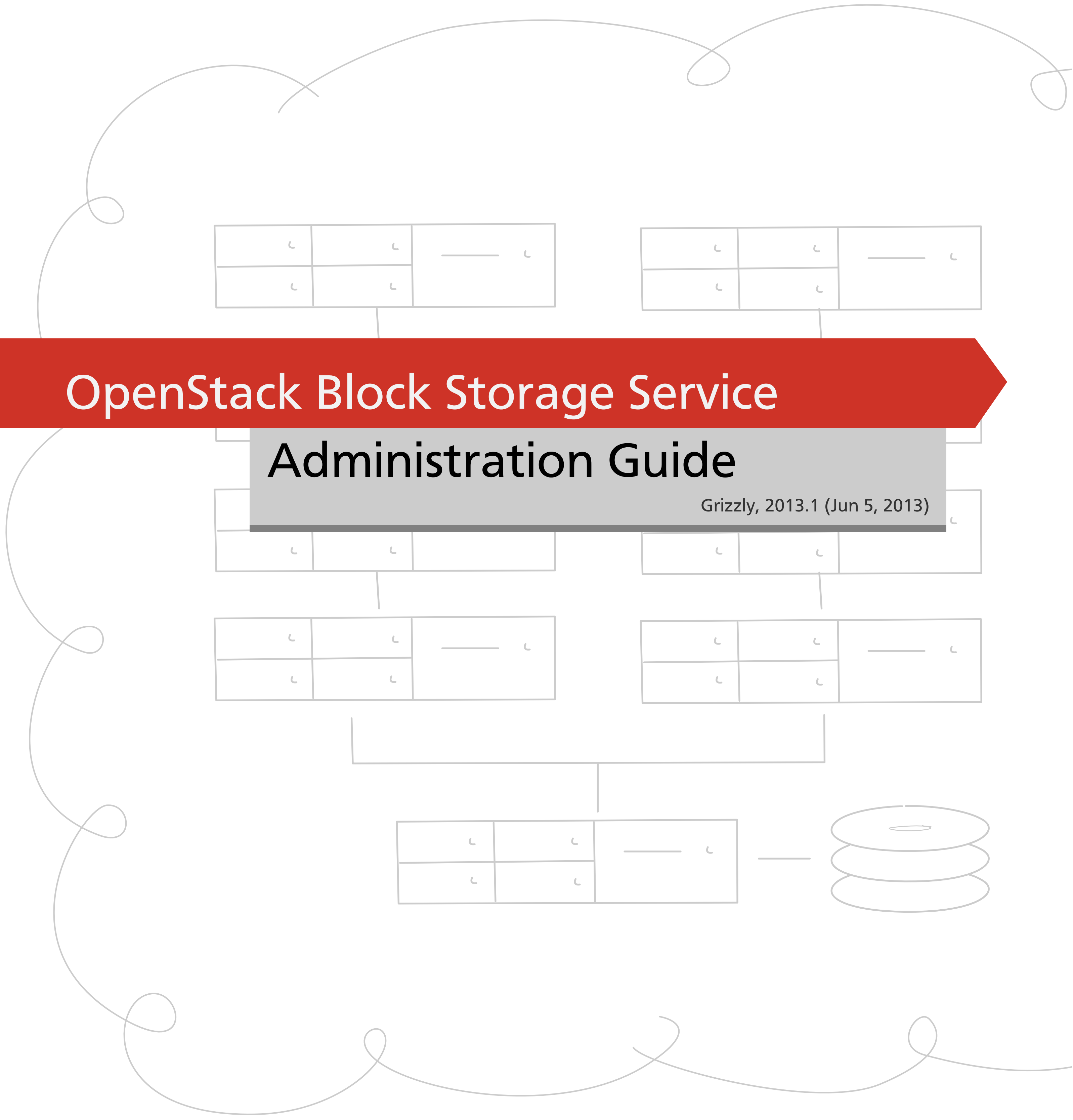


OpenStack Block Storage Service Administration Guide

Grizzly, 2013.1 (Jun 5, 2013)

docs.openstack.org

OpenStack Block Storage Service Administration Guide

Grizzly, 2013.1 (2013-06-05)

This document is for system administrators of the OpenStack Block Storage Service.

Table of Contents

List of Figures

List of Tables

Preface

The OpenStack Block Storage service works though the interaction of a series of daemon processes named cinder-* that reside persistently on the host machine or machines. The binaries can all be run from a single node, or spread across multiple nodes. They can also be run on the same node as other OpenStack services.

To administer the OpenStack Block Storage service, it is helpful to understand a number of concepts. You must make certain choices when you configure the Block Storage service in OpenStack. The bulk of the options come down to two choices, single node or multi-node install. You can read a longer discussion about storage decisions in Storage Decisions in the *OpenStack Operations Guide*.

Document Change History

This version of the guide replaces and obsoletes all previous versions. The following table describes the latest changes:

Revision Date	Summary of Changes
May 16, 2013	• Merges to include more content from Compute Admin Manual.
May 2, 2013	• Fixed broken link.

1. Introduction to the OpenStack Block Storage Service

The OpenStack Block Storage service provides persistent block storage resources that OpenStack Compute instances can consume. This includes secondary attached storage similar to the Amazon Elastic Block Storage (EBS) offering. In addition, you can write images to an OpenStack Block Storage device for OpenStack Compute to use as a bootable persistent instance.

The OpenStack Block Storage service differs slightly from the Amazon EBS offering. The OpenStack Block Storage service does not provide a shared storage solution like NFS. With the OpenStack Block Storage service, you can attach a device to only one instance.

The OpenStack Block Storage service provides:

- **cinder-api**. A WSGI app that authenticates and routes requests throughout the Block Storage service. It supports the OpenStack APIs only, although there is a translation that can be done through Nova's EC2 interface, which calls in to the cinderclient.
- **cinder-scheduler**. Schedules and routes requests to the appropriate volume service. As of Grizzly; depending upon your configuration this may be simple round-robin scheduling to the running volume services, or it can be more sophisticated through the use of the Filter Scheduler. The Filter Scheduler is the default in Grizzly and enables filter on things like Capacity, Availability Zone, Volume Types and Capabilities as well as custom filters.
- **cinder-volume**. Manages Block Storage devices, specifically the back-end devices themselves.
- **cinder-backup** Provides a means to back up a Cinder Volume to OpenStack Object Store (SWIFT).

The OpenStack Block Storage service contains the following components:

- **Backend Storage Devices**. The OpenStack Block Storage service requires some form of back-end storage that the service is built on. The default implementation is to use LVM on a local volume group named "cinder-volumes." In addition to the base driver implementation, the OpenStack Block Storage service also provides the means to add support for other storage devices to be utilized such as external Raid Arrays or other storage appliances.
- **Users and Tenants (Projects)**. The OpenStack Block Storage service is designed to be used by many different cloud computing consumers or customers, basically tenants on a shared system, using role-based access assignments. Roles control the actions that a user is allowed to perform. In the default configuration, most actions do not require a particular role, but this is configurable by the system administrator editing the appropriate `policy.json` file that maintains the rules. A user's access to particular volumes is limited by tenant, but the username and password are assigned per user. Key pairs granting access to a volume are enabled per user, but quotas to control resource consumption across available hardware resources are per tenant.

  For tenants, quota controls are available to limit:

  - The number of volumes that can be created
  - The number of snapshots that can be created
  - The total number of GBs allowed per tenant (shared between snapshots and volumes)

- **Volumes, Snapshots, and Backups**. The basic resources offered by the OpenStack Block Storage service are volumes and snapshots, which are derived from volumes, and backups:
  - **Volumes**. Allocated block storage resources that can be attached to instances as secondary storage or they can be used as the root store to boot instances. Volumes are persistent R/W block storage devices most commonly attached to the Compute node through iSCSI.
  - **Snapshots**. A read-only point in time copy of a volume. The snapshot can be created from a volume that is currently in use (through the use of '–force True') or in an available state. The snapshot can then be used to create a new volume through create from snapshot.
  - **Backups**. An archived copy of a volume currently stored in OpenStack Object Storage (Swift).

2. Managing Volumes

The OpenStack Block Storage service enables you to add extra block-level storage to your OpenStack Compute instances. This service is similar to the Amazon EC2 Elastic Block Storage (EBS) offering.

The default OpenStack Block Storage service implementation is an iSCSI solution that uses Logical Volume Manager (LVM) for Linux.

Note

The OpenStack Block Storage service is not a shared storage solution like SAN of NFS, where you can attach a volume to multiple servers. With the OpenStack Block Storage service, you can attach a volume to only one instance at a time.

The OpenStack Block Storage service also provides drivers that enable you to use several vendors' back-end storage devices, in addition to or instead of the base LVM implementation.

The following high-level procedure shows you how to create and attach a volume to a server instance.

Procedure 2.1. To create and attach a volume to a server instance:

1. You must configure both OpenStack Compute and the OpenStack Block Storage service through the `cinder.conf` file.
2. Create a volume through the **cinder create** command. This command creates an LV into the volume group (VG) "cinder-volumes."
3. Attach the volume to an instance through the **nova volume-attach** command. This command creates a unique iSCSI IQN that is exposed to the compute node.
    a. The compute node, which runs the instance, now has an active ISCSI session and new local storage (usually a /dev/sdX disk).
    b. libvirt uses that local storage as storage for the instance. The instance get a new disk, usually a /dev/vdX disk.

For this particular walk through, there is one cloud controller running `nova-api`, `nova-scheduler`, `nova-objectstore`, `nova-network` and `cinder-*` services. There are two additional compute nodes running `nova-compute`. The walk through uses a custom partitioning scheme that carves out 60GB of space and labels it as LVM. The network uses `FlatManger` is the `NetworkManager` setting for OpenStack Compute (Nova).

Please note that the network mode doesn't interfere at all with the way cinder works, but networking must be set up for cinder to work. Please refer to Networking Administration for more details.

To set up Compute to use volumes, ensure that Block Storage is installed along with lvm2. The guide will be split in four parts :

- Installing the Block Storage service on the cloud controller.
- Configuring the `cinder-volumes` volume group on the compute nodes.
- Troubleshooting your installation.
- Backup your nova volumes.

Installing and configuring Block Storage (Cinder)

Install the packages for OpenStack Block Storage (cinder) on the cloud controller.

```
$ sudo apt-get install cinder-api
cinder-scheduler cinder-volume open-iscsi python-cinderclient tgt
```

```
# yum install openstack-cinder openstack-cinder-doc \
        iscsi-initiator-utils scsi-target-utils
```

Edit /etc/cinder/api-paste.ini (filter authtoken).

```
[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 192.168.206.130
service_port = 5000
auth_host = 192.168.206.130
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = cinder
admin_password = openstack
```

Edit /etc/cinder/cinder.conf to reflect your settings.

```

[DEFAULT]
rootwrap_config=/etc/cinder/rootwrap.conf
sql_connection = mysql://cinder:openstack@192.168.127.130/cinder
api_paste_config = /etc/cinder/api-paste.ini

iscsi_helper=tgtadm
volume_name_template = volume-%s
volume_group = cinder-volumes
verbose = True
auth_strategy = keystone
#osapi_volume_listen_port=5900
```

Configure messaging (RabbitMQ or Qpid) also in /etc/cinder/cinder.conf.

```
# Ubuntu
rabbit_host = 10.10.10.10
rabbit_port = 5672
rabbit_userid = rabbit
rabbit_password = secure_password
rabbit_virtual_host = /nova
```

```
# Red Hat, Fedora, CentOS
qpid_hostname=192.168.206.130
```

Verify entries in `nova.conf`. The volume_api_class setting is the default setting for grizzly.

```
volume_api_class=nova.volume.cinder.API
```

Set up the cinder database.

```
CREATE DATABASE cinder;
GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

Add a filter entry to the devices section `/etc/lvm/lvm.conf` to keep LVM from scanning devices used by virtual machines. NOTE: You must add every physical volume that is needed for LVM on the Cinder host. You can get a list by running **pvdisplay**. Each item in the filter array starts with either an "a" for accept, or an "r" for reject. Physical volumes that are needed on the Cinder host begin with "a". The array must end with "`r/.*/`"

```
devices {
...
filter = [ "a/sda1/", "a/sdb1/", "r/.*/"]
...
}
```

Setup the target file *NOTE: `$state_path=/var/lib/cinder/` and `$volumes_dir=$state_path/volumes` by default and path MUST exist!*.

