IMPLEMENTING A LIGHTWEIGHT OPENSTACK JUNO/CINDER HOST AS A VIPR CONTROLLER STORAGE PROVIDER

ABSTRACT

This white paper explains how to build a lightweight VM or physical host as a ViPR Controller 2.2 third-party storage provider using Ubuntu LTS 14.04. The approach illustrated is intended for evaluation and test purposes but similar techniques could be used for production deployments.

March, 2015

To learn more about how EMC products, services, and solutions can help solve your business and IT challenges, contact your local representative or authorized reseller, visit www.emc.com, or explore and compare products in the EMC Store

Part Number H13996

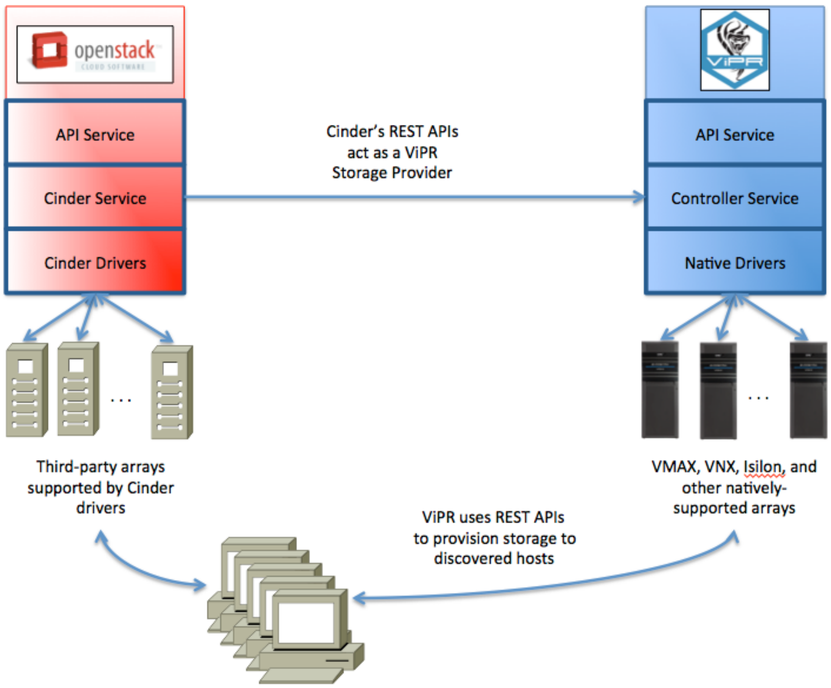
TABLE OF CONTENTS

EXECUTIVE SUMMARY

The explosive growth of OpenStack for instance hosting raises a need for flexible and open storage platform management, EMC's ViPR® Controller product abstracts the storage control path from the underlying hardware arrays so that access and management of multi-vendor storage infrastructures can be centrally executed in software. Using ViPR Controller and OpenStack together, users can create "single-pane-of-glass" management portals from both the storage and instance viewpoints, easily providing the right resource management tool for either group.

This document describes how to build a ViPR Controller storage provider based on the freely available Ubuntu LTS 14.04 release. Whether built as a physical host or as a virtual machine, the technique allows ViPR Controller to use almost any third-party storage supported by the OpenStack Juno release. The configuration looks like this:

Figure 1. ViPR CONTROLLER STORAGE PROVIDER BASED ON UBUNTU LTS 14.04



Although this implementation is suitable for demonstration and test purposes, EMC strongly advises a thorough review of the ViPR Controller and OpenStack documentation and appropriate validation to determine suitability before deploying in any production environment.

WHY OPENSTACK JUNO?

The OpenStack cloud software stack contains modular components to handle compute (Nova), object storage (Swift), block storage (Cinder), and networking (Neutron), among other functions. Cinder provides a consistent, open layer for persistent block storage independent of any vendor API requirements. Block storage volumes exposed via Cinder fully integrate into the ViPR Controller services via a consistent interface, allowing cloud users to manage device creation, snapshot, and other storage functions while hiding vendor-specific implementation and control details. Juno is the current release of OpenStack as of this writing.

WHY VIPR CONTROLLER?

ViPR Controller provides separation of control and data planes in storage management, allowing tiering, provisioning, pooling, and other functions across multiple-vendor physical storage array installations. Through the use of REST APIs, different front-end consoles (such as OpenStack) can present a unified interface to ViPR Controller control functions, thus consuming storage from ViPR Controller in a clean, vendor-neutral fashion. In addition, ViPR Controller can act as a front-end to storage presented from OpenStack, allowing control and use of Cinder volumes created from arrays that ViPR Controller may not natively support.

IS THERE AN EASIER WAY TO DO THIS?

This paper demonstrates a bare-metal approach to building an OpenStack provider for use with ViPR Controller. If you're interested in a turnkey download, look at https://community.emc.com/docs/DOC-37248, which describes a pre-built VMware virtual machine specifically designed to provide Cinder and Keystone services to ViPR Controller. There's also information about certain third-party arrays and other useful community-provided material at that link.

AUDIENCE

This white paper is intended for system architects, administrators, and implementors who want to use Cinder as a mechanism to interface third-party storage arrays to their ViPR Controller installation.

HOST REQUIREMENTS

- Cinder host: Minimum requirements include an x86_64 processor with at least two cores, at least 2 GB of RAM, at least 8 GB of disk space and at least one NIC, as well as connection to an array. A virtual machine meeting these specifications will work (and was used for this document).
- A VMware environment containing a ViPR Controller 2.x instance
- At least one "target" host (Windows or Linux) to consume storage from the ViPR Controller instance.
- A local-area network connecting all of the above components.
- DNS, NTP, and Internet connectivity to download the components.

This document was developed using:

- OpenStack host: Ubuntu 14.04 LTS x86_64 with attachment to a NetApp 8.2 7-mode iSCSI block storage provider
- ViPR Controller host: ViPR Controller 2.2.0.0 (build 758) hosted on VMware ESXi 5.5.0
- Target host: CentOS 6.5 with iscsi-initiator-utils installed

NOTES

- In this document, commands performed at the shell prompt while logged in as root are prefixed by "#", and those performed within the database engine are prefixed by ">".
- If you're copying-and-pasting from the text, watch for space and dash conversions, and note that lines ending in " \" are continuations. You may want to paste your text into an editor, check the conversions and join continuation lines, (eliminating the \ character) and then paste the result to your command line. Most OpenStack command arguments start with a double dash ("--").

CREATE THE HOST ENVIRONMENT

INSTALL A BASIC LINUX HOST

To build the Cinder host, download Ubuntu 14.04 LTS from http://releases.ubuntu.com/14.04.1/ubuntu-14.04.1-server-amd64.iso), boot the host or VM on that image, and perform a normal OS installation. Take the default settings unless local requirements dictate otherwise. Select the "basic" and "OpenSSH" server options; the GUI's not required. Set GRUB as the master boot loader and let the host reboot at the end of the installation.

Log into the newly installed host at the console, su to root, and open /etc/network/interfaces in an editor. OpenStack likes static addressing for its nodes, so set the configuration to something similar to the following, inserting appropriate values for your network:

```
# The primary network interface
auto eth0
iface eth0 inet static
address xxx.xxx.xxx.xxx
netmask xxx.xxx.xxx.xxx
```

```
gateway xxx.xxx.xxx.xxx
dns-nameservers xxx.xxx.xxx.xxx xxx.xxx.xxx.xxx
```

Note the name server list is space, not comma, separated; this differs from Red Hat Linux variants. If name resolution doesn't work after you reboot, make sure you didn't add a comma between your nameserver addresses.

ViPR Controller needs a root login via SSH to work properly on Ubuntu. Edit /etc/ssh/sshd_config, changing:

```
PermitRootLogin without-password
StrictModes yes
```

to:

```
PermitRootLogin yes
StrictModes no
```

While still in the root shell, set up a password for the root's login (using "passwd root"), Next, edit /etc/hosts. Remove the address line starting with "127.0.0.1" and create an entry using an explicit IPv4 address, adding the aliases "localhost" and "controller". The file should contain the following:

```
xxx.xxx.xxx.xxx  <host>  <host.domain> controller  localhost
127.0.0.1      loopback
```

Reboot to pick up the network configuration. Once the host is back up, you should be able to ssh in as root.

