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

ABSTRACT

This white paper describes how to build a lightweight VM or physical host as a ViPR Controller 2.3 third-party storage provider using Red Hat (CentOS) Linux 7.1 via a Fibre Channel connection. The approach illustrated is intended for evaluation and test purposes but similar techniques could be used for production deployments.

July, 2015

REDEFINE

EMC WHITE

EMC²

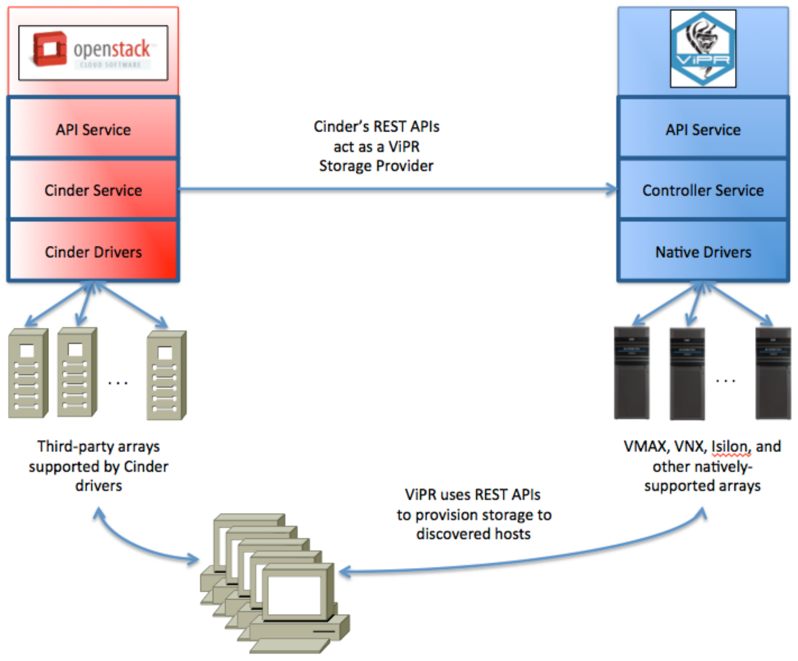
TABLE OF CONTENTS

EXECUTIVE SUMMARY

The explosive growth of OpenStack raises a need for flexible and open storage platform management. EMC® ViPR® Controller 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, users can create "single-pane-of-glass" management portals from both storage and instance viewpoints, 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 CentOS 7.1 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 Kilo release. The configuration looks like this:

Figure 1. ViPR CONTROLLER STORAGE PROVIDER BASED ON CentOS 7.1


openstack
API Service
Cinder Service
Cinder Drivers
Cinder's REST APIs act as a ViPR Storage Provider
ViPR
API Service
Controller Service
Native Drivers
...
Third-party arrays supported by Cinder drivers
VMAX, VNX, Isilon, and other natively-supported arrays
ViPR uses REST APIs to provision storage to discovered hosts

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

WHY OPENSTACK KILO?

The OpenStack cloud software stack contains modular components to handle compute (Nova), object storage (Swift), block storage (Cinder), and networking (Neutron), among others. 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 allowing cloud users to manage device creation, snapshot, and other storage functions while hiding vendor-specific implementation and control details. Kilo 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.

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 8 GB of RAM, at least 8 GB of disk space and at least one NIC. A VM meeting these specifications will work.
- A second hard disk (or available partition space)
- A VMware environment containing a ViPR Controller 2.x instance
- At least one “target” host connected to the array and supported by ViPR Controller. This host will consume the storage
- 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: Red Hat Linux 7.1 running OpenStack Kilo with IP connections to the target array and switch(es)
- ViPR Controller host: ViPR Controller 2.3.0.0.828 hosted on VMware ESXi 5.5
- Target host: CentOS 7.1 using a QLogic Fibre Channel HBA attached to an IBM SVC (SAN Volume Controller)

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), then paste it to your command line. Most OpenStack command arguments start with a double dash (“--“).

CREATE THE HOST ENVIRONMENT

INSTALL A BASIC RED HAT LINUX HOST

To build the Cinder host, obtain Red Hat Linux through your normal channels. Boot your host on the image and perform a minimal OS installation, adding “debugging tools”, “compatibility libraries”, and “development tools” from the Software Selection page. Automatic disk partitioning is appropriate in most situations; be sure to set up your network and host name, using static IP addressing rather than DHCP.

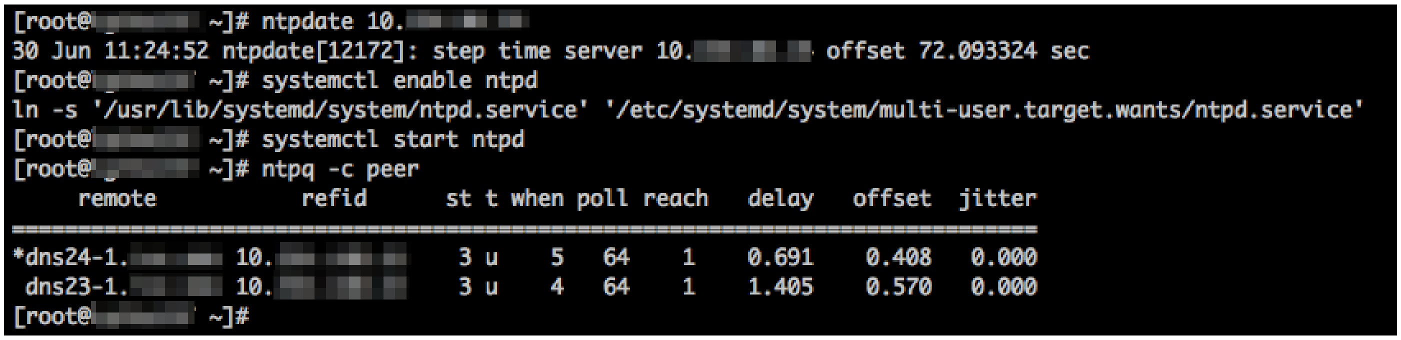
When your installation is complete, SSH into the host. Set up NTP time synchronization if a server is available on your network:

```
# yum –y install ntp
```