```
$ sudo sh -c "echo 'include $volumes_dir/*' >> /etc/tgt/conf.d/cinder.conf"
```

Restart the **tgt** service.

```
$ sudo restart tgt
```

Populate the database.

```
$ sudo cinder-manage db sync
```

Restart the services.

```
$ sudo service cinder-volume restart
$ sudo service cinder-api restart
$ sudo service cinder-scheduler restart
```

Create a 1 GB test volume.

```
$ cinder create --display_name test 1
$ cinder list
```

```
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
|                  ID                  |   Status  | Display Name | Size |
 Volume Type | Attached to |
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
| 5bbad3f9-50ad-42c5-b58c-9b6b63ef3532 | available |     test     |  1   |
 None    |             |
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
```

Backup your Block Storage disks

While you can use the snapshot functionality (using LVM snapshot), you can also back up your volumes. The advantage of this method is that it reduces the size of the backup; only existing data will be backed up, instead of the entire volume. For this example, assume that a 100 GB volume has been created for an instance, while only 4 gigabytes are used. This process will back up only those 4 gigabytes, with the following tools:

1. **lvm2**, directly manipulates the volumes.
2. **kpartx** discovers the partition table created inside the instance.
3. **tar** creates a minimum-sized backup
4. **sha1sum** calculates the backup checksum, to check its consistency

1- Create a snapshot of a used volume

- In order to backup our volume, we first need to create a snapshot of it. An LVM snapshot is the exact copy of a logical volume, which contains data in a frozen state. This prevents data corruption, because data will not be manipulated during the process of creating the volume itself. Remember the volumes created through a **nova volume-create** exist in an LVM's logical volume.

  Before creating the snapshot, ensure that you have enough space to save it. As a precaution, you should have at least twice as much space as the potential snapshot size. If insufficient space is available, there is a risk that the snapshot could become corrupted.

  Use the following command to obtain a list of all volumes.

  ```
  $ lvdisplay
  ```

  In this example, we will refer to a volume called `volume-00000001`, which is a 10GB volume. This process can be applied to all volumes, not matter their size. At the end of the section, we will present a script that you could use to create scheduled backups. The script itself exploits what we discuss here.

  First, create the snapshot; this can be achieved while the volume is attached to an instance :

  ```
  $ lvcreate --size 10G --snapshot --name volume-00000001-snapshot /dev/nova-volumes/volume-00000001
  ```

  We indicate to LVM we want a snapshot of an already existing volume with the `--snapshot` configuration option. The command includes the size of the space reserved for the snapshot volume, the name of the snapshot, and the path of an already existing volume (In most cases, the path will be `/dev/nova-volumes/`*`$volume_name`*).

  The size doesn't have to be the same as the volume of the snapshot. The size parameter designates the space that LVM will reserve for the snapshot volume. As a precaution, the size should be the same as that of the original volume, even if we know the whole space is not currently used by the snapshot.

We now have a full snapshot, and it only took few seconds !

Run **lvdisplay** again to verify the snapshot. You should see now your snapshot :

```
                  --- Logical volume ---
  LV Name                /dev/nova-volumes/volume-00000001
  VG Name                nova-volumes
  LV UUID                gI8hta-p21U-IW2q-hRN1-nTzN-UC2G-dKbdKr
  LV Write Access        read/write
  LV snapshot status     source of
                         /dev/nova-volumes/volume-00000026-snap [active]
  LV Status              available
  # open                 1
  LV Size                15,00 GiB
  Current LE             3840
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           251:13

  --- Logical volume ---
  LV Name                /dev/nova-volumes/volume-00000001-snap
  VG Name                nova-volumes
  LV UUID                HlW3Ep-g5I8-KGQb-IRvi-IRYU-lIKe-wE9zYr
  LV Write Access        read/write
  LV snapshot status     active destination for /dev/nova-volumes/
volume-00000026
  LV Status              available
  # open                 0
  LV Size                15,00 GiB
  Current LE             3840
  COW-table size         10,00 GiB
  COW-table LE           2560
  Allocated to snapshot  0,00%
  Snapshot chunk size    4,00 KiB
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           251:14
```

2- Partition table discovery

- If we want to exploit that snapshot with the **tar** program, we first need to mount our partition on the Block Storage server.

  kpartx is a small utility which performs table partition discoveries, and maps it. It can be used to view partitions created inside the instance. Without using the partitions created inside instances, we won' t be able to see its content and create efficient backups.

```
$ kpartx -av /dev/nova-volumes/volume-00000001-snapshot
```

If no errors are displayed, it means the tools has been able to find it, and map the partition table. Note that on a Debian flavor distro, you could also use **apt-get install kpartx**.

You can easily check the partition table map by running the following command:

```
$ ls /dev/mapper/nova*
```

You should now see a partition called `nova--volumes-volume--00000001--snapshot1`

If you created more than one partition on that volumes, you should have accordingly several partitions; for example. `nova--volumes-volume--00000001--snapshot2`, `nova--volumes-volume--00000001--snapshot3` and so forth.

We can now mount our partition :

```
$ mount /dev/mapper/nova--volumes-volume--volume--00000001--snapshot1 /mnt
```

If there are no errors, you have successfully mounted the partition.

You should now be able to directly access the data that were created inside the instance. If you receive a message asking you to specify a partition, or if you are unable to mount it (despite a well-specified filesystem) there could be two causes :

- You didn't allocate enough space for the snapshot
- **kpartx** was unable to discover the partition table.

Allocate more space to the snapshot and try the process again.

3- Use tar in order to create archives

- Now that the volume has been mounted, you can create a backup of it :

```
$ tar --exclude={"lost+found","some/data/to/exclude"} -czf volume-00000001.
tar.gz -C /mnt/ /backup/destination
```

This command will create a tar.gz file containing the data, *and data only*. This ensures that you do not waste space by backing up empty sectors.

4- Checksum calculation I

- You should always have the checksum for your backup files. The checksum is a unique identifier for a file.

  When you transfer that same file over the network, you can run another checksum calculation. If the checksums are different, this indicates that the file is corrupted; thus, the checksum provides a method to ensure your file has not been corrupted during its transfer.

The following command runs a checksum for our file, and saves the result to a file :

```
$ sha1sum volume-00000001.tar.gz > volume-00000001.checksum
```

Be aware the **sha1sum** should be used carefully, since the required time for the calculation is directly proportional to the file's size.

For files larger than ~4-6 gigabytes, and depending on your CPU, the process may take a long time.

5- After work cleaning

- Now that we have an efficient and consistent backup, the following commands will clean up the file system.

  1. Unmount the volume: **unmount /mnt**

  2. Delete the partition table: **kpartx -dv /dev/nova-volumes/volume-00000001-snapshot**

  3. Remove the snapshot: **lvremove -f /dev/nova-volumes/volume-00000001-snapshot**

  And voila :) You can now repeat these steps for every volume you have.

6- Automate your backups

Because you can expect that more and more volumes will be allocated to your Block Storage service, you may want to automate your backups. This script here will assist you on this task. The script performs the operations from the previous example, but also provides a mail report and runs the backup based on the `backups_retention_days` setting. It is meant to be launched from the server which runs the Block Storage component.

Here is an example of a mail report:

```
Backup Start Time - 07/10 at 01:00:01
Current retention - 7 days

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-00000019/
volume-00000019_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-00000019 - 0 h 1 m and 21 seconds. Size - 3,5G

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-0000001a/
volume-0000001a_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-0000001a - 0 h 4 m and 15 seconds. Size - 6,9G
---------------------------------------
Total backups size - 267G - Used space : 35%
Total execution time - 1 h 75 m and 35 seconds
```

The script also provides the ability to SSH to your instances and run a mysqldump into them. In order to make this to work, ensure the connection via the nova's project keys is enabled. If you don't want to run the mysqldumps, you can turn off this functionality by adding `enable_mysql_dump=0` to the script.

Troubleshoot your cinder installation

This section is intended to help solve some basic and common errors that are encountered during setup and configuration of Cinder. The focus here is on failed creation of volumes. The most important thing to know is where to look in case of a failure. There are two log files that are especially helpful in the case of a volume creation failure. The first is the `cinder-api` log, and the second is the `cinder-volume` log.

The `cinder-api` log is useful in determining if you have endpoint or connectivity issues. If you send a request to create a volume and it fails, it's a good idea to look here first and see if the request even made it to the Cinder service. If the request seems to be logged, and there are no errors or trace-backs then you can move to the `cinder-volume` log and look for errors or trace-backs there.

There are some common issues to look out for. The following describes some common issues hit during configuration and some suggested solutions.

Create commands are in cinder-api log with no error

- **`state_path` and `volumes_dir` settings**

  Cinder uses **tgtd** as the default iscsi helper and implements persistent targets. This means that in the case of a tgt restart or even a node reboot your existing volumes on that node will be restored automatically with their original IQN.

  In order to make this possible the iSCSI target information needs to be stored in a file on creation that can be queried in case of restart of the tgt daemon. By default, Cinder uses a `state_path` variable, which if installing via Yum or APT should be set to `/var/lib/cinder/`. The next part is the `volumes_dir` variable, by default this just simply appends a "`volumes`" directory to the `state_path`. The result is a file-tree `/var/lib/cinder/volumes/`.

  While this should all be handled for you by you installer, it can go wrong. If you're having trouble creating volumes and this directory does not exist you should see an error message in the `cinder-volume` log indicating that the `volumes_dir` doesn't exist, and it should give you information to specify what path exactly it was looking for.

- **persistent tgt include file**

  Along with the `volumes_dir` mentioned above, the iSCSI target driver also needs to be configured to look in the correct place for the persist files. This is a simple entry in `/etc/tgt/conf.d`, and you should have created this when you went through the install guide. If you haven't or you're running into issues, verify that you have a file `/etc/tgt/conf.d/cinder.conf`.

  If the file is not there, you can create it easily by doing the following:

  ```
  sudo sh -c "echo 'include /var/lib/cinder/volumes/*' >> /etc/tgt/conf.d/
  cinder.conf"
  ```

No sign of attach call in the `cinder-api` log

This is most likely going to be a minor adjustment to your `nova.conf` file. Make sure that your `nova.conf` has the following entry:

```
volume_api_class=nova.volume.cinder.API
```

And make certain that you EXPLICITLY set enabled_apis as the default will include osapi_volume:

```
enabled_apis=ec2,osapi_compute,metadata
```

Failed to create iscsi target error in the `cinder-volume.log`

```
2013-03-12 01:35:43 1248 TRACE cinder.openstack.common.rpc.amqp
 ISCSITargetCreateFailed: Failed to create iscsi target for volume
 volume-137641b2-af72-4a2f-b243-65fdccd38780.
```

You may see this error in `cinder-volume.log` after trying to create a volume that is 1 GB. To fix this issue:

Change content of the `/etc/tgt/targets.conf` from "include /etc/tgt/conf.d/*.conf" to: include /etc/tgt/conf.d/cinder_tgt.conf:

```
            include /etc/tgt/conf.d/cinder_tgt.conf
            include /etc/tgt/conf.d/cinder.conf
            default-driver iscsi
```

Then restart tgt and `cinder-*` services so they pick up the new configuration.

Configure a Multiple-Storage Backend

This section presents the multi-backend storage feature introduced with the Grizzly release. Multi-backend allows the creation of several backend storage solutions serving the same OpenStack Compute configuration. Basically, multi-backend launches one `cinder-volume` per backend.

In a multi-backend configuration, each backend has a name (`volume_backend_name`). Several backends can have the same name. In that case, the scheduler properly decides in which backend the volume has to be created.

The name of the backend is declared as an extra-specification of a volume type (e.g. `volume_backend_name=LVM_iSCSI`). At a volume creation, according to the volume type specified by the user, the scheduler will choose an appropriate backend to handle the request.

Enable Multi-Backend

The multi-backend configuration is done into the `cinder.conf` file. The `enabled_backends` flag has to be set up. This flag defines the names (separated by a comma) of the config groups for the different backends: one name is associated to one config group for a backend (e.g. `[lvmdriver-1]`).

Note

The config group name is not related to the `volume_backend_name`.

The options for a config group have to be defined in the group (or default options will be used). All the standard Cinder configuration options (`volume_group`, `volume_driver`, etc) may be used in a config group. Config values in the `[DEFAULT]` config group will not be used.

The following example shows three backends:

```
# a list of backends that will be served by this compute node
enabled_backends=lvmdriver-1,lvmdriver-2,lvmdriver-3
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
[lvmdriver-2]
volume_group=cinder-volumes-2
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
[lvmdriver-3]
volume_group=cinder-volumes-3
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI_b
```

In this configuration `lvmdriver-1` and `lvmdriver-2` have the same `volume_backend_name`. This means that, if a volume creation requests the `LVM_iSCSI` backend name, the scheduler will choose between `lvmdriver-1` and `lvmdriver-2` which one is the most suitable. This is done thanks to the capacity filter scheduler which is enabled by default (the following section gives more information on that point). In addition, this example presents a third backend named `lvmdriver-3`. This one has a different backend name.

Cinder Scheduler Configuration With Multi-Backend

Multi-backend has to be used with `filter_scheduler` enabled. Filter scheduler acts in two steps:

1. First, filter scheduler filters the available backends. By default, `AvailabilityZoneFilter`, `CapacityFilter` and `CapabilitiesFilter` are enabled.

2. Secondly, filter scheduler weights the previously filtered backends. By default, `CapacityWeigher` is enabled. The `CapacityWeigher` attributes high score to backends with the most available space.

According to the filtering and weighing, the scheduler will be able to pick "the best" backend in order to handle the request. In that way, filter scheduler achieves the goal that one can explicitly creates volume on specifics backends using volume types.

Note

To enable the filter scheduler, the following line has to be added into the `cinder.conf` configuration file:

```
scheduler_driver=cinder.scheduler.filter_scheduler.FilterScheduler
```

However, `filter_scheduler` is the default Cinder Scheduler in Grizzly, this line is not mandatory.

Volume Type

Before using it, a volume type has to be declared to Cinder. This can be done by the following command:

```
$ cinder --os-username admin --os-tenant-name admin type-create lvm
```

Then, an extra-specification have to be created to link the volume type to a backend name. This can be done by the following command:

```
$ cinder --os-username admin --os-tenant-name admin type-key lvm set
 volume_backend_name=LVM_iSCSI
```

In this example we have created a volume type named `lvm` with `volume_backend_name=LVM_iSCSI` as extra-specifications.

We complete this example by creating another volume type:

```
$ cinder --os-username admin --os-tenant-name admin type-create lvm_gold
$ cinder --os-username admin --os-tenant-name admin type-key lvm_gold set
 volume_backend_name=LVM_iSCSI_b
```

This second volume type is named `lvm_gold` and has `LVM_iSCSI_b` as backend name.

Note

To list the extra-specifications, use the following command line:

```
$ cinder --os-username admin --os-tenant-name admin extra-specs-list
```

Note

If a volume type points to a `volume_backend_name` which does not exist in the Cinder configuration, the `filter_scheduler` will return an error mentioning that it is not able to find a valid host with the suitable backend.

Usage

When creating a volume, the volume type has to be specified. The extra-specifications of the volume type will be used to determine which backend has to be used.

```
cinder create --volume_type lvm --display_name test_multi_backend 1
```

Considering the `cinder.conf` described above, the scheduler will create this volume on `lvmdriver-1` or `lvmdriver-2`.

```
cinder create --volume_type lvm_gold --display_name test_multi_backend 1
```

This second volume will be created on `lvmdriver-3`.

Adding Block Storage Nodes

To offer more storage to your tenant's VMs, add another volume node running cinder services by following these steps.

1. Install the required packages for cinder.
2. Create a volume group called cinder-volumes (configurable using the `cinder_volume` parameter in `cinder.conf`).
3. Configure tgtd with its `targets.conf` file and start the `tgtd` service.
4. Connect the node to the Block Storage (cinder) database by configuring the `cinder.conf` file with the connection information.
5. Make sure the `iscsi_ip_address` setting in `cinder.conf` matches the public IP of the node you're installing, then restart the cinder services.

When you issue a **cinder-manage host list** command you should see the new volume node listed. If not, look at the logs in `/var/log/cinder/volume.log` for issues.

Boot From Volume

In some cases, instances can be stored and run from inside volumes. This is explained in further detail in the Boot From Volume section of the Compute Admin manual.

3. Volume Drivers

You can alter the default cinder-volume behavior by using different volume drivers. These are included in the Cinder repository (https://github.com/openstack/cinder). To set a volume driver, use the `volume_driver` flag. The default is:

```
volume_driver=cinder.volume.driver.ISCSIDriver
iscsi_helper=tgtadm
```

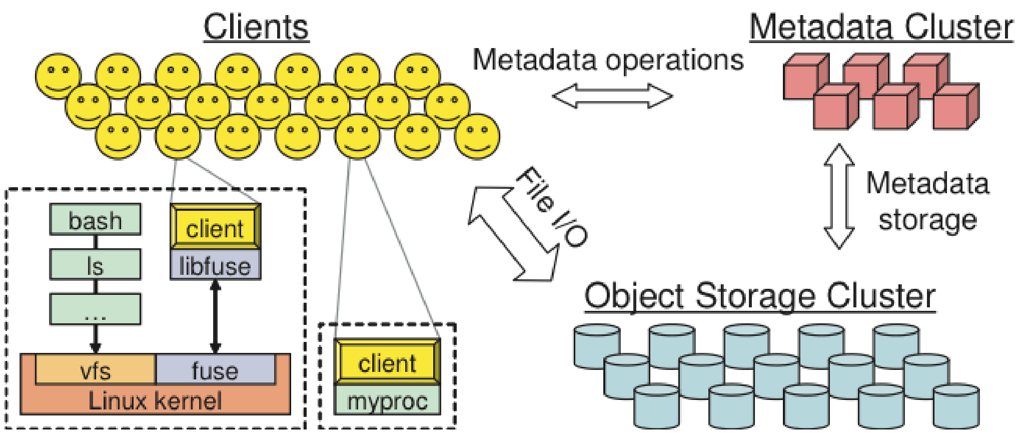
Ceph RADOS Block Device (RBD)

By Sebastien Han from http://www.sebastien-han.fr/blog/2012/06/10/introducing-ceph-to-openstack/

If you are using KVM or QEMU as your hypervisor, the Compute service can be configured to use Ceph's RADOS block devices (RBD) for volumes.

Ceph is a massively scalable, open source, distributed storage system. It is comprised of an object store, block store, and a POSIX-compliant distributed file system. The platform is capable of auto-scaling to the exabyte level and beyond, it runs on commodity hardware, it is self-healing and self-managing, and has no single point of failure. Ceph is in the Linux kernel and is integrated with the OpenStack™ cloud operating system. As a result of its open source nature, this portable storage platform may be installed and used in public or private clouds.

Figure 3.1. Ceph-architecture.png



RADOS?

You can easily get confused by the denomination: Ceph? RADOS?

RADOS: Reliable Autonomic Distributed Object Store is an object storage. RADOS takes care of distributing the objects across the whole storage cluster and replicating them for fault tolerance. It is built with 3 major components:

- *Object Storage Device (ODS)*: the storage daemon - RADOS service, the location of your data. You must have this daemon running on each server of your cluster. For each OSD you can have an associated hard drive disks. For performance purposes, it’s usually better to pool your hard drive disk with raid arrays, LVM or btrfs pooling. By default, three pools are created: data, metadata and RBD.

- *Meta-Data Server (MDS)*: this is where the metadata are stored. MDSs builds POSIX file system on top of objects for Ceph clients. However if you are not using the Ceph File System, you do not need a meta data server.

- *Monitor (MON)*: this lightweight daemon handles all the communications with the external applications and the clients. It also provides a consensus for distributed decision making in a Ceph/RADOS cluster. For instance when you mount a Ceph shared on a client you point to the address of a MON server. It checks the state and the consistency of the data. In an ideal setup you need to run at least 3 `ceph-mon` daemons, on separate servers.

Ceph developers recommend you use btrfs as a file system for the storage. Using XFS is also possible and might be a better alternative for production environments. Neither Ceph nor Btrfs are ready for production, so it could be risky to use them in combination. This is why XFS is an excellent alternative to btrfs. The ext4 file system is also compatible but doesn’t take advantage of all the power of Ceph.

Note

We recommend configuring Ceph to use the XFS file system in the near term, and btrfs in the long term once it is stable enough for production.

See ceph.com/docs/master/rec/file system/ for more information about usable file systems.

Ways to store, use and expose data

There are several ways to store and access your data.

- *RADOS*: as an object, default storage mechanism.

- *RBD*: as a block device. The Linux kernel RBD (rados block device) driver allows striping a Linux block device over multiple distributed object store data objects. It is compatible with the kvm RBD image.

- *CephFS*: as a file, POSIX-compliant file system.

Ceph exposes its distributed object store (RADOS) and it can be accessed via multiple interfaces:

- *RADOS Gateway*: Swift and Amazon-S3 compatible RESTful interface. See RADOS_Gateway for further information.

- *librados* and the related C/C++ bindings.

- *rbd and QEMU-RBD*: Linux kernel and QEMU block devices that stripe data across multiple objects.

For detailed installation instructions and benchmarking information, see http://www.sebastien-han.fr/blog/2012/06/10/introducing-ceph-to-openstack/.

EMC SMI-S iSCSI Driver

The EMCISCSIDriver is based on the existing ISCSIDriver, with the ability to create/delete and attach/detach volumes and create/delete snapshots, and so on.

The EMCISCSIDriver executes the volume operations by communicating with the backend EMC storage. It uses a CIM client in python called PyWBEM to make CIM operations over HTTP.

The EMC CIM Object Manager (ECOM) is packaged with the EMC SMI-S Provider. It is a CIM server that allows CIM clients to make CIM operations over HTTP, using SMI-S in the backend for EMC storage operations.

The EMC SMI-S Provider supports the SNIA Storage Management Initiative (SMI), an ANSI standard for storage management. It supports VMAX and VNX storage systems.

System Requirements

EMC SMI-S Provider V4.5.1 and higher is required. SMI-S can be downloaded from EMC's Powerlink web site. Refer to the EMC SMI-S Provider release notes for installation instructions.

EMC storage VMAX Family and VNX Series are supported.

Supported Operations

The following operations will be supported on both VMAX and VNX arrays:

- Create volume
- Delete volume
- Attach volume
- Detach volume
- Create snapshot
- Delete snapshot
- Create cloned volume
- Copy image to volume
- Copy volume to image

The following operations will be supported on VNX only:

- Create volume from snapshot

Only thin provisioning is supported.

Preparation

- Install python-pywbem package. For example:

```
$sudo apt-get install python-pywbem
```

- Setup SMI-S. Download SMI-S from PowerLink and install it following the instructions of SMI-S release notes. Add your VNX/VMAX/VMAXe arrays to SMI-S following the SMI-S release notes.
- Register with VNX.
- Create Masking View on VMAX.

Register with VNX

For a VNX volume to be exported to a Compute node, the node needs to be registered with VNX first.

On the Compute node `1.1.1.1`, do the following (assume `10.10.61.35` is the iscsi target):

```
$ sudo /etc/init.d/open-iscsi start
```

```
$ sudo iscsiadm -m discovery -t st -p 10.10.61.35
```

```
$ cd /etc/iscsi
```

```
$ sudo more initiatorname.iscsi
```

```
$ iscsiadm -m node
```

Log in to VNX from the Compute node using the target corresponding to the SPA port:

```
$ sudo iscsiadm -m node -T iqn.1992-04.com.emc:cx.apm01234567890.a0 -p 10.10.
61.35 -l
```

Assume `iqn.1993-08.org.debian:01:1a2b3c4d5f6g` is the initiator name of the Compute node. Login to Unisphere, go to VNX00000->Hosts->Initiators, Refresh and wait until initiator `iqn.1993-08.org.debian:01:1a2b3c4d5f6g` with SP Port `A-8v0` appears.

Click the "Register" button, select "CLARiiON/VNX" and enter the host name `myhost1` and IP address `myhost1`. Click Register. Now host `1.1.1.1` will appear under Hosts->Host List as well.

Log out of VNX on the Compute node:

```
$ sudo iscsiadm -m node -u
```

Log in to VNX from the Compute node using the target corresponding to the SPB port:

```
$ sudo iscsiadm -m node -T iqn.1992-04.com.emc:cx.apm01234567890.b8 -p 10.10.
10.11 -l
```

In Unisphere register the initiator with the SPB port.

Log out:

```
$ sudo iscsiadm -m node -u
```

Create Masking View on VMAX

For VMAX, user needs to do initial setup on the Unisphere for VMAX server first. On the Unisphere for VMAX server, create initiator group, storage group, port group, and put them in a masking view. Initiator group contains the initiator names of the openstack hosts. Storage group should have at least 6 gatekeepers.

Config file `cinder.conf`

Make the following changes in `/etc/cinder/cinder.conf`.

For VMAX, we have the following entries where `10.10.61.45` is the IP address of the VMAX iscsi target.

```
iscsi_target_prefix = iqn.1992-04.com.emc
iscsi_ip_address = 10.10.61.45
volume_driver = cinder.volume.emc.EMCISCSIDriver
cinder_emc_config_file = /etc/cinder/cinder_emc_config.xml
```

For VNX, we have the following entries where `10.10.61.35` is the IP address of the VNX iscsi target.

```
iscsi_target_prefix = iqn.2001-07.com.vnx
iscsi_ip_address = 10.10.61.35
volume_driver = cinder.volume.emc.EMCISCSIDriver
cinder_emc_config_file = /etc/cinder/cinder_emc_config.xml
```

Restart the cinder-volume service.

Config file `cinder_emc_config.xml`

Create the file `/etc/cinder/cinder_emc_config.xml`. We don't need to restart service for this change.

For VMAX, we have the following in the xml file:

```
<?xml version='1.0' encoding='UTF-8'?>
<EMC>
<StorageType>xxxx</StorageType>
<MaskingView>xxxx</MaskingView>
<EcomServerIp>x.x.x.x</EcomServerIp>
<EcomServerPort>xxxx</EcomServerPort>
<EcomUserName>xxxxxxxx</EcomUserName>
<EcomPassword>xxxxxxxx</EcomPassword>
</EMC>
```

For VNX, we have the following in the xml file:

```
<?xml version='1.0' encoding='UTF-8'?>
<EMC>
<StorageType>xxxx</StorageType>
<EcomServerIp>x.x.x.x</EcomServerIp>
<EcomServerPort>xxxx</EcomServerPort>
<EcomUserName>xxxxxxxx</EcomUserName>
<EcomPassword>xxxxxxxx</EcomPassword>
</EMC>
```

MaskingView is required for attaching VMAX volumes to an OpenStack VM. A Masking View can be created using Unisphere for VMAX. The Masking View needs to have an Initiator Group that contains the initiator of the OpenStack compute node that hosts the VM.

StorageType is the thin pool where user wants to create the volume from. Only thin LUNs are supported by the plugin. Thin pools can be created using Unisphere for VMAX and VNX.

EcomServerIp and EcomServerPort are the IP address and port number of the ECOM server which is packaged with SMI-S. EcomUserName and EcomPassword are credentials for the ECOM server.

GlusterFS Driver

GlusterFS is an open-source scalable distributed filesystem that is able to grow to petabytes and beyond in size. More information can be found on Gluster's homepage.

This driver enables use of GlusterFS in a similar fashion as the NFS driver. It supports basic volume operations, and like NFS, does not support snapshot/clone.

Note

You must use a Linux kernel of version 3.4 or greater (or version 2.6.32 or greater in RHEL/CentOS 6.3+) when working with Gluster-based volumes. See Bug 1177103 for more information.

To use Cinder with GlusterFS, first set the `volume_driver` in `cinder.conf`:

```
volume_driver=cinder.volume.drivers.glusterfs.GlusterfsDriver
```

Table 3.1. List of configuration flags for GlusterFS

Flag Name	Type	Default	Description
`glusterfs_shares_config`	Mandatory		File with the list of available gluster shares.
`glusterfs_mount_point_base`	Optional	`$state_path/mnt`	Base dir where gluster expected to be mounted.
`glusterfs_disk_util`	Optional	`df`	Use du or `df` for free space calculation.
`glusterfs_sparsed_volumes`	Optional	True	Create volumes as sparsed files which take no space. If set to False, volume is created using "**dd**" or a similar command to create the full-sized file, so volume creation takes a greater amount of time.

HP 3PAR Fibre Channel and iSCSI Drivers

The `HP3PARFCDriver` and `HP3PARISCSIDriver` are based on the Block Storage (Cinder) plug-in architecture. The drivers execute the volume operations by communicating with the HP 3PAR storage system over HTTP/HTTPS and SSH connections. The HTTP/HTTPS communications use the `hp3parclient`, which is part of the Python standard library.

For information about managing HP 3PAR storage systems, refer to the HP 3PAR user documentation.

System Requirements

To use the HP 3PAR drivers, install the following software and components on the HP 3PAR storage system:

- HP 3PAR Operating System software version 3.1.2 or higher
- HP 3PAR Web Services API Server must be enabled and running
- One domain and one Common Provisioning Group (CPG)
- Additionally, you must intall the `hp3parclient` from the Python standard library on the system with the enabled Block Storage volume drivers.

Supported Operations

- Create volumes.
- Delete volumes.
- Attach volumes.
- Detach volumes.
- Create snapshots.
- Delete snapshots.
- Create volumes from snapshots.
- Create cloned volumes.

- Copy images to volumes (HP 3PAR iSCSI driver only).
- Copy volumes to images (HP 3PAR iSCSI driver only).

Volume type support for both HP 3PAR drivers includes the ability to set the following capabilities in the OpenStack Cinder API `cinder.api.contrib.types_extra_specs` volume type extra specs extension module:

- `cpg`
- `snap_cpg`
- `provisioning`
- `persona`

The key values are case sensitive. Run **cinder help type-key** for information about setting the key value pairings and associating them with a volume type.

If volume types are not used or a particular key is not set for a volume type, the following defaults are used.

- `cpg` - Defaults to the `hp3par_cpg` setting in the `cinder.conf` file.
- `snap_cpg` - Defaults to the `hp3par_snap` setting in the `cinder.conf` file. If `hp3par_snap` is not set, it defaults to the `hp3par_cpg` setting.
- `provisioning` - Defaults to thin provisioning, the valid values are `thin` and `full`.
- `persona` - Defaults to the `1 – Generic` persona. The valid values are, `1 – Generic`, `2 - Generic-ALUA`, `6 - Generic-legacy`, `7 - HPUX-legacy`, `8 - AIX-legacy`, `9 – EGENERA`, `10 - ONTAP-legacy`, and `11 – VMware`.

Enabling the HP 3PAR Fibre Channel and iSCSI Drivers

The `HP3PARFCDriver` and `HP3PARISCSIDriver` are installed with the OpenStack software.

1. Install the `hp3parclient` Python package on the OpenStack Block Storage system.

   ```
   $sudo pip install hp3parclient
   ```

2. Verify that the HP 3PAR Web Services API server is enabled and running on the HP 3PAR storage system.

   a. Log onto the Service Processor with administrator access.

      ```
      #ssh 3paradm@<SP IP Address>
      ```

   b. View the current state of the Web Services API Server.

      ```
      #showwsapi
      ```

      ```
      -Service- -State- -HTTP_State-
              HTTP_Port -HTTPS_State- HTTPS_Port -Version-
      ```

      ```
      Enabled   Active Enabled       8008
      ```

```
        Enabled        8080          1.1
```

c. If the Web Services API Server is disabled, start it.

```
#startwsapi
```

3. If the HTTP or HTTPS state is disabled, enable one of them.

```
#setwsapi -http enable
```

or

```
#setwsapi -https enable
```

Note

To stop the Web Services API Server, use the stopwsapi command. For other options run the **setwsapi –h** command.

4. If you are not using an existing domain and CPG, create a domain and CPG on the HP 3PAR storage system to be used as the default location for creating volumes.

5. Make the following changes in the `/etc/cinder/cinder.conf` file.

```

                        ## REQUIRED SETTINGS
# 3PAR WS API Server URL
  hp3par_api_url=https://10.10.0.141:8080/api/v1

# 3PAR Super user username
  hp3par_username=3paradm

# 3PAR Super user password
  hp3par_password=3parpass

# 3PAR domain to use
  hp3par_domain=TEST_DOMAIN

# 3PAR CPG to use for volume creation
  hp3par_cpg=OpenStackCPG_RAID5_NL

# IP address of SAN controller for SSH access to the array
  san_ip=10.10.22.241

# Username for SAN controller for SSH access to the array
  san_login=3paradm

# Password for SAN controller for SSH access to the array
  san_password=3parpass

# FIBRE CHANNEL(uncomment the next line to enable the FC driver)
  # volume_driver=cinder.volume.drivers.san.hp.hp_3par_fc.HP3PARFCDriver

# iSCSI (uncomment the next line to enable the iSCSI driver and the
 iscsi_ip_address)
  # volume_driver=cinder.volume.drivers.san.hp.hp_3par_iscsi.
HP3PARISCSIDriver

# The port that the iSCSI daemon is listening on
```

```
  #iscsi_ip_address="10.10.220.253"
                           ## OPTIONAL SETTING
# Enable HTTP debugging to 3PAR
  hp3par_debug=False