UPDATE THE HOST AND SET UP BASIC SERVICES

After logging in as root, verify your static network settings and DNS, then pull down the updates from the Ubuntu repository:

```
# apt-get -y update
```

In order to get the OpenStack Juno code (which provides the Cinder storage management functionality), configure apt to use the Juno repository. Repeat the update to pick up any Juno-specific info, and then upgrade the distribution to the latest code:

```
# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu" "trusty-updates/juno main" >
/etc/apt/sources.list.d/cloudarchive-juno.list
# apt-get install ubuntu-cloud-keyring
# apt-get -y update
# apt-get -y dist-upgrade
```

Set up NTP for time synchronization and if needed, install the iSCSI initiator (the example installation uses an iSCSI NetApp as a back end):

```
# apt-get -y install ntp
# apt-get -y install open-iscsi
```

Edit ntp.conf to install your local time servers, then reboot (as the kernel may have updated).

INSTALL AND CONFIGURE CINDER COMPONENTS

DEFINE PASSWORDS

Create a table similar to the one below containing the passwords for your installation. Table 1 provides the variable names (matching those in the OpenStack documentation) and the passwords reflect the demonstration environment:

Table 1 VARIABLE NAMES, PASSWORDS AND DESCRIPTIONS

Variable name	Password	Description
<database>	dbpass	Root password for the database
RABBIT_PASS	rbpass	Password of user guest of RabbitMQ
KEYSTONE_DBPASS	kypass	Database password of Identity service
ADMIN_PASS	adpass	Password of user admin
CINDER_DBPASS	cdpass	Database password for the Block Storage service
CINDER_PASS	cdpass	Password of Block Storage service user cinder

INSTALL DATABASE AND MESSAGE QUEUE SERVICES

After logging in as root, you next install the Python configuration scripts for the APT distribution system:

```
# apt-get -y install python-software-properties
```

Next, add the link for the Juno repositories into APT:

```
# add-apt-repository cloud-archive:juno
```

OpenStack relies heavily on a database to store and manage its information. Install mysql using the <database> password from the chart above:

```
# apt-get -y install python-mysqldb mariadb-server
```

To configure the SQL engine, edit /etc/mysql/my.cnf, and under [mysqld] add the following:

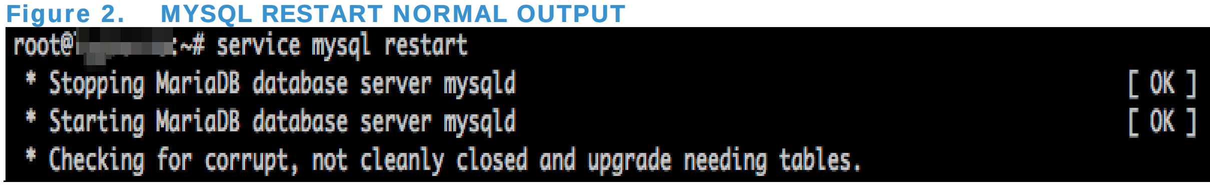
```
default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

Start the database:

```
# service mysql restart
```

Figure 2 provides an example of a normal output.

Figure 2. MYSQL RESTART NORMAL OUTPUT



Install the Rabbit message queue service:

```
# apt-get -y install rabbitmq-server
```

Rabbit's installer starts the service automatically. Change the guest password to the RABBIT_PASS password in your table:

```
# rabbitmqctl change_password guest rbpass
# service rabbitmq-server restart
```

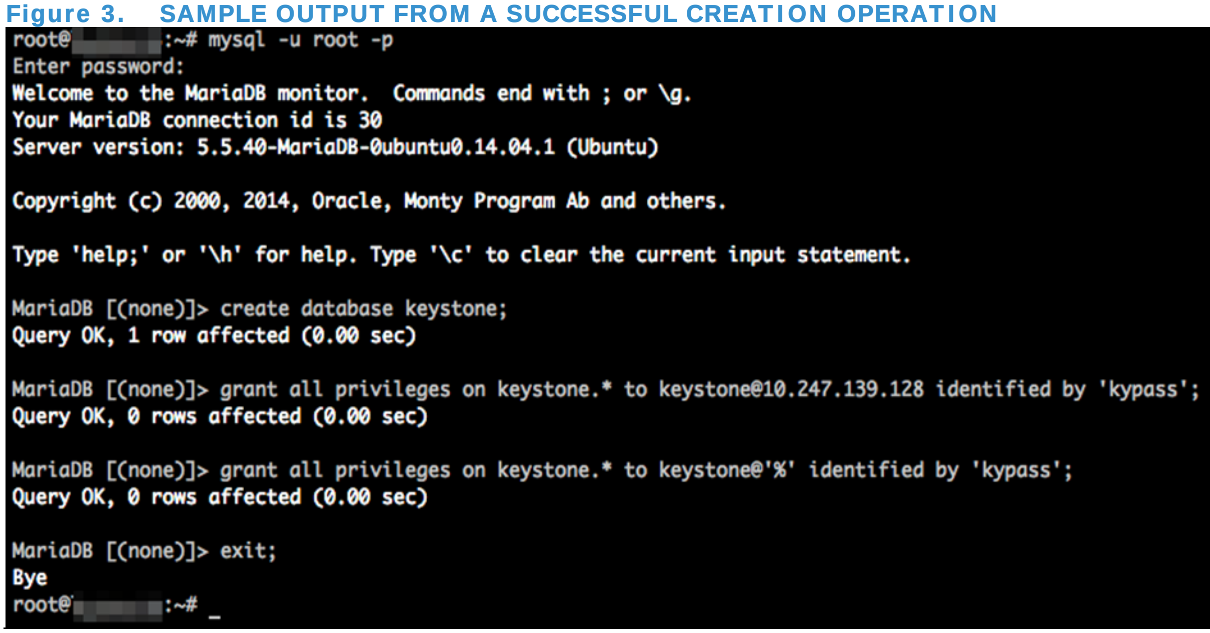
INSTALL AND CONFIGURE AUTHENTICATION SERVICE

While logged in as root, build the database structures for the Keystone authentication service using the following commands (note that 'kypass' was our Keystone password from the table). Don't forget to end each database command with a semicolon, and substitute your Cinder host's IP address for <host_ip>:

```
# mysql -u root -p
> create database keystone;
> grant all privileges on keystone.* to keystone@<host_ip> identified by 'kypass';
> grant all privileges on keystone.* to keystone@'%' identified by 'kypass';
> exit;
```

Figure 3 provides a sample output from a successful creation operation.

Figure 3. SAMPLE OUTPUT FROM A SUCCESSFUL CREATION OPERATION



```
root@[redacted]:~# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 30
Server version: 5.5.40-MariaDB-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle, Monty Program Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database keystone;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> grant all privileges on keystone.* to keystone@10.247.139.128 identified by 'kypass';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> grant all privileges on keystone.* to keystone@'%' identified by 'kypass';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> exit;
Bye
root@[redacted]:~# _
```

Next, install the Python client for Keystone:

```
# apt-get -y install keystone python-keystoneclient
```

Now that the database is installed, you can configure Keystone. Start by generating a random token value for use in initial setup, copy the string to your clipboard, and export it along with your host's IP address information:

```
# openssl rand –hex
<returns token value>
# export OS_SERVICE_TOKEN=<token value>
# export OS_SERVICE_ENDPOINT=http://<host_ip>:35357/v2.0
```

Verify by checking the environment, which should look similar to Figure 4.

Figure 4. VERIFY BY CHECKING THE ENVIRONMENT

Open /etc/keystone/keystone.conf in an editor, and make the following changes:

- Under [DEFAULT], uncomment "admin_token" and replace with the random string you generated.
- Under [database], change database connection string to:

```
connection = mysql://keystone:kypass@127.0.0.1/keystone
```

(Replace “kypass” with your Keystone password, but leave the rest alone; this CANNOT be "controller", an alias, or the host's IP; just use the loopback address.)

- Under [token], uncomment:

```
driver=keystone.token.backends.sql.Token
```

Populate the initial database schema as follows (note that the first command produces no output):

```
# keystone-manage db_sync
# service keystone restart
```

Keystone tends to keep expired tokens around which can eventually fill up your database and disk. Run the following command to create a cron job that will purge them out hourly:

```
# (crontab -l -u keystone 2>&1 | grep -q token_flush) || echo '@hourly /usr/bin/keystone-manage
token_flush > /var/log/keystone/keystone-tokenflush.log 2>&1' >> /var/spool/cron/crontabs/keystone
```

Create the admin, cinder, & service users / tenants, and then build the Keystone endpoint. This is easiest from a script file, so create build_keystone.sh containing the following, substituting your admin password from the table (and e-mail if desired). Everything else is literal and stays as-is:

```
keystone tenant-create --name admin --description "Admin Tenant"
keystone user-create --name admin --pass adpass --email null@void.com
keystone role-create --name admin
```

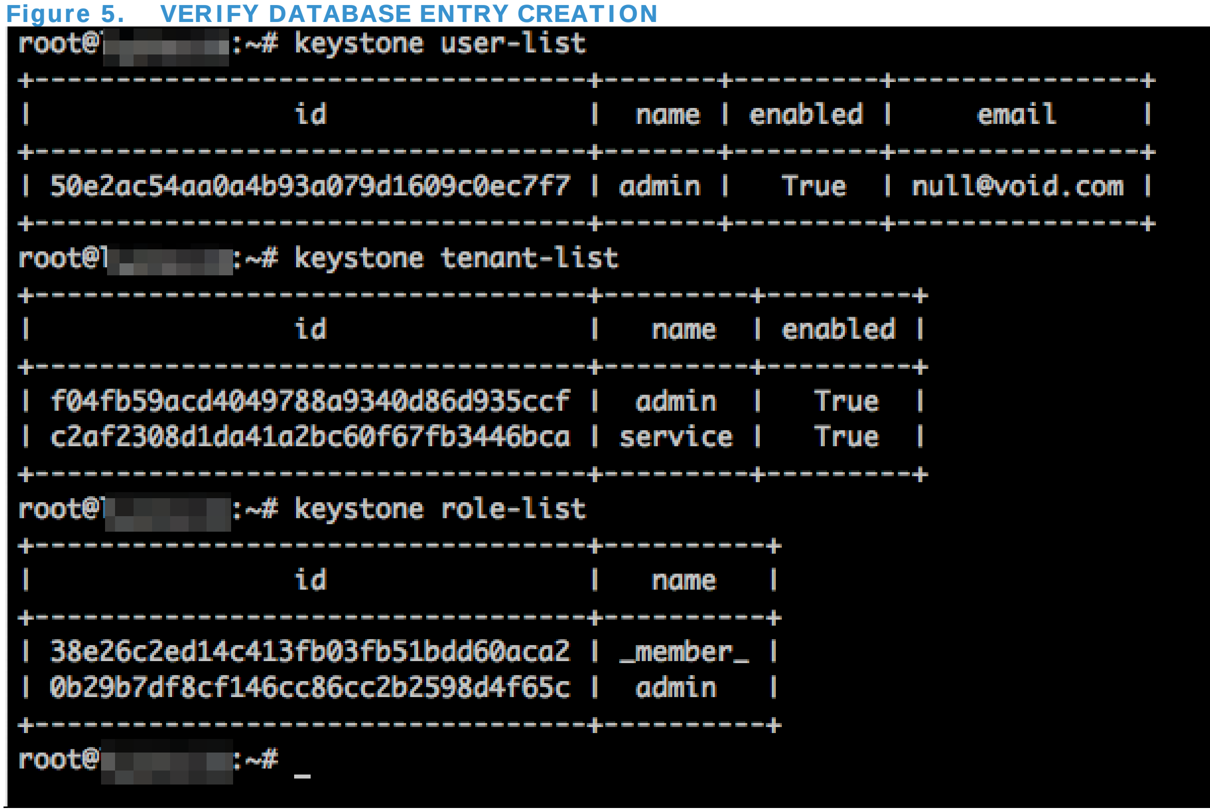
```
keystone user-role-add --tenant admin --user admin --role admin
keystone role-create --name _member_
keystone user-role-add --tenant admin --user admin --role _member_
keystone tenant-create --name service --description "Service Tenant"
keystone service-create --name keystone --type identity --description "Openstack identity"
keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ identity / {print $2}') \
  --publicurl http://controller:5000/v2.0 \
  --internalurl http://controller:5000/v2.0 \
  --adminurl http://controller:35357/v2.0
```

Execute the script file to perform the actions (you'll see a lot of output fly by):

```
# chmod +x build_keystone.sh
# build_keystone.sh
```

Verify database entry creation by running the command sequence as noted in the display in Figure 5.

Figure 5. VERIFY DATABASE ENTRY CREATION



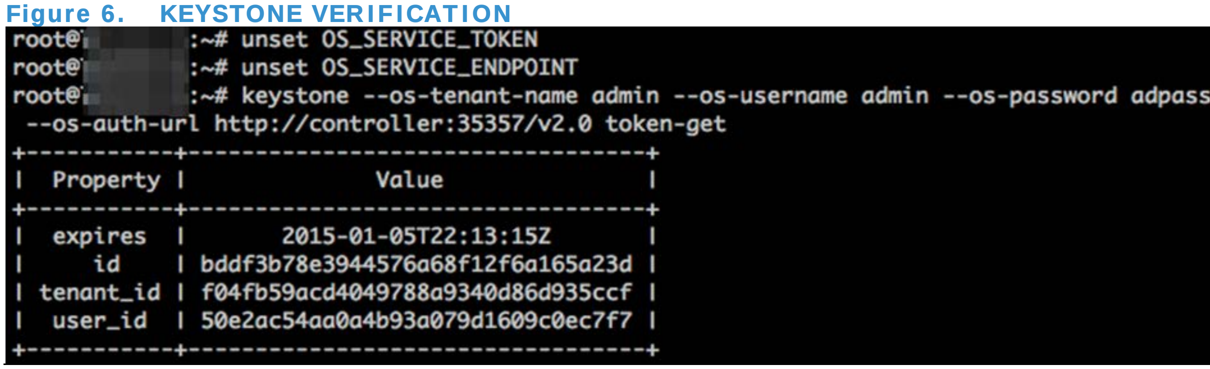
Perform an end-to-end test to ensure Keystone's working properly. The password is that of the _admin_ tenant (adpass):

```
# unset OS_SERVICE_TOKEN
# unset OS_SERVICE_ENDPOINT
```

```
# keystone --os-tenant-name admin --os-username admin --os-password adpass --os-auth-url
http://controller:35357/v2.0 token-get
```

You should see output similar to the one shown in Figure 6.

Figure 6. KEYSTONE VERIFICATION



```
root@[redacted]:~# unset OS_SERVICE_TOKEN
root@[redacted]:~# unset OS_SERVICE_ENDPOINT
root@[redacted]:~# keystone --os-tenant-name admin --os-username admin --os-password adpass
 --os-auth-url http://controller:35357/v2.0 token-get
+-----------+----------------------------------+
|  Property |              Value               |
+-----------+----------------------------------+
|  expires  |       2015-01-05T22:13:15Z       |
|     id    | bddf3b78e3944576a68f12f6a165a23d |
| tenant_id | f04fb59acd4049788a9340d86d935ccf |
|  user_id  | 50e2ac54aa0a4b93a079d1609c0ec7f7 |
+-----------+----------------------------------+
```

As a last step, create a .bash_profile file in root's home directory, changing the password if needed, then source the file:

```
export OS_TENANT_NAME=admin

export OS_USERNAME=admin

export OS_PASSWORD=adpass

export OS_AUTH_URL=http://controller:35357/v2.0
```

```
# source ~/.bash_profile
```

INSTALL AND CONFIGURE BLOCK STORAGE SERVICE

As root, create the database entries for the Cinder block storage service using the following commands (note that 'cdpass' is the Cinder password from the table):

```
# mysql -u root -p
> create database cinder;
> grant all privileges on cinder.* to cinder@<host_ip> identified by 'cdpass';
> grant all privileges on cinder.* to cinder@'%' identified by 'cdpass';
> exit;
```

The output should resemble the Keystone table sequence. Next, build the user, service, and endpoints (one for Cinder v1, and one for Cinder v2). Again, the easiest approach is to copy the text below into build_cinder.sh, changing only the cinder password if needed:

```
keystone user-create --name cinder --pass cdpass
keystone user-role-add --user cinder --tenant service --role admin
keystone service-create --name cinder --type volume --description "Openstack block storage (v1)"
keystone service-create --name cinderv2 --type volumev2 --description "Openstack block storage (v2)"
keystone endpoint-create  --service-id $(keystone service-list | awk '/ volume / {print $2}') \
  --publicurl http://controller:8776/v1/%\(tenant_id\)s \
```

```
  --internalurl http://controller:8776/v1/%\(tenant_id\)s \
  --adminurl http://controller:8776/v1/%\(tenant_id\)s
keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ volumev2 / {print $2}') \
  --publicurl http://controller:8776/v2/%\(tenant_id\)s \
  --internalurl http://controller:8776/v2/%\(tenant_id\)s \
  --adminurl http://controller:8776/v2/%\(tenant_id\)s
```

Execute the script file to perform the build:

```
# chmod +x build_cinder.sh
# build_cinder.sh
```

Install the Cinder infrastructure packages (you'll verify the database entries in a later step:

```
# apt-get -y install cinder-api cinder-scheduler cinder-volume python-cinderclient
```

Cinder installs a partial cinder.conf file but you need to add authentication, message queue, and other customizations before things will work. Append the following text to /etc/cinder/cinder.conf, substituting your IP address and passwords as needed. Don't change the "localhost" entry – mysql may fail if you do:

```
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = rbpass
my_ip = <host_ip>
rpc_response_timeout=300
[database]
connection = mysql://cinder:cdpass@localhost/cinder

[keystone_authtoken]
auth_uri = http://<host_ip>:5000/v2.0
identity_uri = http://<host_ip>:35357
auth_host = <host_ip>
auth_protocol = http
auth_port = 35357
admin_user = cinder
admin_tenant_name = service
admin_password = cdpass
```

To complete the Cinder initial installation, populate the database and reboot. Note that the cinder-manage syntax is slightly different than keystone-manage:

```
# cinder-manage db sync
# reboot
```

CONFIGURE CINDER FOR THIRD-PARTY STORAGE

Now that Cinder is installed, configure it to work with your particular block storage system. The details vary and are documented by the storage system vendor. This example uses a NetApp 8.2 7-mode iSCSI block storage provider, and the configuration information needed is available in the OpenStack Juno documentation set.

To begin, log in as root and add the appropriate information at the end of /etc/cinder/cinder.conf. In the NetApp case:

```
[netapp-iscsi]
volume_driver=cinder.volume.drivers.netapp.common.NetAppDriver
volume_backend_name=netapp-iscsi
netapp_login=<netapp_user_login>
netapp_password=<netapp_user_password>
netapp_server_hostname=<netapp_IP_address>
netapp_storage_family=ontap_7mode
netapp_storage_protocol=iscsi
netapp_transport_type=http
```

In the same file, under the "[DEFAULT]" heading, add:

```
enabled_backends=netapp-iscsi
```

Cinder uses the information you added in cinder.conf to define the array in its database. The following commands actually create the database entries:

```
# cinder type-create netapp-iscsi
# cinder type-key netapp-iscsi set volume_backend_name=netapp-iscsi
```

You can use "cinder extra-specs-list" to verify the operation. The command output should look similar to Figure 7.

Figure 7. CINDER TYPE CREATION OUTPUT

```
root@[redacted]:~# cinder type-create netapp-iscsi
+--------------------------------------+--------------+
|                  ID                  |     Name     |
+--------------------------------------+--------------+
| e44d5f93-4d83-4e35-9d58-bd64d628482d | netapp-iscsi |
+--------------------------------------+--------------+
root@[redacted]:~# cinder type-key netapp-iscsi set volume_backend_name=netapp-iscsi
root@[redacted]:~# cinder extra-specs-list
+--------------------------------------+--------------+-----------------------------------------+
|                  ID                  |     Name     |               extra_specs               |
+--------------------------------------+--------------+-----------------------------------------+
| e44d5f93-4d83-4e35-9d58-bd64d628482d | netapp-iscsi | {u'volume_backend_name': u'netapp-iscsi'} |
+--------------------------------------+--------------+-----------------------------------------+
```

Log into the target array via iSCSI and verify connection:

```
# iscsiadm -m discovery -t st -p <array_iSCSI_ip>
# iscsiadm -m node -L all
# iscsiadm -m node
```

The output should resemble Figure 8.

Figure 8. iSCSI DISCOVERY OUTPUT

```
root@[redacted]:~# iscsiadm -m discovery -t st -p 1[redacted].121
1[redacted].121:3260,1000 iqn.1992-08.com.netapp:sn.40[redacted]2
root@l[redacted]:~# iscsiadm -m node -L all
Logging in to [iface: default, target: iqn.1992-08.com.netapp:sn.40[redacted]2, portal: 10.[redacted].121,3260] (multiple)
Login to [iface: default, target: iqn.1992-08.com.netapp:sn.40[redacted]2, portal: 1[redacted].121,3260] successful.
root@[redacted]:~# iscsiadm -m node
1[redacted].121:3260,1000 iqn.1992-08.com.netapp:sn.40[redacted]2
```

Use the array utilities to configure an initiator record and reboot the host to pick up the Cinder configuration. When the host comes up, create a Cinder LUN. Build the LUN large enough that ViPR Controller can carve it up for host use.

First, check the iSCSI connection:

```
# iscsiadm -m node
```

You should see an iSCSI record appear. Verify that no devices are allocated, then create a volume (here, the “10” indicates a 10 GB LUN) using the commands below:

```
# cinder list
# cinder create --volume-type netapp-iscsi --display_name storage_pool 10
# cinder list
```

A valid output from this sequence will resemble Figure 9.

Figure 9. SEQUENCE OUTPUT

```
root@[redacted]:~# iscsiadm -m node
[redacted]:3260,1000 iqn.1992-08.com.netapp:sn.40[redacted]2
root@[redacted]:~# cinder list
+----+--------+--------------+------+-------------+----------+-------------+
| ID | Status | Display Name | Size | Volume Type | Bootable | Attached to |
+----+--------+--------------+------+-------------+----------+-------------+
+----+--------+--------------+------+-------------+----------+-------------+
root@[redacted]:~# cinder create --volume-type netapp-iscsi --display-name storage_pool 10
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                 nova                 |
|       bootable      |                false                 |
|      created_at     |      2015-01-06T15:05:18.535731      |
| display_description |                 None                 |
|     display_name    |             storage_pool             |
|      encrypted      |                False                 |
|          id         | 3c62251f-681c-4a39-a00a-bf4301056d27 |
|       metadata      |                  {}                  |
|         size        |                  10                  |
|     snapshot_id     |                 None                 |
|     source_volid    |                 None                 |
|        status       |               creating               |
|     volume_type     |             netapp-iscsi             |
+---------------------+--------------------------------------+
root@[redacted]:~# cinder list
+--------------------------------------+-----------+--------------+------+--------------+----------+-------------+
|                  ID                  |   Status  | Display Name | Size | Volume Type  | Bootable | Attached to |
+--------------------------------------+-----------+--------------+------+--------------+----------+-------------+
| 3c62251f-681c-4a39-a00a-bf4301056d27 | available | storage_pool |  10  | netapp-iscsi |  false   |             |
+--------------------------------------+-----------+--------------+------+--------------+----------+-------------+
```

This completes the Cinder installation. Remember, you now have a total of 10 GB available for ViPR Controller to assign as virtual storage.

WATCH OUT FOR QUOTA LIMITS

There's some great installation and troubleshooting information at https://community.emc.com/docs/DOC-3724. One of the tips highlights a problem you might hit after setting up Cinder. You'll probably want to go ahead and start creating volumes. However, after you build a few, you're likely to hit a problem as shown in Figure 10.

Figure 10. EXAMPLE OF A POTENTIAL PROBLEM