Edit /etc/ntp.conf to match your local preferences, removing “nopeer” and “noquery” from the restrict clause. Start up your ntp daemon and test it out. A normal startup sequence is shown below:

```
# ntpdate <ip-address-of-your-local-NTP-server>
# systemctl enable ntpd
# systemctl start ntpd
# ntpq –c peer
```

Figure 2. SAMPLE OUTPUT FROM NTPD SETUP



```
[root@[redacted] ~]# ntpdate 10.[redacted]
30 Jun 11:24:52 ntpdate[12172]: step time server 10.[redacted] offset 72.093324 sec
[root@[redacted] ~]# systemctl enable ntpd
ln -s '/usr/lib/systemd/system/ntpd.service' '/etc/systemd/system/multi-user.target.wants/ntpd.service'
[root@[redacted] ~]# systemctl start ntpd
[root@[redacted] ~]# ntpq -c peer
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*dns24-1.[redacted] 10.[redacted]     3 u    5   64    1    0.691    0.408   0.000
 dns23-1.[redacted] 10.[redacted]     3 u    4   64    1    1.405    0.570   0.000
[root@[redacted] ~]#
```

If your Cinder host is on a VMware virtual machine, it's a good idea to install the integration tools:

```
# yum –y install open-vm-tools
```

Firewalls and SELinux can cause problems with OpenStack. Although it is possible to run with them enabled, during this demonstration turn off both services for this installation, then edit /etc/selinux/config and set "SELINUX=disabled".

```
# systemctl stop firewalld
# systemctl disable firewalld
# setenforce 0
```

Cinder requires a few aliases to ease configuration, so edit /etc/hosts and delete any lines that may be in the file, then create your host entry and loopback as follows (note the "controller" and "localhost" entries):

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

Create a logical volume for Cinder's use. If you have a second hard drive, then use fdisk to build a partition, and if you're using a single drive, then use spare space on the disk to create a partition. Then, use the following commands to create the Cinder volume (using /dev/sdb1 as our example). The name "cinder-volumes" is significant, so be sure to use it:

```
# pvcreate /dev/sdb1
# vgcreate cinder-volumes /dev/sdb1
```

Install the Kilo and EPEL release repositories, then apply any needed upgrades and reboot the host:

```
# yum –y install http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
# yum -y install https://repos.fedorapeople.org/repos/openstack/openstack-kilo/rdo-release-kilo-
1.noarch.rpm
# yum –y upgrade
# reboot
```

INSTALL AND CONFIGURE KEYSTONE 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 OPENSTACK COMPONENT CREDENTIALS

Variable Name	Password	Description
<database>	dbpass	Root password for the database
RABBIT_PASS	rbpass	RabbitMQ user password
KEYSTONE_PASS	kypass	Keystone identity administration
ADMIN_PASS	adpass	OpenStack admin user
CINDER_PASS	cdpass	Cinder user credential

INSTALL DATABASE AND MESSAGE QUEUE SERVICES

After logging into the freshly-rebooted host as root, install the database and Python extensions:

```
# yum -y install mariadb mariadb-server MySQL-python
```

Configure the SQL engine by creating /etc/my.cnf.d/mariadb_openstack.cnf containing the following information. Insert your Cinder node’s IP address in the bind-address line:

```
[mysqld]:
bind-address = <your-ip-address>
default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

After starting the database and setting it to start at boot, make the installation secure using the mysql_secure_installation script:

```
# systemctl enable mariadb
# systemctl start mariadb
# mysql_secure_installation
```

Press <enter> at the password prompt (as none has been set), enter the database password from Table 1, and take the defaults for the remaining options by pressing <enter> at each prompt.

Next, install the messaging server. After installation, enable automatic start, start the server, and create the user, then set the needed permissions. Use the Rabbit password from Table 1:

```
# yum -y install rabbitmq-server
# systemctl enable rabbitmq-server
# systemctl start rabbitmq-server
# rabbitmqctl add_user openstack rbpass
# rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