# The CPG to use for Snapshots for volumes. If empty hp3par_cpg will be
 used.
  hp3par_snap_cpg=OpenStackSNAP_CPG

# Time in hours to retain a snapshot. You can't delete it before this
 expires.
  hp3par_snapshot_retention=48

# Time in hours when a snapshot expires and is deleted. This must be larger
 than retention.
  hp3par_snapshot_expiration=72
```

Note

You can enable only one driver on each cinder instance unless you enable multiple backend support. See the Cinder multiple backend support instructions to enable this feature.

6. Save the changes to the `cinder.conf` file and restart the `cinder-volume` service.

The HP 3PAR Fibre Channel and iSCSI drivers should now be enabled on your OpenStack system. If you experience any problems, check the Block Storage log files for errors.

HP / LeftHand SAN

HP/LeftHand SANs are optimized for virtualized environments with VMware ESX & Microsoft Hyper-V, though the OpenStack integration provides additional support to various other virtualized environments (Xen, KVM, OpenVZ etc) by exposing the volumes via ISCSI to connect to the instances.

The HpSanISCSIDriver allows you to use a HP/Lefthand SAN that supports the Cliq interface. Every supported volume operation translates into a cliq call in the backend.

To use Cinder with HP/Lefthand SAN, you should set the following required parameters in `cinder.conf`:

- set `volume_driver=cinder.volume.san.HpSanISCSIDriver`.
- set `san_ip` flag to the hostname or VIP of your Virtual Storage Appliance (VSA).
- set `san_login` and `san_password` to the username and password of the ssh user with all necessary privileges on the appliance.
- set `san_ssh_port=16022` the default is set to 22, but the default for the VSA is usually 16022.
- set `san_clustername` to the name of the cluster on which the associated volumes will be created.

Some of the optional settings with their default values:

- `san_thin_provision=True` set it to False to disable thin provisioning.
- `san_is_local=False` This is almost always False for this driver. Setting it to True will try and run the cliq commands locally instead of over ssh.

Configuring the VSA

In addition to configuring the cinder-volume service some pre configuration has to happen on the VSA for proper functioning in an Openstack environment.

- Configure Chap on each of the nova-compute nodes.
- Add Server associations on the VSA with the associated Chap and initiator information. Note that the name should correspond to the *'hostname'* of the nova-compute node. For Xen this will be the hypervisor hostname. This can either be done through Cliq or the Centralized Management Console.

Huawei Storage Driver

Huawei storage drivers integrate OceanStor T series unified storage with OceanStor Dorado high-performance storage to provide block storage services for OpenStack.

Supported Operations

OceanStor T series unified storage supports the following operations:

- Create volume
- Delete volume
- Attach volume
- Detach volume
- Create snapshot
- Delete snapshot
- Create volume from snapshot

OceanStor Dorado5100 supports the following operations :

- Create volume
- Delete volume
- Attach volume
- Detach volume
- Create snapshot
- Delete snapshot

OceanStor Dorado2100 G2 supports the following operations :

- Create volume
- Delete volume
- Attach volume
- Detach volume

Configuring Cinder Nodes

In `/etc/cinder`, create the driver configuration file named `cinder_huawei_conf.xml`.

For OceanStor T series unified storage, the driver configuration file is shown as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<config>
    <Storage>
        <ControllerIP0>x.x.x.x</ControllerIP0>
        <ControllerIP1>x.x.x.x</ControllerIP1>
        <UserName>xxxxxxxx</UserName>
        <UserPassword>xxxxxxxx</UserPassword>
    </Storage>
    <LUN>
        <LUNType>Thick</LUNType>
        <StripUnitSize>64</StripUnitSize>
        <WriteType>1</WriteType>
        <MirrorSwitch>1</MirrorSwitch>
        <Prefetch Type="3" value="0"/>
        <StoragePool Name="xxxxxxxx"/>
        <StoragePool Name="xxxxxxxx"/>
    </LUN>
    <iSCSI>
        <DefaultTargetIP>x.x.x.x</DefaultTargetIP>
        <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
        <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
    </iSCSI>
</config>
```

For OceanStor Dorado5100, the driver configuration file is shown as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<config>
    <Storage>
        <ControllerIP0>x.x.x.x</ControllerIP0>
        <ControllerIP1>x.x.x.x</ControllerIP1>
        <UserName>xxxxxxxx</UserName>
        <UserPassword>xxxxxxxx</UserPassword>
    </Storage>
    <LUN>
        <StripUnitSize>64</StripUnitSize>
        <WriteType>1</WriteType>
        <MirrorSwitch>1</MirrorSwitch>
        <StoragePool Name="xxxxxxxx"/>
        <StoragePool Name="xxxxxxxx"/>
    </LUN>
    <iSCSI>
```

```
        <DefaultTargetIP>x.x.x.x</DefaultTargetIP>
        <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
        <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
    </iSCSI>
</config>
```

For OceanStor Dorado2100 G2, the driver configuration file is shown as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<config>
    <Storage>
        <ControllerIP0>x.x.x.x</ControllerIP0>
        <ControllerIP1>x.x.x.x</ControllerIP1>
        <UserName>xxxxxxxx</UserName>
        <UserPassword>xxxxxxxx</UserPassword>
    </Storage>
    <LUN>
        <LUNType>Thick</LUNType>
        <WriteType>1</WriteType>
        <MirrorSwitch>1</MirrorSwitch>
    </LUN>
    <iSCSI>
        <DefaultTargetIP>x.x.x.x</DefaultTargetIP>
        <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
        <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
    </iSCSI>
</config>
```

To add `volume-driver` and `cinder_huawei_conf_file` items, you can modify configuration file `cinder.conf` as follows:

```
volume_driver = cinder.volume.drivers.huawei.huawei_iscsi.HuaweiISCSIDriver
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf.xml
```

Configuration File Details

All flags of a configuration file are described as follows:

Table 3.2. List of configuration flags for Huawei Storage Driver

Flag name	Type	Default	Description
`ControllerIP0`	Mandatory		IP address of a primary controller.
`ControllerIP1`	Mandatory		IP address of a secondary controller.
`UserName`	Mandatory		Administrator user name.
`UserPassword`	Mandatory		Administrator password.
`LUNType`	Optional	Thick	Type of a created LUN. The value can be `Thick` or `Thin`.
`StripUnitSize`	Optional	64	Stripe depth of a created LUN. The value is expressed in KB. Note: This flag is invalid for a thin LUN.
`WriteType`	Optional	1	Cache write method. The method can be write back, write through, or mandatory write back. The default value is `1`, indicating write back.

Flag name	Type	Default	Description
`MirrorSwitch`	Optional	1	Cache mirroring policy. The default value is 1, indicating that a mirroring policy is used.
`Prefetch Type`	Optional	3	Cache prefetch strategy. The strategy can be constant prefetch, variable prefetch, or intelligent prefetch. The default value is 3, indicating intelligent prefetch.
`Prefetch Value`	Optional	0	Cache prefetch value.
`StoragePool`	Mandatory		Name of a storage pool that you want to use.
`DefaultTargetIP`	Mandatory		Default IP address of the iSCSI port provided for compute nodes.
`Initiator Name`	Optional		Name of a compute node initiator.
`Initiator TargetIP`	Optional		IP address of the iSCSI port provided for compute nodes.

Note

Multiple initiators can be configured in one configuration file, supporting multiple compute nodes. If a compute node initiator is not configured, the compute node connects the default target.

Multiple storage pools can be configured in one configuration file, supporting multiple storage pools in a storage system.

For more details, see command `createlun` in a specific command-line interface (CLI) reference or run **help -c createlun** in a storage system CLI.

After a driver is loaded, the storage system obtains any modification of the driver configuration file in real time and you do not need to restart the cinder-volume service.

IBM Storwize Family and SVC Volume Driver

The volume management driver for Storwize family and SAN Volume Controller (SVC) provides OpenStack Compute instances with access to IBM Storwize family or SVC storage systems.

Configuring the Storwize Family and SVC System

Network Configuration

The Storwize family or SVC system must be configured for iSCSI, Fibre Channel, or both.

If using iSCSI, each Storwize family or SVC node should have at least one iSCSI IP address. The IBM Storwize/SVC driver uses an iSCSI IP address associated with the volume's preferred node (if available) to attach the volume to the instance, otherwise it uses the first available iSCSI IP address of the system. The driver obtains the iSCSI IP address directly from the storage system; there is no need to provide these iSCSI IP addresses directly to the driver.

Note

If using iSCSI, ensure that the compute nodes have iSCSI network access to the Storwize family or SVC system.

Note

OpenStack Nova's Grizzly version supports iSCSI multipath. Once this is configured on the Nova host (outside the scope of this documentation), multipath will be enabled.

If using Fibre Channel (FC), each Storwize family or SVC node should have at least one WWPN port configured. If the `storwize_svc_multipath_enabled` flag is set to True in the Cinder configuration file, the driver will use all available WWPNs to attach the volume to the instance (details about the configuration flags appear in the next section). If the flag is not set, the driver uses the WWPN associated with the volume's preferred node (if available), otherwise it uses the first available WWPN of the system. The driver obtains the WWPNs directly from the storage system; there is no need to provide these WWPNs directly to the driver.

Note

If using FC, ensure that the compute nodes have FC connectivity to the Storwize family or SVC system.

iSCSI CHAP Authentication

If using iSCSI for data access, all new hosts created by the driver on the Storwize family or SVC system will have a randomly-generated CHAP secret associated with them. OpenStack compute nodes will use these secrets when creating iSCSI connections.

Note

CHAP secrets are not added to existing hosts.

Note

CHAP secrets are passed from Cinder to Nova in clear text. This communication should be secured to ensure that CHAP secrets are not discovered.

Configuring storage pools

The IBM Storwize/SVC driver allocates all volumes in a single pool. The pool should be created in advance and be provided to the driver using the `storwize_svc_volpool_name` configuration flag. Details about the configuration flags and how to provide the flags to the driver appear in the next section.

Configuring user authentication for the driver

The driver requires access to the Storwize family or SVC system management interface. The driver communicates with the management using SSH. The driver should be provided with the Storwize family or SVC management IP using the `san_ip` flag, and the management port should be provided by the `san_ssh_port` flag. By default, the port value is configured to be port 22 (SSH).

Note

Make sure the compute node running the nova-volume management driver has SSH network access to the storage system.

To allow the driver to communicate with the Storwize family or SVC system, you must provide the driver with a user on the storage system. The driver has two authentication methods: password-based authentication and SSH key pair authentication. The user should have an Administrator role. It is suggested to create a new user for the management driver. Please consult with your storage and security administrator regarding the preferred authentication method and how passwords or SSH keys should be stored in a secure manner.

Note

When creating a new user on the Storwize or SVC system, make sure the user belongs to the Administrator group or to another group that has an Administrator role.

If using password authentication, assign a password to the user on the Storwize or SVC system. The driver configuration flags for the user and password are `san_login` and `san_password`, respectively.

If you are using the SSH key pair authentication, create SSH private and public keys using the instructions below or by any other method. Associate the public key with the user by uploading the public key: select the "choose file" option in the Storwize family or SVC management GUI under "SSH public key". Alternatively, you may associate the SSH public key using the command line interface; details can be found in the Storwize and SVC documentation. The private key should be provided to the driver using the `san_private_key` configuration flag.

Creating a SSH key pair using OpenSSH

You can create an SSH key pair using OpenSSH, by running:

```
ssh-keygen -t rsa
```

The command prompts for a file to save the key pair. For example, if you select 'key' as the filename, two files will be created: `key` and `key.pub`. The `key` file holds the private SSH key and `key.pub` holds the public SSH key.

The command also prompts for a pass phrase, which should be empty.

The private key file should be provided to the driver using the `san_private_key` configuration flag. The public key should be uploaded to the Storwize family or SVC system using the storage management GUI or command line interface.

Note

Ensure that Cinder has read permissions on the private key file.

