluster resources are running and available. When a service or an entire node in a cluster fails, Pacemaker can restart the resource, take the node out of the cluster, or reboot the node. Requests to most of those services are done through HAProxy.

For information on using Pacemaker for troubleshooting, refer to the Using Pacemaker chapter of the *Understanding Red Hat OpenStack Platform High Availability* guide.

APPENDIX A. GOOGLE CLOUD STORAGE CONFIGURATION

Configuring the Block Storage service to use Google Cloud Storage as a backup back end involves the following steps:

- Creating and downloading the service account credentials of your Google account.
- Creating an environment file to map out the Block Storage settings required. This environment file will also use the service account credentials created in the previous step.
- Re-deploying the overcloud using the environment file you created.

A.1. REQUIREMENTS

Deploying Google Cloud Storage for backups requires that:

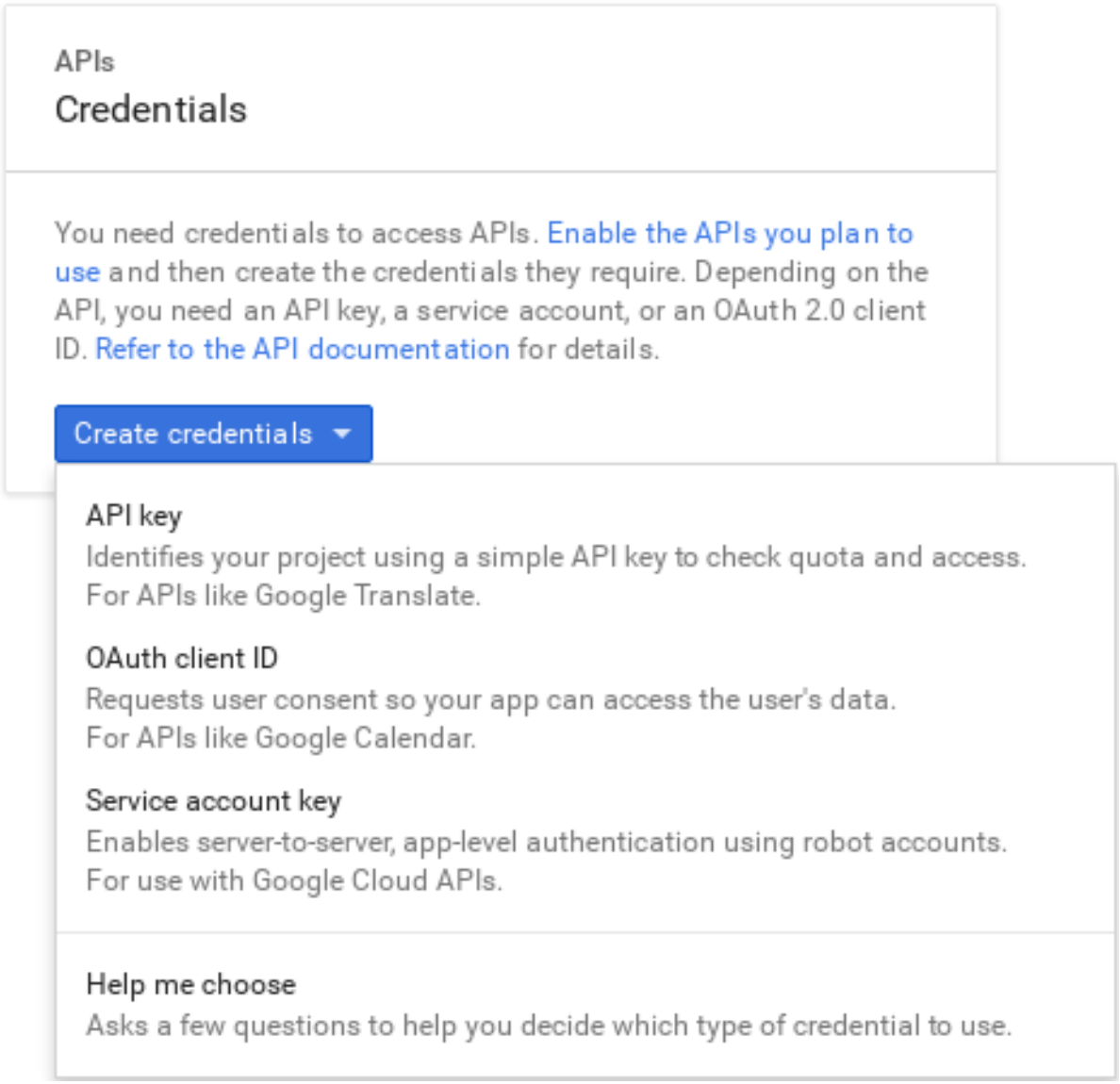
- You have the username and password of an account with elevated privileges. You can use the same account that was created to deploy the overcloud; in the *Director Installation and Usage Guide*, a user named **stack** is created for this purpose.
- You have a Google account with access to Google Cloud Platform. This account will be used by the Block Storage service to access and use Google Cloud for storing backups.