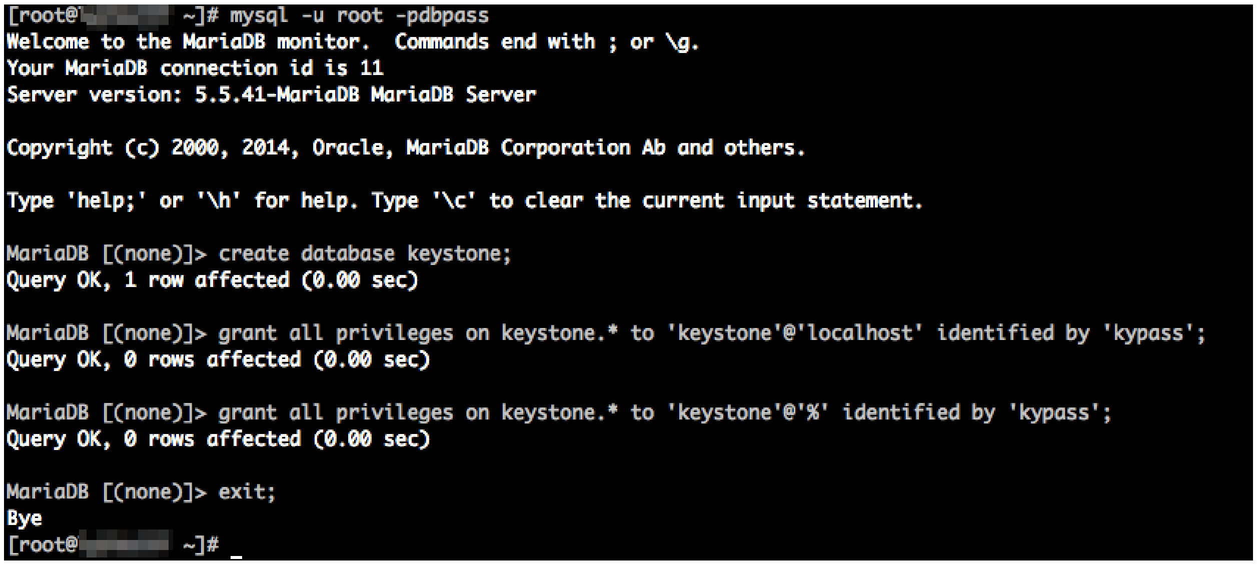
INSTALL AND CONFIGURE AUTHENTICATION SERVICES

Now that mariadb is up and running, build the database for the Keystone authentication service. Log into the database (the parameters are “-u root” for the user, and “-pdbpass” to use dbpass as the password; there’s no space in the latter). When creating the database and assigning privileges, terminate each command with a semicolon. Use the Keystone password from Table 1:

```
# mysql –u root –pdbpass
> create database keystone;
> grant all privileges on keystone.* to 'keystone'@'localhost' identified by 'kypass';
> grant all privileges on keystone.* to 'keystone'@'%' identified by 'kypass';
> exit;
```

Figure 3 provides sample output from a successful database creation.

Figure 3. MARIADB CONFIGURATION



```
[root@[redacted] ~]# mysql -u root -pdbpass
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 5.5.41-MariaDB MariaDB Server

Copyright (c) 2000, 2014, Oracle, MariaDB Corporation 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'@'localhost' 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] ~]# _
```

When the database installation is complete, install the keystone components. We'll use the Apache HTTP server to handle data requests and memcached to store tokens (instead of using a SQL database):

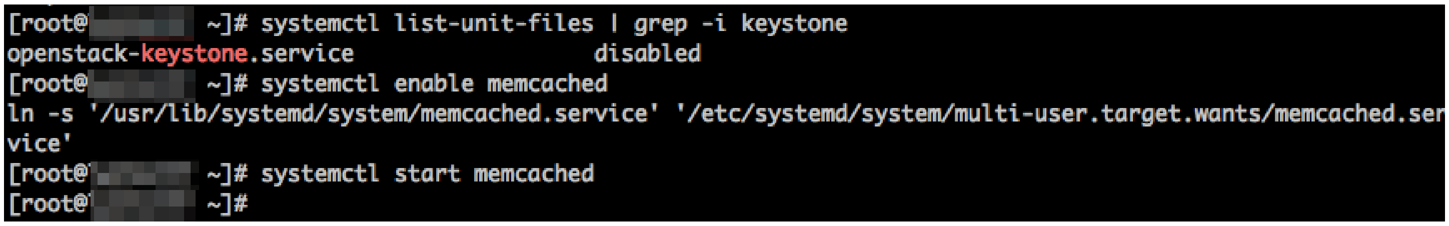
```
# yum -y install openstack-keystone httpd mod_wsgi python-openstackclient memcached python-
memcached
```

Your Keystone service should install in a disabled state. Verify installation using systemctl, then start the memory cache daemon:

```
# systemctl list-unit-files | grep –i keystone
# systemctl enable memcached
# systemctl start memcached
```

The sequence should resemble this:

Figure 4. MEMCACHED STARTUP



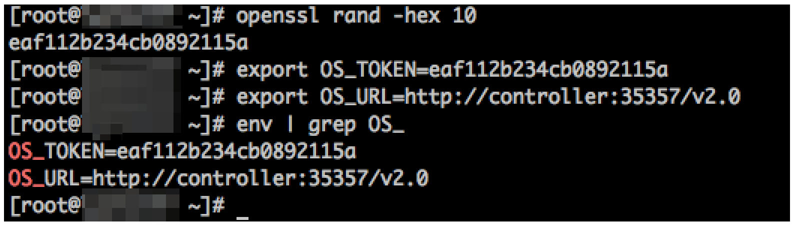
```
[root@[redacted] ~]# systemctl list-unit-files | grep -i keystone
openstack-keystone.service                    disabled
[root@[redacted] ~]# systemctl enable memcached
ln -s '/usr/lib/systemd/system/memcached.service' '/etc/systemd/system/multi-user.target.wants/memcached.ser
vice'
[root@[redacted] ~]# systemctl start memcached
[root@[redacted] ~]#
```

Generate a random hex digit string to use as an initial token and copy the output to the clipboard. Using that value, set the environment variables needed in the Keystone configuration procedure:

```
# openssl rand -hex 10
# export OS_TOKEN=<your token>
# export OS_URL=http://controller:35357/v2.0
# env | grep OS_
```

Here is an example of the token setup operation:

Figure 5.   TOKEN AND ENVIRONMENT VARIABLES



```
[root@[redacted] ~]# openssl rand -hex 10
eaf112b234cb0892115a
[root@[redacted] ~]# export OS_TOKEN=eaf112b234cb0892115a
[root@[redacted] ~]# export OS_URL=http://controller:35357/v2.0
[root@[redacted] ~]# env | grep OS_
OS_TOKEN=eaf112b234cb0892115a
OS_URL=http://controller:35357/v2.0
[root@[redacted] ~]# _
```

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

- Under [DEFAULT], uncomment “admin_token” and append your random string value.
- Under [database], change “connection = <None>” to “connection = mysql://keystone:**kypass**@controller/keystone”, replacing “kypass” with the Keystone password from Table 1.
- Under [memcache], uncomment “servers = localhost:11211”
- Under [revoke], uncomment “driver = keystone.contrib.revoke.backends.sql.Revoke”
- Under [token], uncomment “provider = keystone.token.providers.uuid.Provider”
- Under [token], uncomment “driver = keystone.token.persistence.backends.**sql**.Token”, changing “sql” to “memcache”

Keystone will populate its database when you run the following command (it produces no output on a normal run):

```
# keystone-manage db_sync
```

CONFIGURE APACHE AND COMPLETE KEYSTONE SETUP

In order for Apache’s HTTPD service to provide WSGI components to Keystone, the server must be configured. First, edit /etc/httpd/conf/httpd.conf and search for ServerName. The parameter line looks similar to:

```
#ServerName www.example.com:80
```