Configuring the Storwize Family and SVC Driver

Enabling the Storwize family and SVC driver

Set the volume driver to the Storwize family and SVC driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver = cinder.volume.drivers.storwize_svc.StorwizeSVCDriver
```

Configuring options for the Storwize family and SVC driver in cinder.conf

The following options specify default values for all volumes. Some can be over-ridden using volume types, which are described below.

Table 3.3. List of configuration flags for Storwize storage and SVC driver

Flag name	Type	Default	Description
`san_ip`	Required		Management IP or host name
`san_ssh_port`	Optional	22	Management port
`san_login`	Required		Management login username
`san_password`	Required [a]		Management login password
`san_private_key`	Required [a]		Management login SSH private key
`storwize_svc_volpool_name`	Required		Default pool name for volumes
`storwize_svc_vol_rsize`	Optional	2	Initial physical allocation (percentage) [b]
`storwize_svc_vol_warning`	Optional	0 (disabled)	Space allocation warning threshold (percentage) [b]
`storwize_svc_vol_autoexpand`	Optional	True	Enable or disable volume auto expand [c]
`storwize_svc_vol_grainsize`	Optional	256	Volume grain size [b] in KB
`storwize_svc_vol_compression`	Optional	False	Enable or disable Real-time Compression [d]
`storwize_svc_vol_easytier`	Optional	True	Enable or disable Easy Tier [e]
`storwize_svc_flashcopy_timeout`	Optional	120	FlashCopy timeout threshold [f] (seconds)
`storwize_svc_connection_protocol`	Optional	iSCSI	Connection protocol to use (currently supports 'iSCSI' or 'FC')
`storwize_svc_multipath_enabled`	Optional	False	Enable multipath for FC connections [g]

[a]The authentication requires either a password (`san_password`) or SSH private key (`san_private_key`). One must be specified. If both are specified the driver will use only the SSH private key.

[b] The driver creates thin-provisioned volumes by default. The `storwize_svc_vol_rsize` flag defines the initial physical allocation percentage for thin-provisioned volumes, or if set to `-1`, the driver creates full allocated volumes. More details about the available options are available in the Storwize family and SVC documentation.

[c] Defines whether thin-provisioned volumes can be auto expanded by the storage system, a value of `True` means that auto expansion is enabled, a value of `False` disables auto expansion. Details about this option can be found in the `–autoexpand` flag of the Storwize family and SVC command line interface `mkvdisk` command.

[d]Defines whether Real-time Compression is used for the volumes created with OpenStack. Details on Real-time Compression can be found in the Storwize family and SVC documentation. The Storwize or SVC system must have compression enabled for this feature to work.

[e] Defines whether Easy Tier is used for the volumes created with OpenStack. Details on EasyTier can be found in the Storwize family and SVC documentation. The Storwize or SVC system must have Easy Tier enabled for this feature to work.

[f] The driver wait timeout threshold when creating an OpenStack snapshot. This is actually the maximum amount of time the driver will wait for the Storwize family or SVC system to prepare a new FlashCopy mapping. The driver accepts a maximum wait time of 600 seconds (10 minutes).

[g] Multipath for iSCSI connections requires no storage-side configuration and is enabled if the compute host has multipath configured.

Working with multiple back-ends

Cinder has support for multiple back-ends. This can be accomplished by running multiple instances of cinder-volume, each with its own configuration file, or by including multiple sections in one configuration file. For example:

```
[DEFAULT]
...
enabled_backends = v7k1,v7k2

[v7k1]
...

[v7k2]
...
```

Here, common options are placed under `[DEFAULT]`, while options specific to a back-end are placed in the appropriate section.

By default, volumes will be allocated between back-ends to balance allocated space. However, `volume types` can be used to have finer-grained control over where volumes will be allocated. Each volume type contains a set of key-value pairs called `extra specs`. Volume types are typically set a priori by an administrator, and can be managed using the cinder client, using the `type-create`, `type-delete`, `type-key`, and `type-list` arguments.

The `extra specs` keys which have the "capabilities" prefix (called "scope") are interpreted by the Cinder scheduler and used to make placement decisions according to the capabilities of the available back-ends.

The following are examples of supported key-values:

In the following example, the volume will be placed on a controller named `myv7000`, in a pool named `openstack`:

```
capabilities:volume_backend_name=my7000_openstack
```

In the next example, the volume will be placed on a controller that supports compression. Specifying False will require the volume to be placed on a back-end that does not support compression (if no constraints on compression support are required, do not set this key):

```
capabilities:compression_support='<is> True'
```

The next example shows how Easy Tier support can be with set with the same semantics as `compression_support`:

```
capabilities:easytier_support='<is> True'
```

The connection protocol used to attach volumes to instances can also be specified using volume types. In this example, the volume will be placed on a controller that supports Fibre Channel, and Fibre Channel will be used when attaching that volume. FC can also be replaced with iSCSI.

```
capabilities:storage_protocol='<in> FC'
```

Configuring per-volume creation options

The volume types previously described can also be used to pass options to the IBM Storwize/SVC driver, which over-ride the default values set in the configuration file. Contrary to the previous examples where the "capabilities" scope was used to pass parameters to the Cinder scheduler, options can be passed to the IBM Storwize/SVC driver with the "drivers" scope, or by omitting the scope.

The following `extra specs` keys are supported by the IBM Storwize/SVC driver:

- rsize
- warning
- autoexpand
- grainsize
- compression
- easytier
- multipath

These keys have the same semantics as their counterparts in the configuration file. They are set similarly; for example, `rsize=2` or `compression=False`.

Complete volume type examples

In the following example, the volume will be placed on a controller named that supports Fibre Channel and compression, FC will be used when attaching the volume, and compression will be enabled:

```
cinder type-create compressed
cinder type-key compressed set capabilities:storage_protocol=''<in> FC'
 capabilities:compression_support='<is> True' drivers:compression=True
```

Volume types can be used, for example, to provide users with different

- performance levels (e.g., allocating entirely on an HDD tier, using Easy Tier for an HDD-SDD mix, or allocating entirely on an SSD tier)

- resiliency levels (e.g., allocating volumes in pools with different RAID levels)
- features (e.g., enabling/disabling Real-time Compression)

NetApp Drivers

NetApp drivers for 7-Mode and clustered Data ONTAP® have been written in two variants namely iSCSI and NFS drivers. Both variants provide OpenStack with access to NetApp 7-Mode controllers and clustered Data ONTAP for provisioning and managing OpenStack volumes.

NetApp iSCSI drivers for 7-Mode and clustered Data ONTAP

The NetApp iSCSI drivers for 7-Mode and clustered Data ONTAP systems provide OpenStack compute instances with access to NetApp 7-Mode storage controllers and clustered Data ONTAP storage systems.

NetApp iSCSI driver for 7-Mode storage controller

The NetApp iSCSI driver for 7-Mode is a driver interface from OpenStack to the NetApp 7-Mode storage controllers for provisioning and managing the SAN block storage entity, that is, NetApp LUN using iSCSI protocol.

The NetApp iSCSI driver for 7-Mode requires additional NetApp management software, namely OnCommand™, installed and configured for using 7-Mode storage controllers before configuring the 7-Mode driver on OpenStack.

Configuration options available for the 7-Mode system driver

Set the volume driver to the NetApp 7-Mode driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver=cinder.volume.drivers.netapp.iscsi.NetAppISCSIDriver
```

Table 3.4. List of configuration flags for NetApp 7-Mode iSCSI driver

Flag name	Type	Default	Description
`netapp_wsdl_url`	Mandatory		WSDL URL for NetApp OnCommand services running on an OnCommand installation. OnCommand is used as intermediate management software between OpenStack and 7-Mode storage systems.
`netapp_login`	Mandatory		Login user name for OnCommand installation.
`netapp_password`	Mandatory		Login password for OnCommand installation.
`netapp_server_hostname`	Mandatory		OnCommand server host name/IP address.
`netapp_server_port`	Optional	8088	OnCommand server port.
`netapp_storage_service`	Optional		Storage service to use while provisioning. Storage service is configured on OnCommand.

Flag name	Type	Default	Description
`netapp_storage_service_prefix`	Optional		Storage service prefix to use on OnCommand if using volume_types.
`netapp_vfiler`	Optional		The vFiler® unit name if using vFiler to host OpenStack volumes. MultiStore® must be enabled before using vFiler for provisioning.

Note

Make sure that at least one of the flags netapp_storage_service or netapp_storage_service_prefix is specified in configuration.

Refer to OpenStack NetApp community for detailed information.

NetApp iSCSI driver for clustered Data ONTAP

The NetApp iSCSI driver for clustered Data ONTAP is a driver interface from OpenStack to clustered Data ONTAP storage systems that allows the provisioning and managing the SAN block storage entity, that is, NetApp LUN using iSCSI protocol.

The NetApp iSCSI driver for clustered Data ONTAP requires additional NetApp management software namely OnCommand, WFA and the NetApp Cloud Web Service application to be installed and configured for using clustered Data ONTAP systems before configuring ONTAP cluster driver on OpenStack.

Configuration options for the clustered Data ONTAP driver

Set the volume driver to the clustered Data ONTAP driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver=cinder.volume.drivers.netapp.iscsi.NetAppCmodeISCSIDriver
```

Table 3.5. List of configuration flags for clustered Data ONTAP iSCSI driver

Flag name	Type	Default	Description
`netapp_wsdl_url`	Mandatory		WSDL URL for NetApp Cloud Web Service application serving as management software. NetApp Cloud Web Service is an intermediate service for propagating requests from the OpenStack driver to different NetApp software installed in the environment and the clustered Data ONTAP systems.
`netapp_server_hostname`	Mandatory		The host name/IP address of NetApp Cloud Web Service installation.
`netapp_server_port`	Mandatory		The port on which NetApp Cloud Web Service listens.
`netapp_login`	Mandatory		Login user name for NetApp Cloud Web Service installation.
`netapp_password`	Mandatory		Login password for NetApp Cloud Web Service installation.

Refer to OpenStack NetApp community for detailed information.

NetApp iSCSI direct driver for 7-Mode storage controller

The NetApp iSCSI direct driver for 7-Mode is a driver interface from OpenStack to the NetApp 7-Mode storage controllers for the provisioning and managing the SAN block storage entity, that is, NetApp LUN using iSCSI protocol.

The NetApp iSCSI direct driver for 7-Mode does not require any additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the 7-Mode storage controller.

Configuration options for the 7-Mode direct driver

Set the volume driver to the NetApp 7-Mode direct driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
  volume_driver=cinder.volume.drivers.netapp.iscsi.
NetAppDirect7modeISCSIDriver
```

Table 3.6. List of configuration flags for NetApp 7-Mode iSCSI direct driver

Flag name	Type	Default	Description
`netapp_server_hostname`	Mandatory		The management IP address for the 7-Mode storage controller.
`netapp_server_port`	Mandatory		The 7-Mode controller port to use for communication. As a custom 80 is used for HTTP and 443 is used for https communication. The default ports can be changed if other ports are used for ONTAPI® on 7-Mode controller.
`netapp_login`	Mandatory		Login user name for 7-Mode controller management.
`netapp_password`	Mandatory		Login password for 7-Mode controller management.
`netapp_transport_type`	Optional	http	Transport protocol to use for communicating with 7-Mode controller. Supported protocols include http and https.
`netapp_size_multiplier`	Optional	1.2	The quantity to be multiplied by the requested OpenStack volume size which then is used to make sure that the final size is available on the 7-Mode controller before creating the OpenStack volume on the same controller.
`netapp_vfiler`	Optional		The vFiler unit to be used for provisioning of OpenStack volumes. Use this only if using MultiStore®.
`netapp_volume_list`	Optional		Comma separated list of NetApp volumes to be used for provisioning on 7-Mode controller. This option is used to restrict provisioning to the specified NetApp controller volumes. In case this option is not specified all NetApp controller volumes except the controller root volume are used for provisioning OpenStack volumes.