A.2. CREATING THE GCS CREDENTIALS FILE

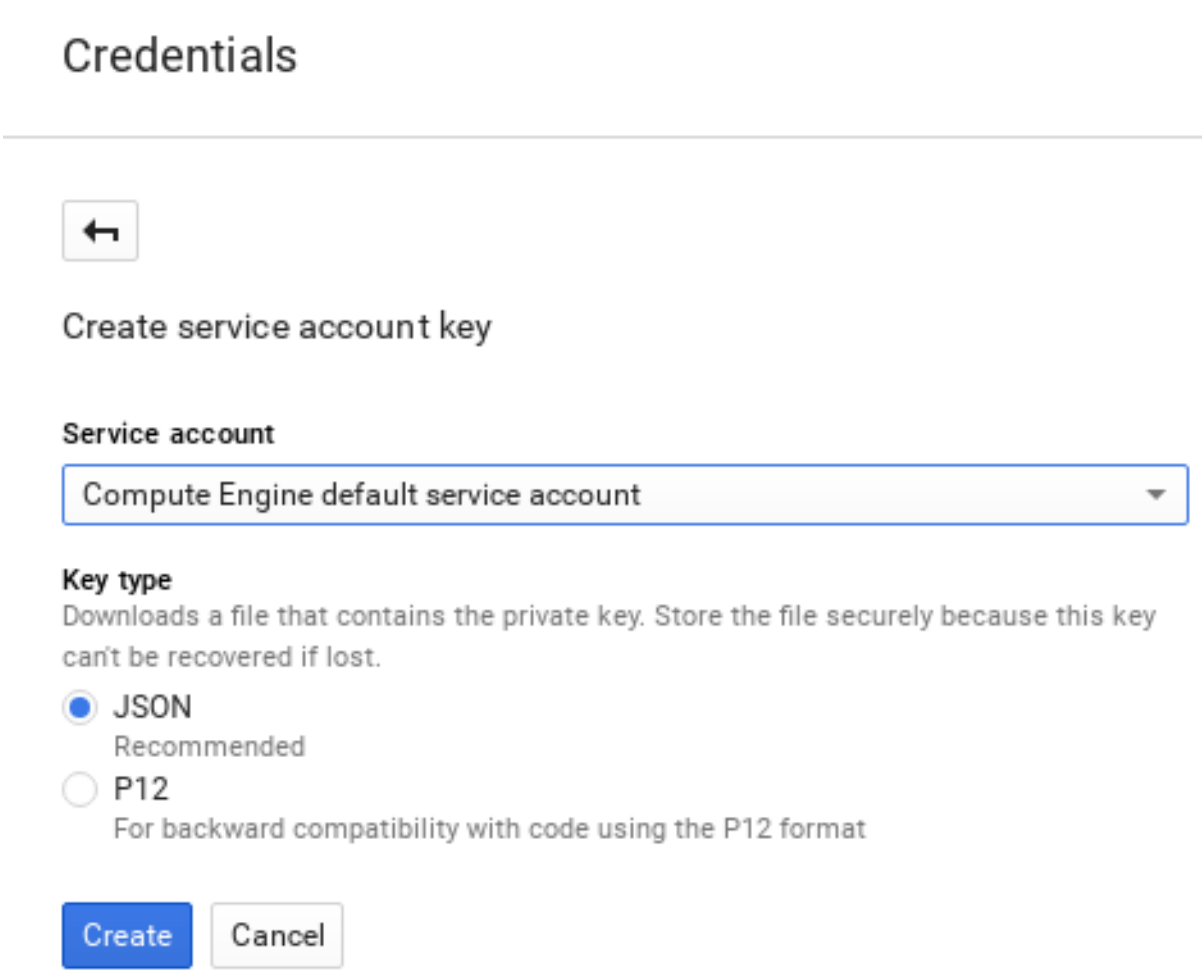
The Block Storage service needs your Google credentials to access and use Google Cloud for backups. You can provide these credentials to Block Storage by creating a service account key.