Replace the line with:

```
ServerName controller
```

Next, create /etc/httpd/conf.d/wsgi-keystone.conf and drop in the following text (no customization is necessary):

```
Listen 5000
Listen 35357
<VirtualHost *:5000>
    WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystone group=keystone display-
name=%{GROUP}
    WSGIProcessGroup keystone-public
    WSGIScriptAlias / /var/www/cgi-bin/keystone/main
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    LogLevel info
    ErrorLogFormat "%{cu}t %M"
    ErrorLog /var/log/httpd/keystone-error.log
    CustomLog /var/log/httpd/keystone-access.log combined
</VirtualHost>
<VirtualHost *:35357>
    WSGIDaemonProcess keystone-admin processes=5 threads=1 user=keystone group=keystone display-
name=%{GROUP}
    WSGIProcessGroup keystone-admin
    WSGIScriptAlias / /var/www/cgi-bin/keystone/admin
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    LogLevel info
```

```
        ErrorLogFormat "%{cu}t %M"
        ErrorLog /var/log/httpd/keystone-error.log
        CustomLog /var/log/httpd/keystone-access.log combined
    </VirtualHost>
```

Create a directory structure for the WSGI components, then copy the components from the upstream repository. Finally, adjust ownership and restart the Apache HTTP server:

```
# mkdir -p /var/www/cgi-bin/keystone
# curl http://git.openstack.org/cgit/openstack/keystone/plain/httpd/keystone.py?h=stable/kilo \
  | tee /var/www/cgi-bin/keystone/main /var/www/cgi-bin/keystone/admin
# chown -R keystone:keystone /var/www/cgi-bin/keystone
# chmod 755 /var/www/cgi-bin/keystone/*
# chmod 777 /var/log/keystone/keystone.log
# systemctl enable httpd.service
# systemctl start httpd.service
```

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 Table 1. Everything else is literal and stays as-is:

```
#!/bin/bash
openstack service create --name keystone --description "OpenStack Identity" identity
openstack endpoint create \
--publicurl http://controller:5000/v2.0 \
--internalurl http://controller:5000/v2.0 \
--adminurl http://controller:35357/v2.0 \
--region RegionOne identity
openstack project create --description "Admin Project" admin
openstack user create --password adpass admin
openstack role create admin
openstack role add --project admin --user admin admin
openstack project create --description "Service Project" service
```

Run the file with the following commands:

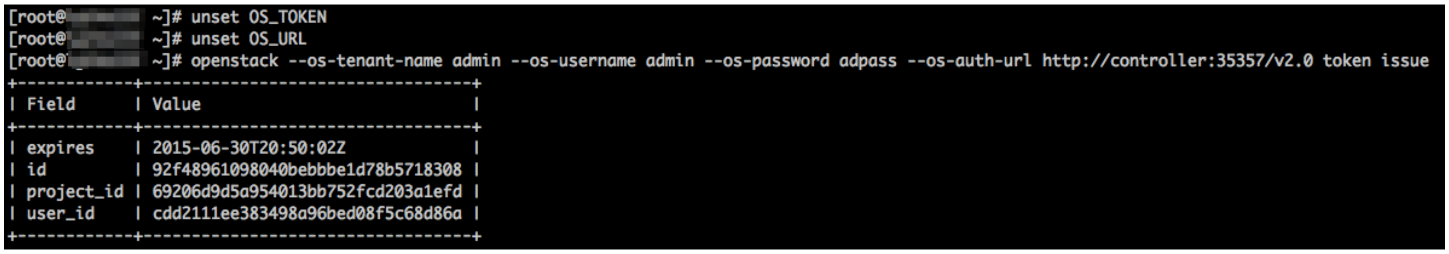
```
# chmod +x build_keystone.sh
./build_keystone.sh
```

The script will output a number of field/value pairs; you should see no traceback or error messages. To verify, unset the environment variables and do a token issue test (replace the password with the admin password from Table 1):

```
# unset OS_TOKEN
# unset OS_URL
# openstack --os-tenant-name admin --os-username admin --os-password adpass --os-auth-url
http://controller:35357/v2.0 token issue
```

Successful output resembles the figure below (your return values will be different):

Figure 6. TESTING THE KEYSTONE INSTALLATION



```
[root@[redacted] ~]# unset OS_TOKEN
[root@[redacted] ~]# unset OS_URL
[root@[redacted] ~]# openstack --os-tenant-name admin --os-username admin --os-password adpass --os-auth-url http://controller:35357/v2.0 token issue
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| expires    | 2015-06-30T20:50:02Z             |
| id         | 92f48961098040bebbbe1d78b5718308 |
| project_id | 69206d9d5a954013bb752fcd203a1efd |
| user_id    | cdd2111ee383498a96bed08f5c68d86a |
+------------+----------------------------------+
```

Complete the Keystone setup by editing ~/.bash_profile and appending the following to the file. Replace the OS_PASSWORD value with the one from Table 1:

```
export OS_PROJECT_DOMAIN_ID=default
export OS_USER_DOMAIN_ID=default
export OS_PROJECT_NAME=admin
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=adpass
export OS_AUTH_URL=http://controller:35357/v3
export OS_VOLUME_API_VERSION=2
```

Run the following command to install those variables in your environment:

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

INSTALL AND CONFIGURE CINDER COMPONENTS

INITIALIZE THE CINDER DATABASE

Like Keystone, Cinder requires entries in the OpenStack database. You create those entries the same way as you did for Keystone but with slightly different parameters. Refer to Table 1 for the passwords:

```
# mysql –u root –pdbpass
> create database cinder;
> grant all privileges on cinder.* to 'cinder'@'localhost' identified by 'cdpass';
> grant all privileges on cinder.* to 'cinder'@'%' identified by 'cdpass';
> exit;
```