```
[redacted]~# cinder list
+--------------------------------------+-----------+--------------+------+--------------+----------+-------------+
|                  ID                  |   Status  | Display Name | Size | Volume Type  | Bootable | Attached to |
+--------------------------------------+-----------+--------------+------+--------------+----------+-------------+
| 48ad6d15-59a1-4feb-92e4-542fc8e24a17 | available |  cap_test_8  |  1   | netapp-iscsi |  false   |             |
| 4994a003-1b8f-4c40-b59b-750de8cedfaf | available |  cap_test_5  |  1   | netapp-iscsi |  false   |             |
| 55c1703d-4b8d-44df-ab9e-5ef7aa584e62 | available |  cap_test_2  |  1   | netapp-iscsi |  false   |             |
| 66098ee6-e62e-4bac-bcf5-f835311ab430 | available |  cap_test_7  |  1   | netapp-iscsi |  false   |             |
| 6f2cc3ea-c12a-4a67-a1b6-c8d0eead15dd | available |  cap_test_6  |  1   | netapp-iscsi |  false   |             |
| 87a49e00-70de-4f7b-88c0-565d12ced13c | available |  cap_test_3  |  1   | netapp-iscsi |  false   |             |
| 96c5bcf2-d367-4129-8294-95b29a6b3da4 | available |  cap_test_1  |  1   | netapp-iscsi |  false   |             |
| 972d1a79-860e-486e-998d-773f206fde1c | available |  cap_test_9  |  1   | netapp-iscsi |  false   |             |
| c50b8729-7c22-4bfc-92a8-71bb04d6225e | available | storage_pool |  10  | netapp-iscsi |  false   |             |
| de280efb-4d56-4f5f-8c8d-bd389bc4a5fa | available |  cap_test_4  |  1   | netapp-iscsi |  false   |             |
+--------------------------------------+-----------+--------------+------+--------------+----------+-------------+
[redacted]:~# cinder create --volume-type netapp-iscsi --display_name cap_test_10 1
ERROR: VolumeLimitExceeded: Maximum number of volumes allowed (10) exceeded (HTTP 413) (Request-ID: req-f24392db-05
[redacted]:~#
```

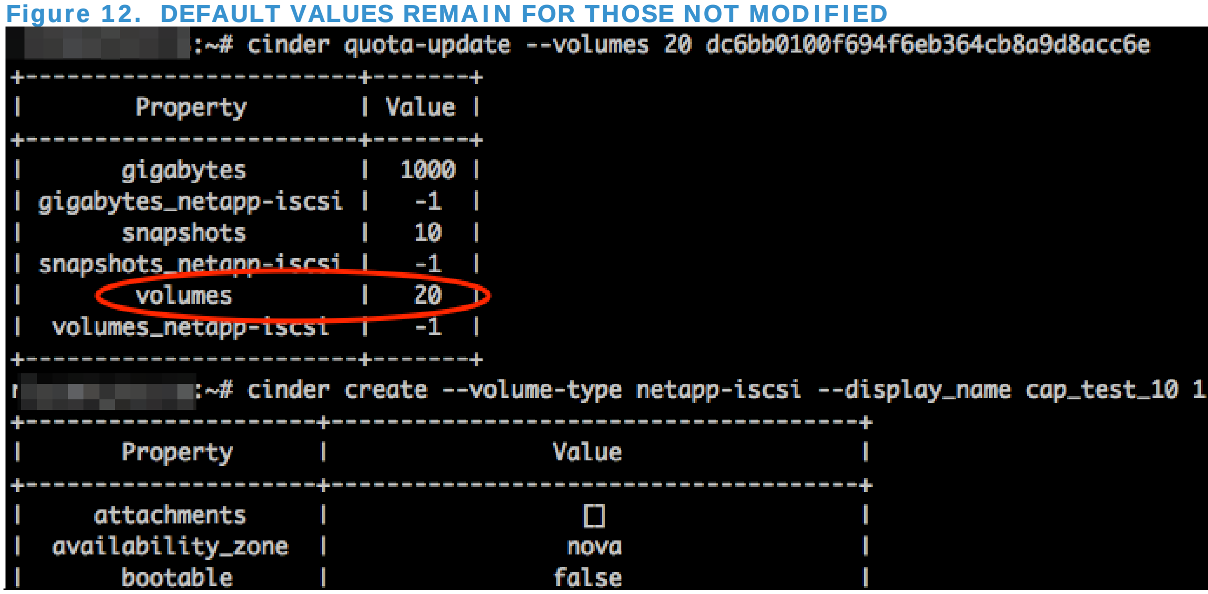
Out of the box Cinder allows 10 volumes per tenant. Interestingly, there's no clean way to list the volumes as a total; you can only see those for the current tenant. In order to see why this happens, use "keystone tenant-list" to determine the GUID for your tenant, and then use "cinder quota-show" to list the quotas as shown in Figure 11.

Figure 11. LIST OF CINDER QUOTAS

```
[redacted]:~# keystone tenant-list
+----------------------------------+---------+---------+
|                id                |   name  | enabled |
+----------------------------------+---------+---------+
| dc6bb0100f694f6eb364cb8a9d8acc6e |  admin  |   True  |
| cd39c532ff784c21989c4b0de02e2e9e | service |   True  |
+----------------------------------+---------+---------+
[redacted]:~# cinder quota-show dc6bb0100f694f6eb364cb8a9d8acc6e
+-----------------------+-------+
|        Property       | Value |
+-----------------------+-------+
|       gigabytes       |  1000 |
| gigabytes_netapp-iscsi |   -1  |
|       snapshots       |   10  |
| snapshots_netapp-iscsi |   -1  |
|        volumes        |   10  |
|  volumes_netapp-iscsi  |   -1  |
+-----------------------+-------+
```

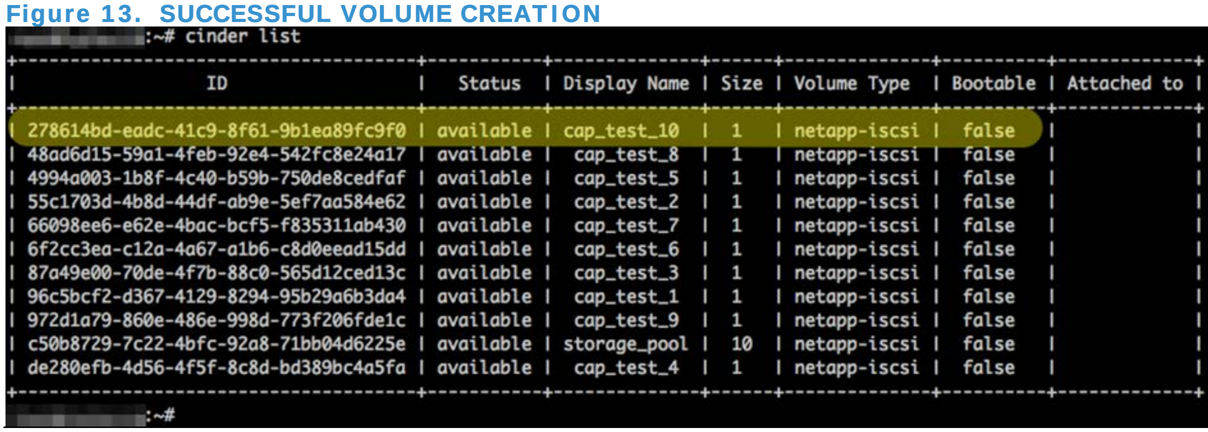
If you're restricting quotas by user account, use "keystone user-list --tenant-id <tenant GUID> instead. Quotas also affect snapshots, and space allocation allowed (1 TB). To change those values, use the GUID of the tenant, and update the desired quantity with "cinder quota-update", noting that default values remain for those not modified as shown in Figure 12.

Figure 12. DEFAULT VALUES REMAIN FOR THOSE NOT MODIFIED



After the quota modifications, the 11th volume creation proceeded without incident as shown below in Figure 13.

Figure 13. SUCCESSFUL VOLUME CREATION



Be sure your infrastructure can actually handle the increased values before modifying quotas.

SET UP VIPR CONTROLLER

ViPR Controller works within a virtual world, but it needs to understand physical configuration before it can construct its virtualization. This section assumes that iSCSI is correctly installed on the target host per the previous sections of this document and that ViPR Controller 2.2 is up and running.

CREATE NETWORK AND HOST OBJECTS

Start by creating a network for your target hosts and array ports. As root, log into ViPR Controller, then under Physical Assets | Networks, click "Add IP network", and provide a name (for example, "iSCSI_network").

Next, identify the target host to ViPR Controller. Log into the target host and obtain its iSCSI initiator name and IP address by running:

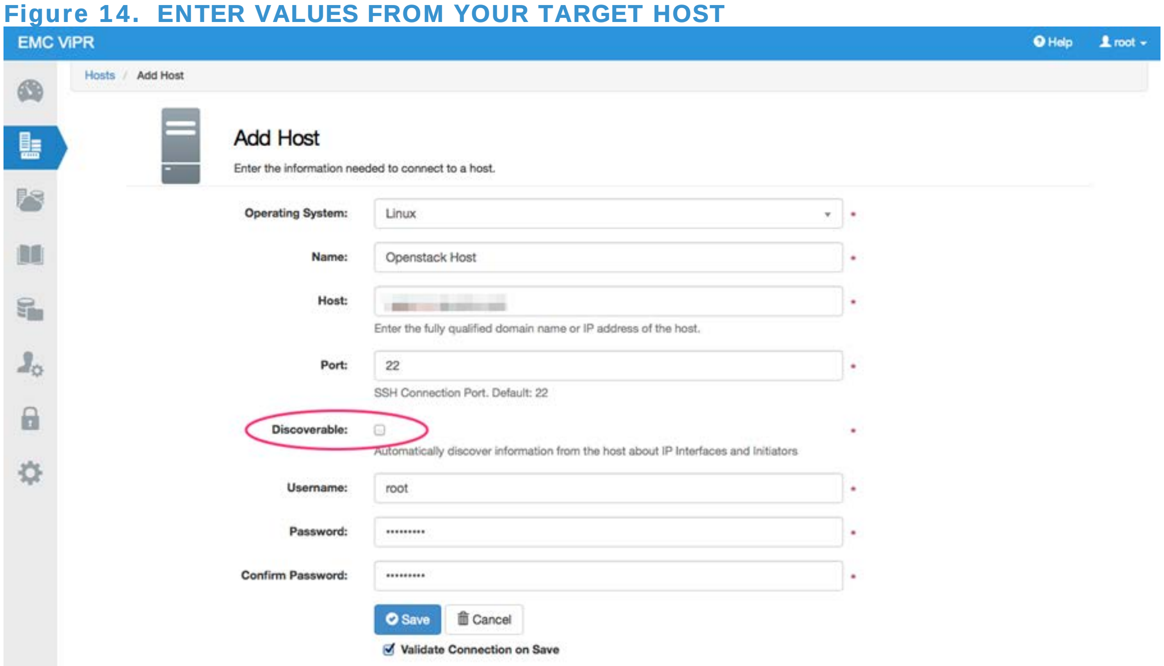
```
# cat /etc/iscsi/initiatorname.iscsi
```

The result contains initiator information similar to:

```
InitiatorName=iqn.1994-05.com.redhat:1be597762a81
```

In ViPR Controller, navigate to Physical Assets | Hosts, then click “Add”, and provide the values from your target host (“name” can be anything you want) as shown in Figure 14.

Figure 14. ENTER VALUES FROM YOUR TARGET HOST



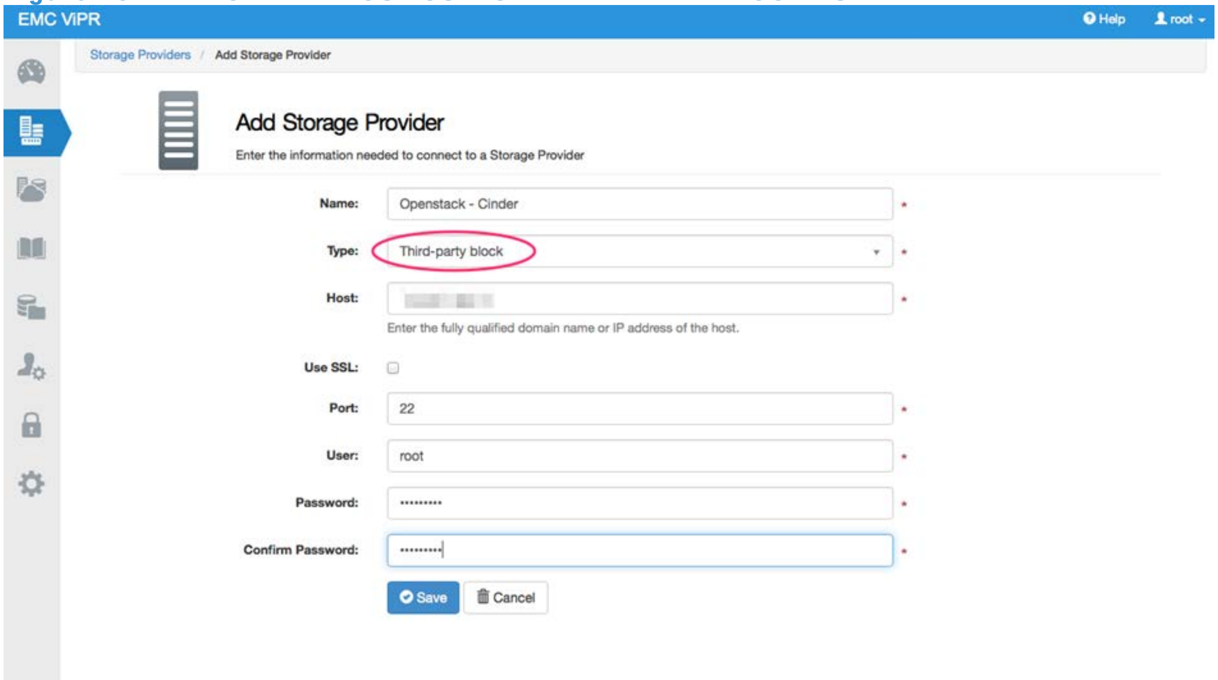
Once the host appears, click on “Initiators”, then “Add”. In the “Port” box, paste in the IQN for your target host, leaving the “Node” box empty.

Why not use auto-discovery? You can, but if your host has Fibre Channel cards installed – whether connected or not – ViPR Controller may see the drivers and add the FC WWNs to its database. Later on, ViPR Controller can get confused as to what communication channels are available. Manual setup keeps ViPR Controller from finding those FC drivers.

CONFIGURE VIPR CONTROLLER FOR CINDER STORAGE

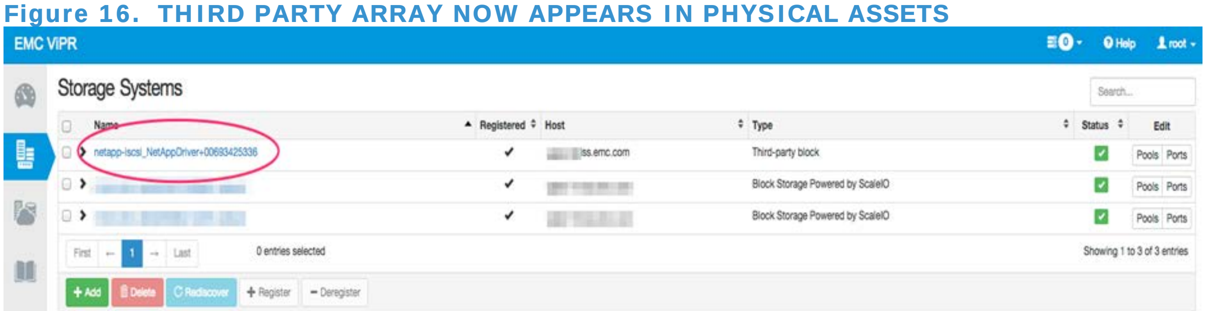
In ViPR Controller, under Tenant Settings | Projects, create a project (“Project1”), and then open Physical Assets | Storage Providers. Add the Cinder host using “Third-party block” as the type as shown in Figure 15; SSL is optional.

Figure 15. ADD CINDER HOST USING THIRD-PARTY BLOCK AS THE TYPE



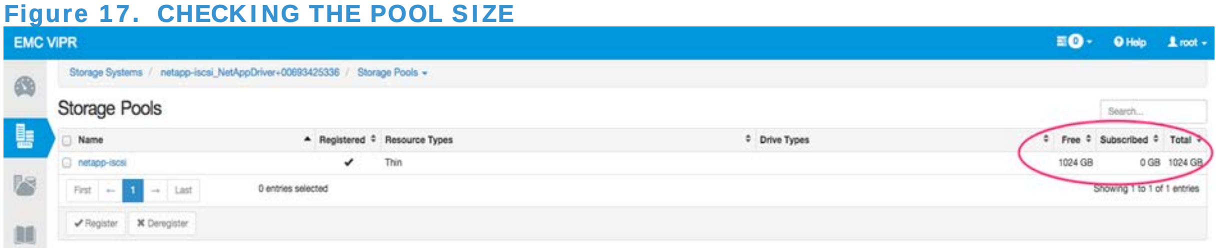
After the save and discovery completes, click on Physical Assets | Storage Systems, and the third-party array should display as shown in Figure 16.

Figure 16. THIRD PARTY ARRAY NOW APPEARS IN PHYSICAL ASSETS



Click on "pools" and you will see that storage exists in the pool as shown below in Figure 17 – as a matter of fact, a lot of storage considering that the volume was only 10 GB in size.

Figure 17. CHECKING THE POOL SIZE

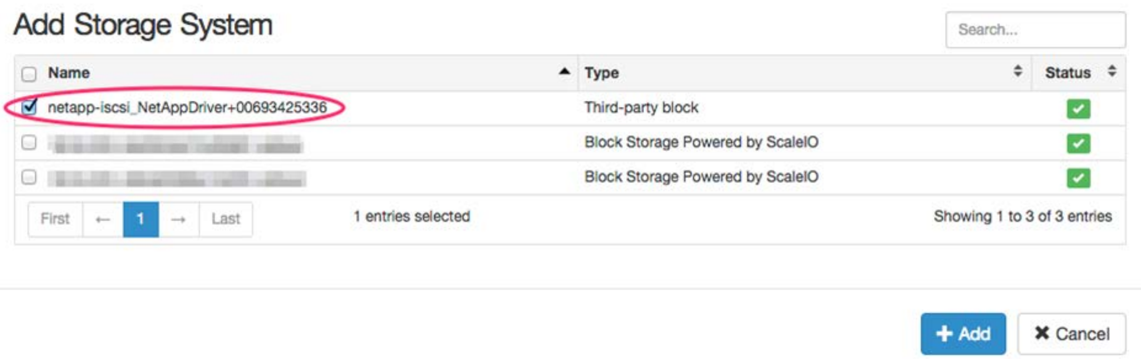


Where did that 1024 GB come from? As no storage has been allocated from this pool, ViPR Controller doesn't have any valid information from Cinder as to what's available. That's a normal display.

CREATE A VIRTUAL ARRAY AND POOL

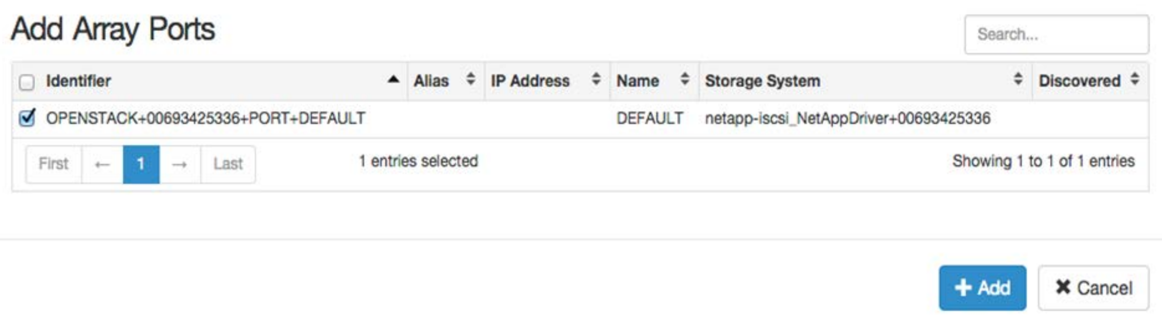