Procedure

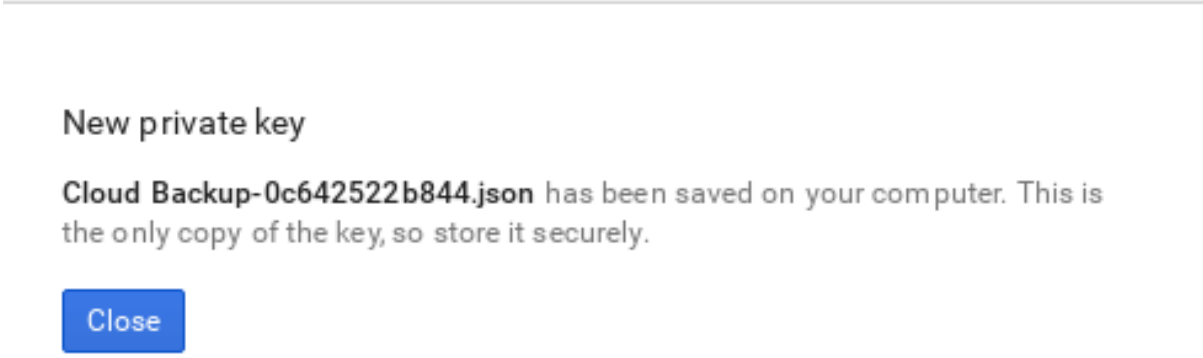
1. Log in to the Google developer console (http://console.developers.google.com) using your Google account.
2. Click the **Credentials** tab. From there, select **Service account key** from the **Create credentials** dropdown.



3. In the **Create service account key** screen, select the service account that the Block Storage service should use from the **Service account** dropdown:



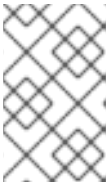
4. In the same screen, select **JSON** from the **Key** type section and click **Create**.
5. The browser will then download the key to its default download location:



6. Open the file and note the value of the **project_id** parameter:

```
{
  "type": "service_account",
  "project_id": "*cloud-backup-1370*",
...
```

7. Save a copy of the GCS JSON credentials to **/home/stack/templates/Cloud-Backup.json**

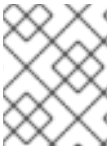


NOTE

Make sure to name the file **Cloud-Backup.json** and do not change the file name. This JSON file needs to be in the same directory location as the [filename}`cinder-backup-gcs.yaml` file created in the next section.

A.3. CREATING CINDER-BACKUP-GCS.YAML

Using the example file provided, create the **cinder-backup-gcs.yaml** file.



NOTE

The white space and format used in this the example (and in your file) are critical. If the white space is changed, then the file may not function as expected.

Procedure

1. Copy the text below, paste it into the new file. Do not make any modifications to the file contents.

```
heat_template_version: rocky

description: >
  Post-deployment for configuration cinder-backup to GCS

parameters:
  servers:
    type: json
  DeployIdentifier:
    type: string

resources:
  CinderBackupGcsExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/bash
            GCS_FILE=/var/lib/config-data/puppet-generated/cinder/etc/cinder/Cloud-
Backup.json
            HOSTNAME=$(hostname -s)
            for NODE in $(hiera -c /etc/puppet/hiera.yaml cinder_backup_short_node_names | tr
-d '[]",'); do
              if [ $NODE == $HOSTNAME ]; then
                cat <<EOF > $GCS_FILE
            GCS_JSON_DATA
            EOF
                chmod 0640 $GCS_FILE
                chown root:42407 $GCS_FILE
              fi
            done
          params:
            GCS_JSON_DATA: {get_file: Cloud-Backup.json}
```

```
CinderBackupGcsDeployment:
  type: OS::Heat::SoftwareDeploymentGroup
  properties:
    servers:  {get_param: servers}
    config: {get_resource: CinderBackupGcsExtraConfig}
    actions: ['CREATE','UPDATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}
```