The output should be similar to that in Figure 3. To populate the database, create build_cinder.sh containing the following, substituting your cinder password from Table 1. Everything else is literal and stays as-is:

```
#!/bin/bash
openstack user create --password cdpass cinder
openstack role add --project service --user cinder admin
openstack service create --name cinder --description "OpenStack Block Storage" volume
openstack service create --name cinderv2 --description "Openstack Block Storage V2" volumev2
openstack endpoint create \
--publicurl http://controller:8776/v1/%\(tenant_id\)s \
--internalurl http://controller:8776/v1/%\(tenant_id\)s \
--adminurl http://controller:8776/v1/%\(tenant_id\)s \
--region RegionOne volume
openstack endpoint create \
--publicurl http://controller:8776/v2/%\(tenant_id\)s \
--internalurl http://controller:8776/v2/%\(tenant_id\)s \
--adminurl http://controller:8776/v2/%\(tenant_id\)s \
--region RegionOne volumev2
```

Run the file with the following commands:

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

You should see a list of field/value pairs with no traceback or error messages – if you do receive any, correct any typing or syntax in your script file, delete any lines that have already run, and retry the script.

INSTALL AND CONFIGURE CINDER

Once that's successfully completed, your next task is to install the Cinder packages, then create a base configuration file:

```
# yum -y install openstack-cinder python-cinderclient python-oslo-db
# mv /etc/cinder/cinder.conf /etc/cinder/cinder.original
# cp /usr/share/cinder/cinder-dist.conf /etc/cinder/cinder.conf
```

The base file (/etc/cinder/cinder.conf) needs more information about how Cinder is configured. Open the file in an editor and make the following changes:

- Under [DEFAULT], add a new entry "rpc_backend = rabbit".

- Under [DEFAULT], add a new entry "my_ip = xxx.xxx.xxx.xxx", using your host's IP address.

- Under [database], in the "connection" line, change the cinder password (it's between the ':' and '@') to the one in Table 1. The default is "mysql://cinder:**cinder**@localhost/cinder", so the password to change is the second "cinder".

- Delete everything under [keystone_authtoken] and replace with the text below. Be sure to change the password and user fields to match the credentials in Table 1, and insert your Cinder host's IP address in the first two lines:

```
[keystone_authtoken]
auth_uri = http://your-ip-address:5000/v2.0
auth_url = http://your-ip-address:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = cinder
password = cdpass
admin_tenant_name = service
admin_password = cdpass
admin_user = cinder

[oslo_messaging_rabbit]
rabbit_host=controller
rabbit_userid=openstack
rabbit_password=rbpass

[oslo_concurrency]
lock_path = /var/lock/cinder
```

Since you copied the template for cinder.conf, the permissions probably aren't correct. Fix them by running:

```
# chmod 755 /etc/cinder/cinder.conf
```

Cinder also wants a lock directory. Create one and set permissions:

```
# mkdir /var/lock/cinder
# chmod 777 /var/lock/cinder
```

Also, add the following lines to /etc/rc.local (as the directory will disappear at reboot):

```
mkdir /var/lock/cinder
chmod 777 /var/lock/cinder
```

When done, set /etc/rc.local to executable:

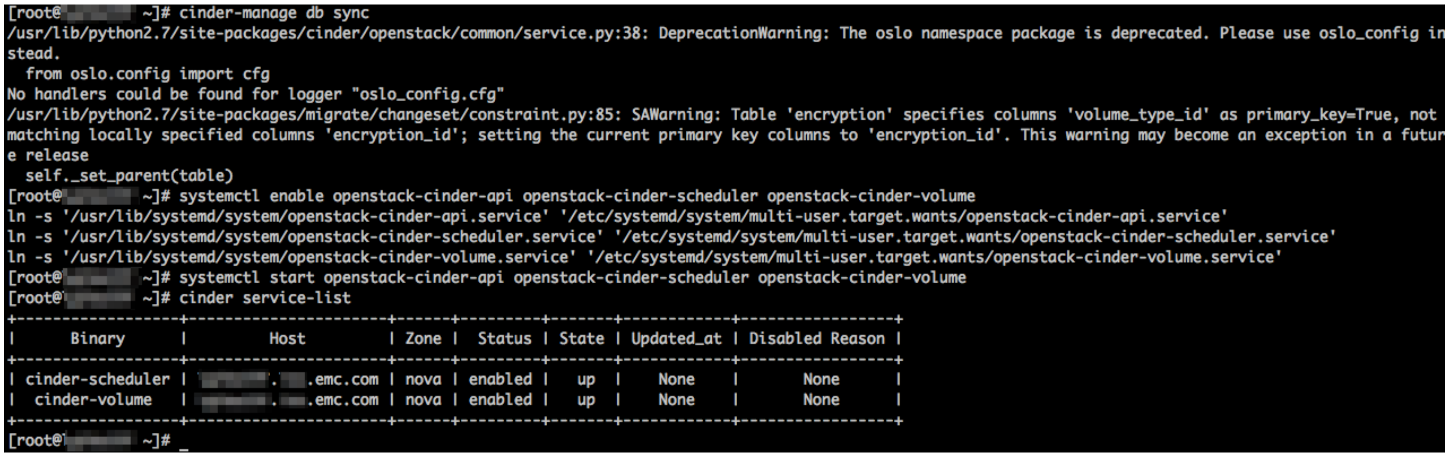
```
chmod +x /etc/rc.local
```

Populate the database with cinder-manage (note the syntax is different than keystone-manage). You may see an error message regarding oslo_config, but the operation will succeed – this is a known issue scheduled for a fix in a later Cinder release). After that completes, enable and start Cinder, then test:

```
# cinder-manage db sync
# systemctl enable openstack-cinder-api openstack-cinder-scheduler openstack-cinder-volume
# systemctl start openstack-cinder-api openstack-cinder-scheduler openstack-cinder-volume
# cinder service-list
```

The error message, and subsequent successful output of the command sequence, looks like this:

Figure 7. CINDER INITIALIZATION



```
[root@[redacted] ~]# cinder-manage db sync
/usr/lib/python2.7/site-packages/cinder/openstack/common/service.py:38: DeprecationWarning: The oslo namespace package is deprecated. Please use oslo_config in
stead.
  from oslo.config import cfg
No handlers could be found for logger "oslo_config.cfg"
/usr/lib/python2.7/site-packages/migrate/changeset/constraint.py:85: SAWarning: Table 'encryption' specifies columns 'volume_type_id' as primary_key=True, not
matching locally specified columns 'encryption_id'; setting the current primary key columns to 'encryption_id'. This warning may become an exception in a futur
e release
  self._set_parent(table)
[root@[redacted] ~]# systemctl enable openstack-cinder-api openstack-cinder-scheduler openstack-cinder-volume
ln -s '/usr/lib/systemd/system/openstack-cinder-api.service' '/etc/systemd/system/multi-user.target.wants/openstack-cinder-api.service'
ln -s '/usr/lib/systemd/system/openstack-cinder-scheduler.service' '/etc/systemd/system/multi-user.target.wants/openstack-cinder-scheduler.service'
ln -s '/usr/lib/systemd/system/openstack-cinder-volume.service' '/etc/systemd/system/multi-user.target.wants/openstack-cinder-volume.service'
[root@[redacted] ~]# systemctl start openstack-cinder-api openstack-cinder-scheduler openstack-cinder-volume
[root@[redacted] ~]# cinder service-list
+------------------+----------------------+------+---------+-------+------------+-----------------+
|      Binary      |         Host         | Zone |  Status | State | Updated_at | Disabled Reason |
+------------------+----------------------+------+---------+-------+------------+-----------------+
| cinder-scheduler | [redacted].[redacted].emc.com | nova | enabled |   up  |    None    |       None      |
|  cinder-volume   | [redacted].[redacted].emc.com | nova | enabled |   up  |    None    |       None      |
+------------------+----------------------+------+---------+-------+------------+-----------------+
[root@[redacted] ~]# _
```

DEFINE THE BACK-END STORAGE

Although the basic Cinder configuration is now complete, Cinder doesn’t know anything about the storage it’s going to manage. To define the back-end storage, you add a section to /etc/cinder.conf which calls out the specific parameters that enable the vendor-specific driver to talk to the array. This document uses an IBM SVC array, and the OpenStack web site (http://docs.openstack.org/kilo/config-reference/content/section_volume-drivers.html) calls out the parameters for the IBM SVC.

In the [DEFAULTS] section of the file, add the following:

```
enabled_backends = ibm-svc-fc
```

This line tells Cinder what backends to use (you could have multiple backends if you have multiple types of arrays or multiple options for a given array type, for example). You use the same string to identify the driver. Next, append the text below to the end of cinder.conf, filling in the variables as noted:

```
[ibm-svc-fc]
volume_driver=cinder.volume.drivers.ibm.storwize_svc.StorwizeSVCDriver
san_ip=<the array’s IP address>
san_login=<the array’s administrative login>
san_password=<the array’s administrative password>
storwize_svc_volpool_name=ViPR_C_pool
volume_backend_name=IBMSVC-FC
storwize_svc_connection_protocol=FC
```

These driver parameters are typical of third-party storage, but each vendor defines their own set, so be sure to check the documentation. In this example, here’s what they do:

```
[ibm-svc-fc]
```

A name for Cinder to refer to the driver backend definition. You can use whatever text you want (keep it short and limited to alphanumeric and punctuation characters), but it must be unique per instance. The value here must be the same as the enabled_backends entry from above.

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

Provides the location of the Python driver code (/usr/lib/python2.7/site-packages/cinder/volume/drivers/ibm/storwize_svc).

```
san_ip=<the array’s IP address>
san_login=<the array’s administrative login>
san_password=<the array’s administrative password>
```

Enables Cinder to log into the array and perform its functions. You can configure passwordless SSH via key-pair authentication, but this example uses a simple password challenge.

```
storwize_svc_volpool_name=ViPR_C_pool
```

The pool on the IBM SVC itself from which Cinder will carve its volumes. This pool must exist prior to Cinder ingestion (the administrator builds it outside of Cinder) and represents the total storage space available to Cinder.

```
volume_backend_name=IBMSVC-FC
```

Cinder's reference to the volume backend; you use it when assigning a specific type to Cinder. This allows flexibility in multiple-driver configuration; for example, you could assign the same backend to service multiple driver instances. By default, ViPR Controller assumes that the host connection is via IP, so if you are using Fibre Channel, the backend or the Cinder driver name must contain the string "FC" so that ViPR Controller knows this array is Fibre Channel connected.

```
storwize_svc_connection_protocol=FC
```

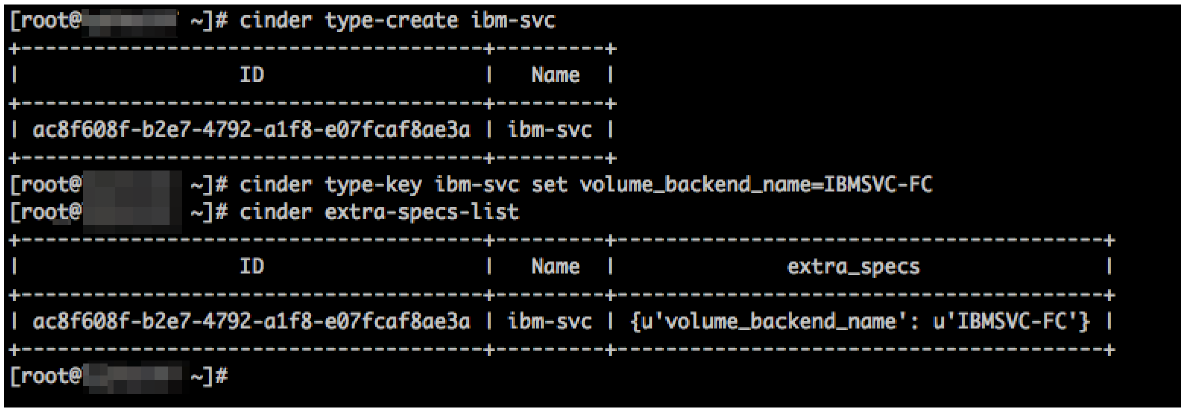
Configures the connection to the array from the Cinder driver itself.