Refer to OpenStack NetApp community for detailed information.

NetApp iSCSI direct driver for clustered Data ONTAP

The NetApp iSCSI direct driver for clustered Data ONTAP is a driver interface from OpenStack to clustered Data ONTAP storage systems for provisioning and managing the SAN block storage entity, that is, NetApp LUN using iSCSI protocol.

The NetApp iSCSI direct driver for clustered Data ONTAP does not require additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the clustered Data ONTAP.

Configuration options for the clustered Data ONTAP direct driver

Set the volume driver to the clustered Data ONTAP direct driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
  volume_driver=cinder.volume.drivers.netapp.iscsi.
NetAppDirectCmodeISCSIDriver
```

Table 3.7. List of configuration flags for clustered Data ONTAP iSCSI direct driver

Flag name	Type	Default	Description
`netapp_server_hostname`	Mandatory		The cluster management IP address for the clustered Data ONTAP.
`netapp_server_port`	Mandatory		The clustered Data ONTAP port to use for communication. As a custom 80 is used for http and 443 is used for https communication. The default ports can be changed if other ports are used for ONTAPI on clustered Data ONTAP.
`netapp_login`	Mandatory		Login user name for clustered Data ONTAP management.
`netapp_password`	Mandatory		Login password for clustered Data ONTAP management.
`netapp_transport_type`	Optional	http	Transport protocol for communicating with clustered Data ONTAP. Supported protocols include http and https.
`netapp_size_multiplier`	Optional	1.2	The quantity to be multiplied to the requested OpenStack volume size which then is used to make sure that the final size is available on clustered Data ONTAP Vserver before creating OpenStack volume on the same.
`netapp_vserver`	Optional	openstack	The Vserver on the cluster on which provisioning of OpenStack volumes will be done.

Refer to OpenStack NetApp community for detailed information.

NetApp NFS drivers for 7-Mode and clustered Data ONTAP systems

The NetApp NFS drivers for 7-Mode and clustered Data ONTAP systems provide OpenStack compute instances with access to NetApp 7-Mode storage controllers and clustered Data

ONTAP storage systems for provisioning and managing entities on NFS exports on NetApp storage controllers.

The NFS exports are mounted on the OpenStack nodes after which the OpenStack volumes can be created and managed using the NetApp NFS drivers on the NFS exports. The OpenStack compute instances get the required block storage device as files on NFS exports managed by OpenStack.

NetApp NFS driver for 7-Mode storage controller

The NetApp NFS driver for 7-Mode is a driver interface from OpenStack to NetApp 7-Mode storage controllers for provisioning and managing OpenStack volumes on NFS exports provided by the 7-Mode storage controller.

The NetApp NFS driver for 7-Mode requires additional NetApp management software namely OnCommand which needs installed and configured for using 7-Mode storage controllers before configuring 7-Mode NFS driver on OpenStack.

Configuration options available for the 7-Mode NFS driver

Set the volume driver to the NetApp 7-Mode driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver=cinder.volume.drivers.netapp.nfs.NetAppNFSDriver
```

Table 3.8. List of configuration flags for NetApp 7-Mode NFS driver

Flag name	Type	Default	Description
`netapp_wsdl_url`	Mandatory		WSDL URL for NetApp OnCommand Webservices running on an OnCommand installation. OnCommand is used as an intermediate management software between OpenStack and 7-Mode storage systems.
`netapp_server_hostname`	Mandatory		OnCommand server host name/IP address.
`netapp_server_port`	Optional	8088	OnCommand server port to connect to.
`netapp_login`	Mandatory		Login user name for OnCommand installation.
`netapp_password`	Mandatory		Login password for OnCommand installation.

Refer to OpenStack NetApp community for detailed information.

NetApp NFS driver for clustered Data ONTAP

The NetApp NFS driver for clustered Data ONTAP is a driver interface from OpenStack to clustered Data ONTAP systems for provisioning and managing OpenStack volumes on NFS exports provided by the clustered Data ONTAP system.

The NetApp NFS driver for clustered Data ONTAP requires additional NetApp management software namely OnCommand, WFA and NetApp Cloud Web Service application to be installed and configured for using clustered Data ONTAP systems before configuring clustered Data ONTAP NFS driver on OpenStack.

Configuration options for the clustered Data ONTAP NFS driver

Set the volume driver to the clustered Data ONTAP NFS driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver=cinder.volume.drivers.netapp.nfs.NetAppCmodeNfsDriver
```

Table 3.9. List of configuration flags for clustered Data ONTAP NFS driver

Flag name	Type	Default	Description
`netapp_wsdl_url`	Mandatory		WSDL URL for NetApp Cloud Web Service serving as management software. NetApp Cloud Web Service is an intermediate service used for propagating requests from the OpenStack driver to different NetApp software installed in the environment and the ONTAP cluster systems.
`netapp_server_hostname`	Mandatory		The host name/IP address of NetApp Cloud Web Service installation.
`netapp_server_port`	Mandatory		The port on which NetApp Cloud Web Service listens.
`netapp_login`	Mandatory		Login user name for NetApp Cloud Web Service installation.
`netapp_password`	Mandatory		Login password for NetApp Cloud Web Service installation.

Refer to OpenStack NetApp community for detailed information.

NetApp NFS direct driver for 7-Mode storage controller

The NetApp NFS direct driver for 7-Mode is a driver interface from OpenStack to NetApp 7-Mode storage controllers for provisioning and managing OpenStack volumes on NFS exports provided by the 7-Mode storage controller.

The NetApp NFS direct driver for 7-Mode does not require any additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the 7-Mode storage controller.

Configuration options for the 7-Mode NFS direct driver

Set the volume driver to the NetApp 7-Mode direct driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver=cinder.volume.drivers.netapp.nfs.NetAppDirect7modeNfsDriver
```

Table 3.10. List of configuration flags for NetApp 7-Mode NFS direct driver

Flag name	Type	Default	Description
`netapp_server_hostname`	Mandatory		The management IP address for the NetApp 7-Mode storage controller.

Flag name	Type	Default	Description
`netapp_server_port`	Mandatory		The 7-Mode controller port to use for communication. As a custom 80 is used for http and 443 is used for https communication. The default ports can be changed if other ports are for ONTAPI on 7-Mode controller.
`netapp_login`	Mandatory		Login user name for 7-Mode controller management.
`netapp_password`	Mandatory		Login password for 7-Mode controller management.
`netapp_transport_type`	Optional	http	Transport protocol to use for communicating with 7-Mode controller. Supported protocols include http and https.

Refer to OpenStack NetApp community for detailed information.

NetApp NFS direct driver for clustered Data ONTAP

The NetApp NFS direct driver for clustered Data ONTAP is a driver interface from OpenStack to clustered Data ONTAP system for provisioning and managing OpenStack volumes on NFS exports provided by the clustered Data ONTAP system.

The NetApp NFS direct driver for clustered Data ONTAP does not require any additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the clustered Data ONTAP.

Configuration options for the clustered Data ONTAP NFS direct driver

Set the volume driver to the NetApp clustered Data ONTAP direct driver by setting the `volume_driver` option in `cinder.conf` as follows:

```
volume_driver=cinder.volume.drivers.netapp.nfs.NetAppDirectCmodeNfsDriver
```

Table 3.11. List of configuration flags for NetApp clustered Data ONTAP NFS direct driver

Flag name	Type	Default	Description
`netapp_server_hostname`	Mandatory		The cluster management IP address for the clustered Data ONTAP.
`netapp_server_port`	Mandatory		The clustered Data ONTAP port for communication. As a custom 80 is used for http and 443 is used for https communication. The default ports can be changed if other ports are used for ONTAPI on clustered Data ONTAP.
`netapp_login`	Mandatory		Login user name for clustered Data ONTAP management.
`netapp_password`	Mandatory		Login password for clustered Data ONTAP management.
`netapp_transport_type`	Optional	http	Transport protocol for communicating with clustered Data ONTAP. Supported protocols include http and https.

Refer to OpenStack NetApp community for detailed information.

Nexenta

NexentaStor Appliance is NAS/SAN software platform designed for building reliable and fast network storage arrays. The the OpenSolaris and uses ZFS as a disk management system. NexentaStor can serve as a storage node for the OpenStack and for the virtual servers via iSCSI protocol.

The Nexenta driver allows you to use Nexenta SA to store Nova volumes. Every Nova volume is represented by a single zvol in a predefined Nexenta volume. For every new volume the driver creates a iSCSI target and iSCSI target group that are used to access it from compute hosts.

To use Nova with Nexenta Storage Appliance, you should:

- set `volume_driver=nova.volume.nexenta.volume.NexentaDriver`.
- set `--nexenta_host` flag to the hostname or IP of your NexentaStor
- set `--nexenta_user` and `--nexenta_password` to the username and password of the user with all necessary privileges on the appliance, including the access to REST API
- set `--nexenta_volume` to the name of the volume on the appliance that you would like to use in Nova, or create a volume named `nova` (it will be used by default)

Nexenta driver has a lot of tunable flags. Some of them you might want to change:

- `nexenta_target_prefix` defines the prefix that will be prepended to volume id to form target name on Nexenta
- `nexenta_target_group_prefix` defines the prefix for target groups
- `nexenta_blocksize` can be set to the size of the blocks in newly created zvols on appliance, with the suffix; for example, the default 8K means 8 kilobytes
- `nexenta_sparse` is boolean and can be set to use sparse zvols to save space on appliance

Some flags that you might want to keep with the default values:

- `nexenta_rest_port` is the port where Nexenta listens for REST requests (the same port where the NMV works)
- `nexenta_rest_protocol` can be set to `http` or `https`, but the default is `auto` which makes the driver try to use HTTP and switch to HTTPS in case of failure
- `nexenta_iscsi_target_portal_port` is the port to connect to Nexenta over iSCSI

NFS Driver

NFS, the Network Filesystem, is a remote file system protocol dating back to the 80's. An NFS server *exports* one or more of its own file systems (known as *shares*). An NFS client can then mount these exported shares onto its own file system. Normal file actions can then be performed on this mounted remote file system as if the file system was local.

How the NFS Driver Works

The NFS driver, and other drivers based off of it, work quite a bit differently than a traditional block storage driver.

The NFS driver doesn't actually allow an instance to have access to a storage device at the block-level. Instead, files are created on an NFS share and then mapped to instances, emulated as a block device. This works very similarly to how QEMU stores instances under `/var/lib/nova/instances`.

Enabling the NFS Driver and Related Options

To use Cinder with the NFS driver, first set the `volume_driver` in `cinder.conf`:

```
volume_driver=cinder.volume.drivers.nfs.NfsDriver
```

The following table contains the options supported by the NFS driver.

Table 3.12. List of configuration flags for NFS

Flag Name	Type	Default	Description
`nfs_shares_config`	Mandatory		(StrOpt) File with the list of available NFS shares.
`nfs_mount_point_base`	Optional	`$state_path/mnt`	(StrOpt) Base dir where nfs shares are expected to be mounted.
`nfs_disk_util`	Optional	`df`	(StrOpt) Use du or `df` for free space calculation.
`nfs_sparsed_volumes`	Optional	True	(BoolOpt) Create volumes as sparsed files which take no space. If set to False, volume is created as regular file. In such case volume creation takes a lot of time.
`nfs_mount_options`	Optional	None	(StrOpt) Mount options passed to the nfs client. See section of the nfs man page for details.

How to Use the NFS Driver

First, you need access to one or more NFS servers. Creating an NFS server is outside the scope of this document. For example purposes, access to the following NFS servers and mountpoints will be assumed:

- `192.168.1.200:/storage`
- `192.168.1.201:/storage`
- `192.168.1.202:/storage`

Three different NFS servers are used in this example to highlight that you can utilize more than one NFS server with this driver. Multiple servers are not required. One will probably be enough in most cases.