2. Save the file as **/home/stack/templates/cinder-backup-gcs.yaml**.

A.4. CREATING THE ENVIRONMENT FILE

The environment file contains the settings you want to apply to the Block Storage service. In this case, the Block Storage service will be configured to store volume backups to Google Cloud. For more information about environment files, see the Director Installation and Usage guide.

Use the example environment file below and update the **backup_gcs_project_id** with the project ID listed in the **Cloud-Backup.json** file. You may also wish to change the **backup_gcs_bucket_location** location from US to a location closer to you. See the Google Cloud Backup Storage backup back end configuration options table for a list of options.

Procedure

1. Copy the environment file example below. Make sure to retain the white space usage.
2. Paste the content into a new file: **/home/stack/templates/cinder-backup-settings.yaml**
3. Change the value for **backup_gcs_project_id** from **cloud-backup-1370** to the project ID listed in the **Cloud-Backup.json** file.
4. Save the file.

Environment file example

Each setting is defined in the environment file. Available options are explained in the table below.

```
resource_registry:
  OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/pacemaker/cinder-backup.yaml
  # For non-pcmk managed implementation
  # OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/docker/services/cinder-backup.yaml
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/cinder-backup-gcs.yaml

parameter_defaults:
  CinderBackupBackend: swift
  ExtraConfig:
    cinder::backup::swift::backup_driver: cinder.backup.drivers.gcs.GoogleBackupDriver
    cinder::config::cinder_config:
      DEFAULT/backup_gcs_credential_file:
        value: /etc/cinder/Cloud-Backup.json
      DEFAULT/backup_gcs_project_id:
        value: cloud-backup-1370
      DEFAULT/backup_gcs_bucket:
```

```
    value: cinder-backup-gcs
DEFAULT/backup_gcs_bucket_location:
    value: us
```

Table A.1. Google Cloud Storage backup back end configuration options

PARAM	Default	*CONFIG* Description
backup_gcs_project_id		Required. The project ID of the service account you are using, as noted in the project_id of the service account key from Section A.2, "Creating the GCS credentials file".
backup_gcs_credential_file		The absolute path to the service account key file you created earlier in Section A.2, "Creating the GCS credentials file".
backup_gcs_bucket		The GCS bucket (or object storage repository) to use, which may or may not exist. If you specify a non-existent bucket, the Google Cloud Storage backup driver creates one using the name you specify here. See Buckets and Bucket name requirements for more information.
backup_gcs_bucket_location	**us**	The location of the GCS bucket. This value is only used if you specify a non-existent bucket in backup_gcs_bucket; in which case, the Google Cloud Storage backup driver will specify this as the GCS bucket location.
backup_gcs_object_size	**52428800**	The size (in bytes) of GCS backup objects.
backup_gcs_block_size	**32768**	The size (in bytes) that changes are tracked for incremental backups. This value must be a multiple of the backup_gcs_object_size value.
backup_gcs_user_agent	**gcscinder**	The HTTP user-agent string for the GCS API.
backup_gcs_reader_chunk_size	**2097152**	GCS objects will be downloaded in chunks of this size (in bytes).

PARAM	Default	*CONFIG* Description
backup_gcs_writer_chunk_size	**2097152**	GCS objects will be uploaded in chunks of this size (in bytes). To upload files as a single chunk instead, use the value -1.
backup_gcs_num_retries	**3**	Number of retries to attempt.
backup_gcs_storage_class	**NEARLINE**	Storage class of the GCS bucket. This value is only used if you specify a non-existent bucket in backup_gcs_bucket; in which case, the Google Cloud Storage backup driver will specify this as the GCS bucket storage class. See Storage Classes for more information.
backup_gcs_retry_error_codes	**429**	List of GCS error codes.
backup_gcs_enable_progress_timer	**True**	Boolean to enable or disable the timer for sending periodic progress notifications to the Telemetry service (ceilometer) during volume backups. This is enabled by default (True)

WARNING

When creating new buckets, Google Cloud Storage charges based on your chosen storage class (**backup_gcs_storage_class**). The default **NEARLINE** class is appropriate for backup services.