Now that Cinder has the information needed to define the array, create the array in the Cinder database with the following commands. Note that the type name can be whatever the administrator wants, but the volume backend name must exactly match the string in the cinder.conf file. The type-key set command produces no output, so use the extra-specs-list command to verify the mapping:

```
# cinder type-create ibm-svc
# cinder type-key ibm-svc set volume_backend_name=IBMSVC-FC
# cinder extra-specs-list
```

Successful completion resembles the following (your GUID values will vary):

Figure 8. ASSOCIATING A DRIVER WITH A DATABASE ENTRY



```
[root@[redacted] ~]# cinder type-create ibm-svc
+--------------------------------------+---------+
|                  ID                  |   Name  |
+--------------------------------------+---------+
| ac8f608f-b2e7-4792-a1f8-e07fcaf8ae3a | ibm-svc |
+--------------------------------------+---------+
[root@[redacted] ~]# cinder type-key ibm-svc set volume_backend_name=IBMSVC-FC
[root@[redacted] ~]# cinder extra-specs-list
+--------------------------------------+---------+--------------------------------------+
|                  ID                  |   Name  |              extra_specs             |
+--------------------------------------+---------+--------------------------------------+
| ac8f608f-b2e7-4792-a1f8-e07fcaf8ae3a | ibm-svc | {u'volume_backend_name': u'IBMSVC-FC'} |
+--------------------------------------+---------+--------------------------------------+
[root@[redacted] ~]#
```

Restart the Cinder services:

```
# systemctl restart openstack-cinder-api openstack-cinder-scheduler openstack-cinder-volume
```

At this point, it's important to understand that the Cinder host has no storage connections to the array – and needs none. All of the communications are via IP to the control interfaces. In fact, our IBM SVC has Fibre-Channel connections, and the Cinder host we are using has no FC cards installed.

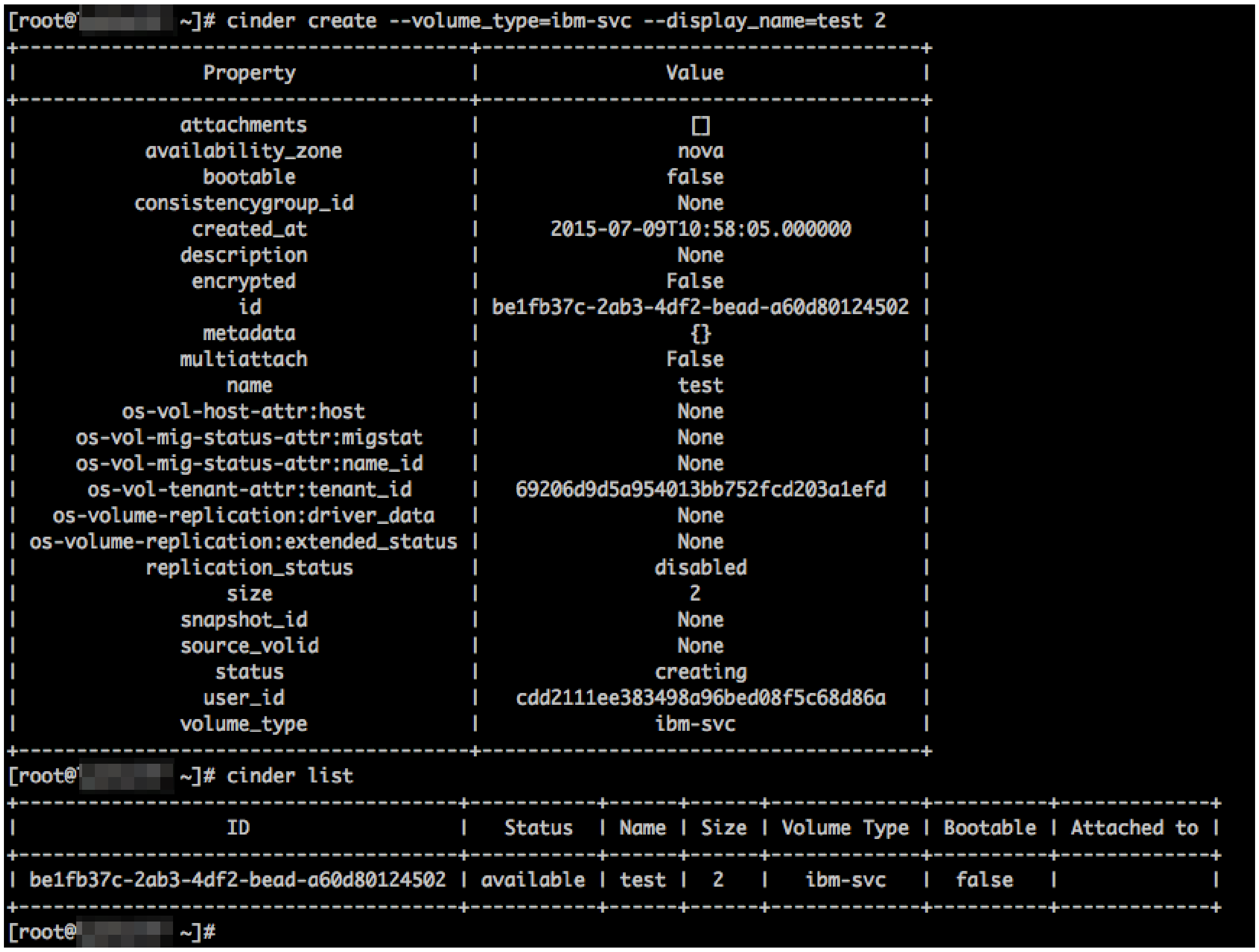
However, it's critical to completely verify Cinder operations before you attempt to integrate Cinder with ViPR Controller. If Cinder isn't working properly, then you need to straighten out any problems; ViPR Controller won't be able to cure Cinder issues. So even though there's no storage connection from the Cinder host to the array, we can still create and delete volumes via the control interface.