Under “Virtual Assets | Virtual Arrays”, create a new virtual array to serve out the third-party storage. Select “Add Storage System”, and check the netapp-iscsi driver as a resource as shown in Figure 18.

Figure 18. ADD NETAPP STORAGE



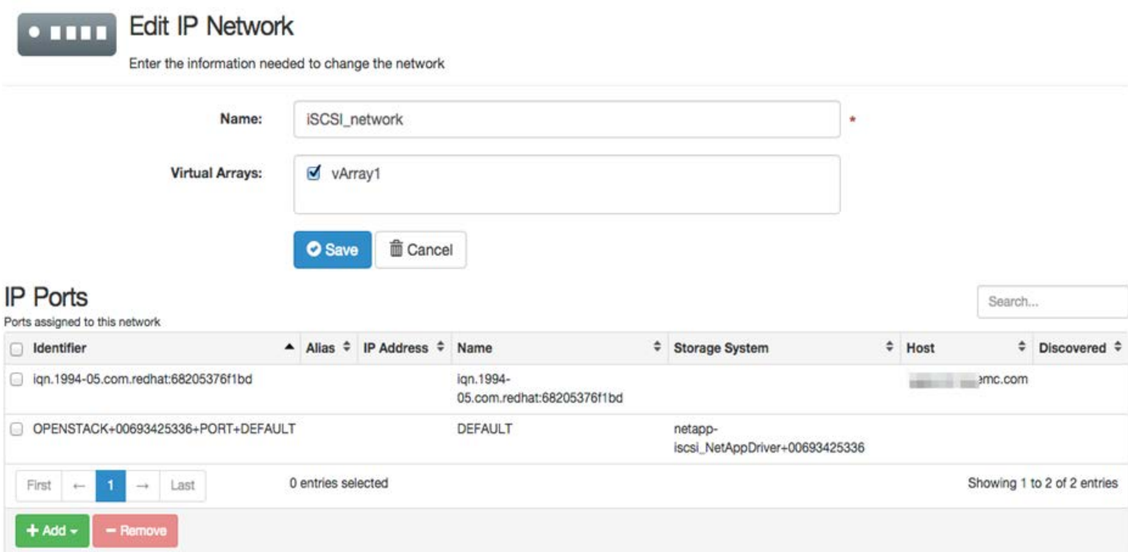
Under Physical Assets | Networks, open the iSCSI network previously created. Use the “Add” button and the “Add Array Ports” dropdown, and the discovered port displays as shown in Figure 19.

Figure 19. ADDING ARRAY PORTS



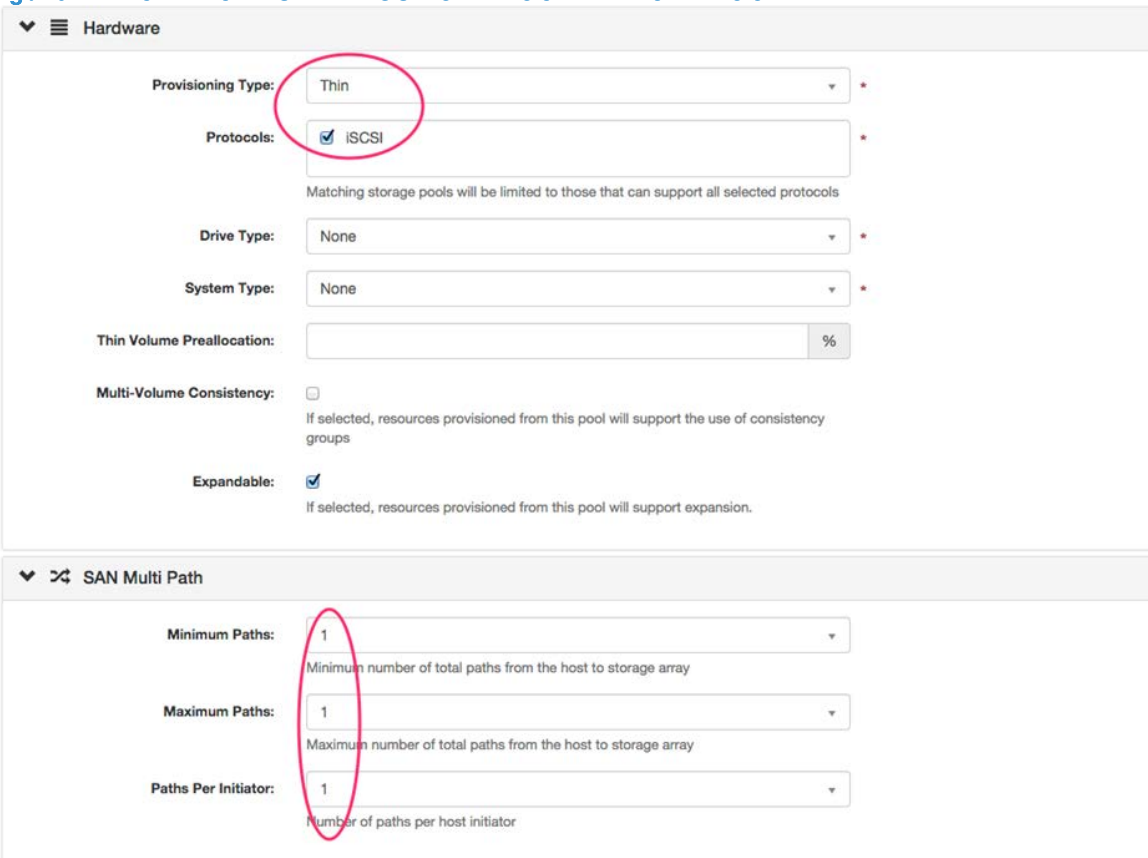
Ensure that the array entry is checked and both the target host and the array port appear, then click “Save”, as shown in Figure 20.

Figure 20. VERIFYING THE TARGET AND ARRAY PORTS



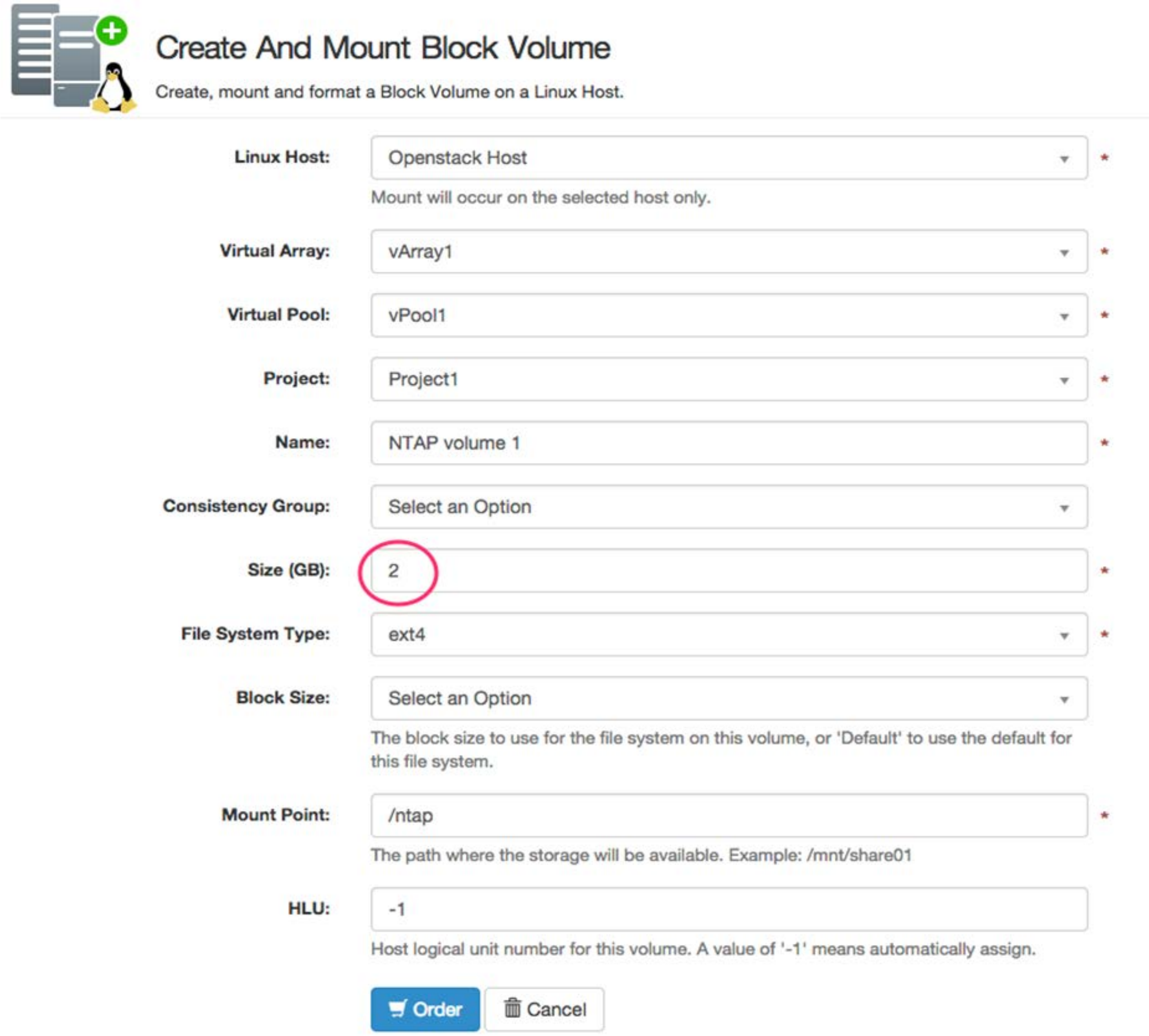
Wait a few minutes for ViPR Controller to rescan, and then create a block virtual pool (vPool1) using the new array. Critical settings include making it a thin iSCSI pool and setting multipath to 1 minimum, 1 maximum, and 1 path per initiator as shown in Figure 21. Otherwise, ViPR Controller may expect multiple paths, and as a result won't consider your array as a viable candidate for this pool:

Figure 21. CRITICAL SETTINGS FOR BLOCK VIRTUAL POOL



Once that completes, in ViPR Controller's Service Catalog, go to "Block Services for Linux", select "Create and Mount Block Volume" as shown in Figure 22, then add the name of the target (physical) host, the new array, the new virtual pool, and so forth. Make sure you provide a mount point (our example is "/ntap") that doesn't exist on the host – otherwise ViPR Controller cannot create it, and a supported filesystem (ext4):

Figure 22. CREATE AND MOUNT BLOCK VOLUME WITHIN VIPR CONTROLLER SERVICE CATALOG



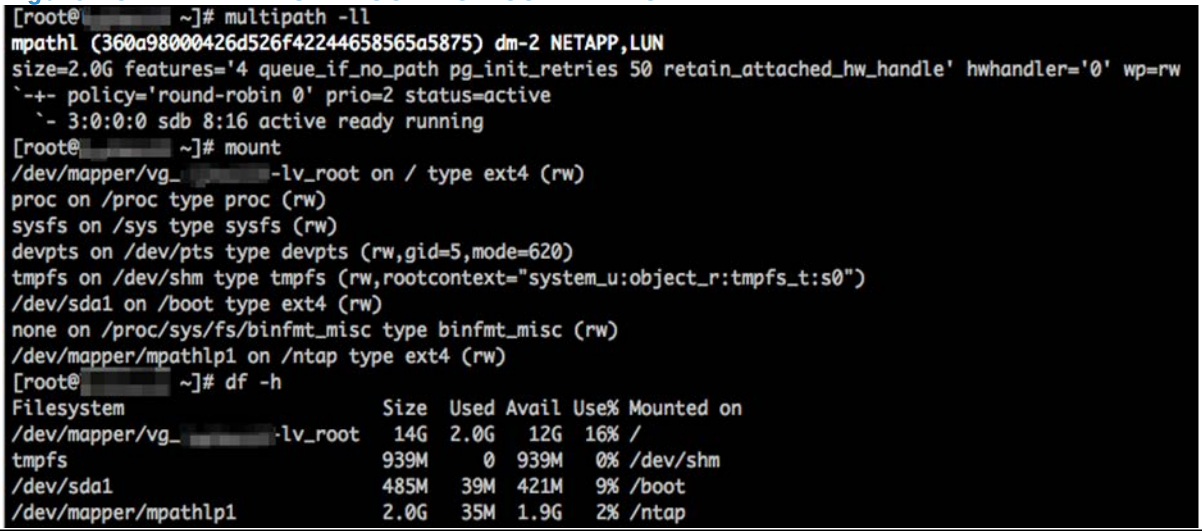
You may need to reboot your target host, but once done, you should see your new volume, formatted and mounted at /ntap. The following commands will confirm the volume's presence, as illustrated below:

 # multipath –ll (look for the "mpath" identifier)

 # mount (make sure the mpath partition is mounted)

 # df –h (check the amount of space available)

Figure 23. VERIFYING ALLOCATION COMPLETION