Add your list of NFS servers to the file you specified with the `nfs_shares_config` option. For example, if the value of this option was set to `/etc/cinder/shares.txt`, then:

```
# cat /etc/cinder/shares.txt
192.168.1.200:/storage
192.168.1.201:/storage
192.168.1.202:/storage
```

Comments are allowed in this file. They begin with a #.

The next step is to configure the `nfs_mount_point_base` option. This is a directory where `cinder-volume` will mount all NFS shares stored in `shares.txt`. For this example, `/var/lib/cinder/nfs` will be used. You can, of course, use the default value of `$state_path/mnt`.

Once these options are set, start the `cinder-volume` service. `/var/lib/cinder/nfs` should now contain a directory for each NFS share specified in `shares.txt`. The name of each directory will be a hashed name:

```
# ls /var/lib/cinder/nfs
...
46c5db75dc3a3a50a10bfd1a456a9f3f
...
```

You can now create volumes as you normally would:

```
# nova volume-create --display-name=myvol 5
# ls /var/lib/cinder/nfs/46c5db75dc3a3a50a10bfd1a456a9f3f
volume-a8862558-e6d6-4648-b5df-bb84f31c8935
```

This volume can also be attached and deleted just like other volumes. However, snapshotting is *not* supported.

NFS Driver Notes

- `cinder-volume` manages the mounting of the NFS shares as well as volume creation on the shares. Keep this in mind when planning your OpenStack architecture. If you have one master NFS server, it might make sense to only have one `cinder-volume` service to handle all requests to that NFS server. However, if that single server is unable to handle all requests, more than one `cinder-volume` service will be needed as well as potentially more than one NFS server.
- Since data is stored in a file and not actually on a block storage device, you might not see the same IO performance as you would with a traditional block storage driver. Please test accordingly.
- Despite possible IO performance loss, having volume data stored in a file might be beneficial. For example, backing up volumes can be as easy as copying the volume files (note: regular IO flushing and syncing still stands).

SolidFire

The SolidFire Cluster is a high performance all SSD iSCSI storage device, providing massive scale out capability and extreme fault tolerance. A key feature of the SolidFire cluster is the ability to set and modify during operation specific QoS levels on a volume per volume basis. The SolidFire cluster offers all of these things along with de-duplication, compression and an architecture that takes full advantage of SSD's.

To configure and use a SolidFire cluster with Cinder modify your `cinder.conf` file as shown below:

```
volume_driver=cinder.volume.drivers.solidfire.SolidFire
san_ip=172.17.1.182          # the address of your MVIP
san_login=sfadmin            # your cluster admin login
san_password=sfpassword      # your cluster admin password
```

XenAPINFS

XenAPINFS is a Block Storage (Cinder) driver which is using an NFS share through XenAPI's Storage Manager to store virtual disk images and exposing those virtual disks as volumes.

This driver is not accessing the NFS share directly, it is only accessing the share through XenAPI Storage Manager. This driver should be considered as a reference implementation for using XenAPI's storage manager in OpenStack (present in XenServer and XCP).

Requirements

- A XenServer/XCP installation acting as Storage Controller. I will refer to this hypervisor as Storage Controller in this document.
- Use XenServer/XCP as your hypervisor for compute nodes.
- An NFS share, that is configured for XenServer/XCP. For the specific requirements, export options, please refer to the administration guide of your specific XenServer version. It is also requirement, that the NFS share is accessible by all the XenServers components within your cloud.
- For creating volumes from XenServer type images (vhd tgz files), XenServer Nova plugins are also required on the Storage Controller.

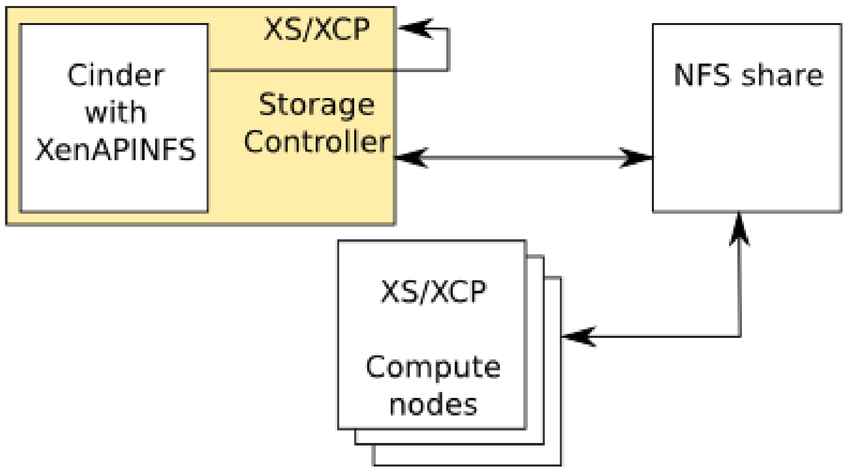
Note

It is possible to use a XenServer as a Storage Controller and as a compute node in the same time, thus the minimal configuration consists of a XenServer/XCP box and an NFS share.

Configuration Patterns

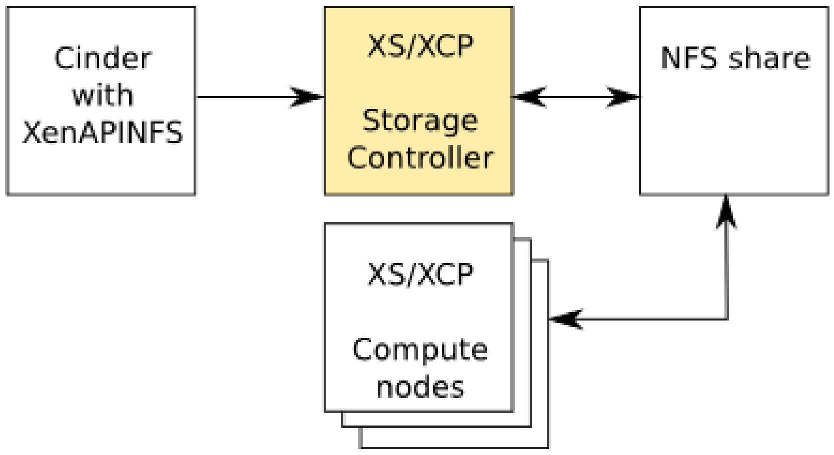
- Local configuration (Recommended): The driver is running in a virtual machine on top of the storage controller. With this configuration, it is possible to create volumes from other formats supported by `qemu-img`.

Figure 3.2. Local configuration



- Remote configuration: The driver is not a guest VM of the storage controller. With this configuration, you can only use XenServer vhd-type images to create volumes.

Figure 3.3. Remote configuration



Configuration Options

Assuming the following setup:

- XenServer box at `10.2.2.1`
- XenServer password is `r00tme`
- NFS server is `nfs.example.com`
- NFS export is at `/volumes`

In order to use XenAPINFS as your cinder driver, the following configuration options needs to be set in `cinder.conf`:

```
volume_driver = cinder.volume.drivers.xenapi.sm.XenAPINFSDriver
xenapi_connection_url = http://10.2.2.1
xenapi_connection_username = root
xenapi_connection_password = r00tme
xenapi_nfs_server = nfs.example.com
xenapi_nfs_serverpath = /volumes
```

Using the XenAPI Storage Manager Volume Driver

The Xen Storage Manager Volume driver (xensm) is a XenAPI hypervisor specific volume driver, and can be used to provide basic storage functionality, including volume creation and destruction, on a number of different storage back-ends. It also enables the capability of using more sophisticated storage back-ends for operations like cloning/snapshots, etc. The list below shows some of the storage plugins already supported in Citrix XenServer and Xen Cloud Platform (XCP):

1. NFS VHD: Storage repository (SR) plugin which stores disks as Virtual Hard Disk (VHD) files on a remote Network File System (NFS).
2. Local VHD on LVM: SR plugin which represents disks as VHD disks on Logical Volumes (LVM) within a locally-attached Volume Group.
3. HBA LUN-per-VDI driver: SR plugin which represents Logical Units (LUs) as Virtual Disk Images (VDIs) sourced by host bus adapters (HBAs). E.g. hardware-based iSCSI or FC support.
4. NetApp: SR driver for mapping of LUNs to VDIs on a NETAPP server, providing use of fast snapshot and clone features on the filer.
5. LVHD over FC: SR plugin which represents disks as VHDs on Logical Volumes within a Volume Group created on an HBA LUN. E.g. hardware-based iSCSI or FC support.
6. iSCSI: Base ISCSI SR driver, provides a LUN-per-VDI. Does not support creation of VDIs but accesses existing LUNs on a target.
7. LVHD over iSCSI: SR plugin which represents disks as Logical Volumes within a Volume Group created on an iSCSI LUN.
8. EqualLogic: SR driver for mapping of LUNs to VDIs on a EQUALLOGIC array group, providing use of fast snapshot and clone features on the array.

Design and Operation

Definitions

- **Backend:** A term for a particular storage backend. This could be iSCSI, NFS, Netapp etc.
- **Backend-config:** All the parameters required to connect to a specific backend. For e.g. For NFS, this would be the server, path, etc.

- **Flavor:** This term is equivalent to volume "types". A user friendly term to specify some notion of quality of service. For example, "gold" might mean that the volumes will use a backend where backups are possible. A flavor can be associated with multiple backends. The volume scheduler, with the help of the driver, will decide which backend will be used to create a volume of a particular flavor. Currently, the driver uses a simple "first-fit" policy, where the first backend that can successfully create this volume is the one that is used.

Operation

The admin uses the nova-manage command detailed below to add flavors and backends.

One or more cinder-volume service instances will be deployed per availability zone. When an instance is started, it will create storage repositories (SRs) to connect to the backends available within that zone. All cinder-volume instances within a zone can see all the available backends. These instances are completely symmetric and hence should be able to service any `create_volume` request within the zone.

On XenServer, PV guests required

Note that when using XenServer you can only attach a volume to a PV guest.

Configuring XenAPI Storage Manager

Prerequisites

1. xensm requires that you use either Citrix XenServer or XCP as the hypervisor. The NetApp and EqualLogic backends are not supported on XCP.

2. Ensure all **hosts** running volume and compute services have connectivity to the storage system.

Configuration

- **Set the following configuration options for the nova volume service: (nova-compute also requires the volume_driver configuration option.)**

```
--volume_driver="nova.volume.xensm.XenSMDriver"
--use_local_volumes=False
```

- **The backend configurations that the volume driver uses need to be created before starting the volume service.**

```
$ nova-manage sm flavor_create <label> <description>

$ nova-manage sm flavor_delete <label>

$ nova-manage sm backend_add <flavor label> <SR type> [config connection
 parameters]

Note: SR type and config connection parameters are in keeping with the
 XenAPI Command Line Interface. http://support.citrix.com/article/CTX124887

$ nova-manage sm backend_delete <backend-id>
```

Example: For the NFS storage manager plugin, the steps below may be used.

```
$ nova-manage sm flavor_create gold "Not all that glitters"

$ nova-manage sm flavor_delete gold

$ nova-manage sm backend_add gold nfs name_label=mybackend server=myserver
 serverpath=/local/scratch/myname

$ nova-manage sm backend_remove 1
```

- **Start cinder-volume and nova-compute with the new configuration options.**

Creating and Accessing the volumes from VMs

Currently, the flavors have not been tied to the volume types API. As a result, we simply end up creating volumes in a "first fit" order on the given backends.

The standard euca-* or OpenStack API commands (such as volume extensions) should be used for creating, destroying, attaching, or detaching volumes.