In addition, you cannot edit the location or class of a bucket once it is created. For more information, see Managing a bucket's storage class or location .

A.5. DEPLOYING THE OVERCLOUD

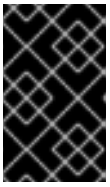
Once you have created the environment file file in **/home/stack/templates/**, deploy the overcloud then restart the cinder-backup service:

Procedure

1. Log in as the **stack** user.

2. Deploy the configuration by running:

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/cinder-backup-settings.yaml
```



IMPORTANT

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

3. Restart the **cinder-backup** service after the deployment finishes.

For more information, see the Including Environment Files in Overcloud Creation in the *Director Installation and Usage Guide* and the Environment Files section of the *Advanced Overcloud Customization Guide*.

APPENDIX B. ADVANCED BLOCK STORAGE BACKUP CONFIGURATION OPTIONS

Prior to director-deployed installations, the cinder.conf file was used to configured the Block Storage service and the backup service. When a value from cinder.conf does not have a Heat template equivalent, the values can still be passed from director using a custom environment. The values are added to an **ExtraConfig** section in the **parameter_defaults** section of a custom environment file (like **cinder-backup-settings.yaml**).

ExtraConfig provides a method for additional hiera configuration to inject into the cluster on all nodes. These settings are included on a dedicated backup node, for example, if you used **ExtraConfig**. If you used **ControllerExtraConfig** instead of **ExtraConfig**, then those settings would only be installed on controller nodes and not on a dedicated backup node.

You can substitute **DEFAULT/[cinder.conf setting]** for the setting that would be used in the **DEFAULT** section of the cinder.conf file. The example below shows how the **ExtraConfig** and entries appear in a YAML file.

```
parameter_defaults:
 ExtraConfig:
   cinder::config::cinder_config:
     DEFAULT/backup_compression_algorithm:
       value: None
```

The options below provide a sample of the backup-related options.

Table B.1. Block Storage backup service configuration options

Option	Type	Default Value	Description
backup_service_inithost_offload	Optional	True	Offload pending backup delete during backup service startup. If false, the backup service remains down until all pending backups are deleted.
use_multipath_for_image_xfer	Optional	False	Attach volumes using multipath, if available, during backup and restore procedures. This affects all cinder attach operations, such as create volume from image, generic cold migrations, and so forth.
num_volume_device_scan_tries	Optional	3	The maximum number of times to rescan targets to find volume during attach.

Option	Type	Default Value	Description
backup_workers	Optional	1	Number of backup processes to run. Performance gains will be significant when running multiple concurrent backups or restores with compression.
backup_native_threads_pool_size	Optional	60	Size of the native threads pool for the backups. Most backup drivers rely heavily on this. The value can be decreased for specific drivers that don't rely on this option.
backup_share	Required		Set to *HOST*:_EXPORT_PATH_.
backup_container	Optional	None	(String) Custom directory to use for backups.
backup_enable_progress_timer	Optional	True	Enable (true) or disable (false) the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the backend storage.
backup_mount_options	Optional		Comma-separated list of options to be specified when mounting the NFS export specified in backup_share.
backup_mount_point_base	Optional	$state_path/backup_mount	(String) Base directory containing mount point for NFS share.

Option	Type	Default Value	Description
backup_compression_algorithm	Optional	zlib	The compression algorithm to be used when sending backup data to the repository. Valid values are **zlib**, **bz2**, and **None**.
backup_file_size	Optional	1999994880	Data from cinder volumes larger than this will be stored as multiple files in the backup repository. This option must be a multiple of backup_sha_block_size_bytes.
backup_sha_block_size_bytes	Optional	32768	Size of cinder volume blocks for digital signature calculation