```
[root@[redacted] ~]# multipath -ll
mpathl (360a98000426d526f42244658565a5875) dm-2 NETAPP,LUN
size=2.0G features='4 queue_if_no_path pg_init_retries 50 retain_attached_hw_handle' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=2 status=active
  `- 3:0:0:0 sdb 8:16 active ready running
[root@[redacted] ~]# mount
/dev/mapper/vg_[redacted]-lv_root on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw,rootcontext="system_u:object_r:tmpfs_t:s0")
/dev/sda1 on /boot type ext4 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/dev/mapper/mpathlp1 on /ntap type ext4 (rw)
[root@[redacted] ~]# df -h
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/vg_[redacted]-lv_root  14G  2.0G   12G  16% /
tmpfs                        939M     0  939M   0% /dev/shm
/dev/sda1                    485M   39M  421M   9% /boot
/dev/mapper/mpathlp1         2.0G   35M  1.9G   2% /ntap
```

CONCLUSION

This document has demonstrated how to build a single-host Cinder platform for ViPR Controller using Ubuntu 14.04. Whether implemented on a virtual or physical host, the technique enables ViPR Controller to use OpenStack’s Cinder modules in provisioning from arrays that ViPR Controller may not natively support.

The author would like to express appreciation to the OpenStack / Cinder development and documentation team, as some of the information contained herein originated in their work.

Again, EMC strongly recommends a thorough review of the ViPR Controller and OpenStack documentation and appropriate validation to determine suitability before deploying in any production environment.