To create a volume, use the "cinder create" command. The volume_type parameter matches your type-create entry, and the display_name is just for identification purposes. The final digit is the size of the volume in GB:

```
# cinder create --volume_type=ibm-svc --display_name=test 2
```

Use the cinder list command to verify creation; an "available" status indicates success. Output should resemble the following:

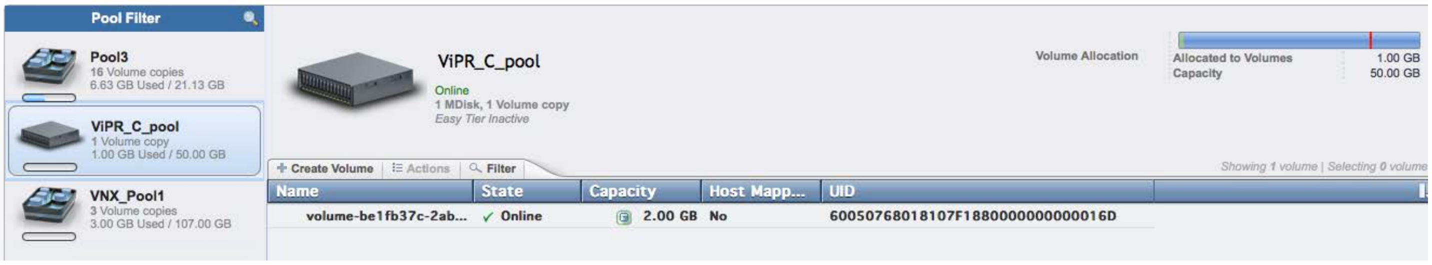
Figure 9. CREATING A TEST VOLUME



```
[root@[redacted] ~]# cinder create --volume_type=ibm-svc --display_name=test 2
+---------------------------------------+--------------------------------------+
|               Property                |                Value                 |
+---------------------------------------+--------------------------------------+
|              attachments              |                  []                  |
|           availability_zone           |                 nova                 |
|               bootable                |                false                 |
|         consistencygroup_id           |                 None                 |
|              created_at               |      2015-07-09T10:58:05.000000      |
|              description              |                 None                 |
|               encrypted               |                False                 |
|                  id                   | be1fb37c-2ab3-4df2-bead-a60d80124502 |
|               metadata                |                  {}                  |
|              multiattach              |                False                 |
|                 name                  |                 test                 |
|        os-vol-host-attr:host          |                 None                 |
|    os-vol-mig-status-attr:migstat     |                 None                 |
|    os-vol-mig-status-attr:name_id     |                 None                 |
|     os-vol-tenant-attr:tenant_id      |   69206d9d5a954013bb752fcd203a1efd   |
|   os-volume-replication:driver_data   |                 None                 |
| os-volume-replication:extended_status |                 None                 |
|          replication_status           |               disabled               |
|                 size                  |                  2                   |
|              snapshot_id              |                 None                 |
|              source_volid             |                 None                 |
|                status                 |               creating               |
|                user_id                |   cdd2111ee383498a96bed08f5c68d86a   |
|              volume_type              |               ibm-svc                |
+---------------------------------------+--------------------------------------+
[root@[redacted] ~]# cinder list
+--------------------------------------+-----------+------+------+-------------+----------+-------------+
|                  ID                  |   Status  | Name | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+------+------+-------------+----------+-------------+
| be1fb37c-2ab3-4df2-bead-a60d80124502 | available | test |  2   |   ibm-svc   |  false   |             |
+--------------------------------------+-----------+------+------+-------------+----------+-------------+
[root@[redacted] ~]#
```

Here's the volume displayed using the IBM SVC console. Note that the pool matches the definition in /etc/cinder.conf and that there are no host mappings.

Figure 10. VOLUME DISPLAY AT THE ARRAY



Pool Filter

Pool3
16 Volume copies
6.63 GB Used / 21.13 GB

ViPR_C_pool
1 Volume copy
1.00 GB Used / 50.00 GB

VNX_Pool1
3 Volume copies
3.00 GB Used / 107.00 GB

ViPR_C_pool
Online
1 MDisk, 1 Volume copy
Easy Tier Inactive

Volume Allocation

Allocated to Volumes	1.00 GB
Capacity	50.00 GB

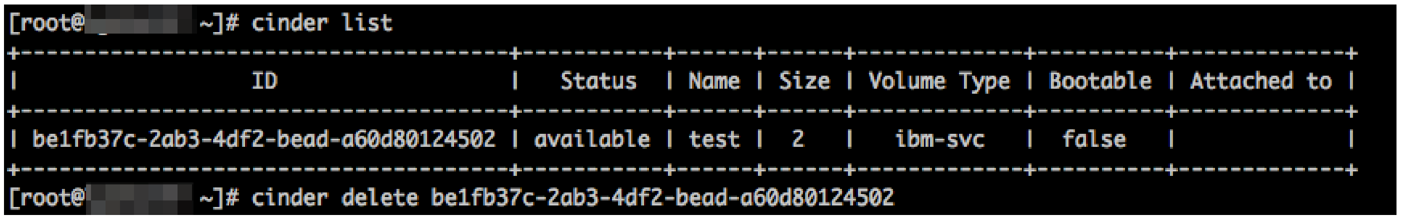
Showing 1 volume | Selecting 0 volume

Name	State	Capacity	Host Mapp...	UID
volume-be1fb37c-2ab...	✓ Online	2.00 GB	No	60050768018107F1880000000000016D

If you want to clean up the environment prior to ViPR Controller integration, use the “cinder delete” command with the GUID from “cinder list”:

```
# cinder delete be1fb37c-2ab3-4df2-bead-a60d80124502
```

Figure 11. DELETING THE TEST VOLUME

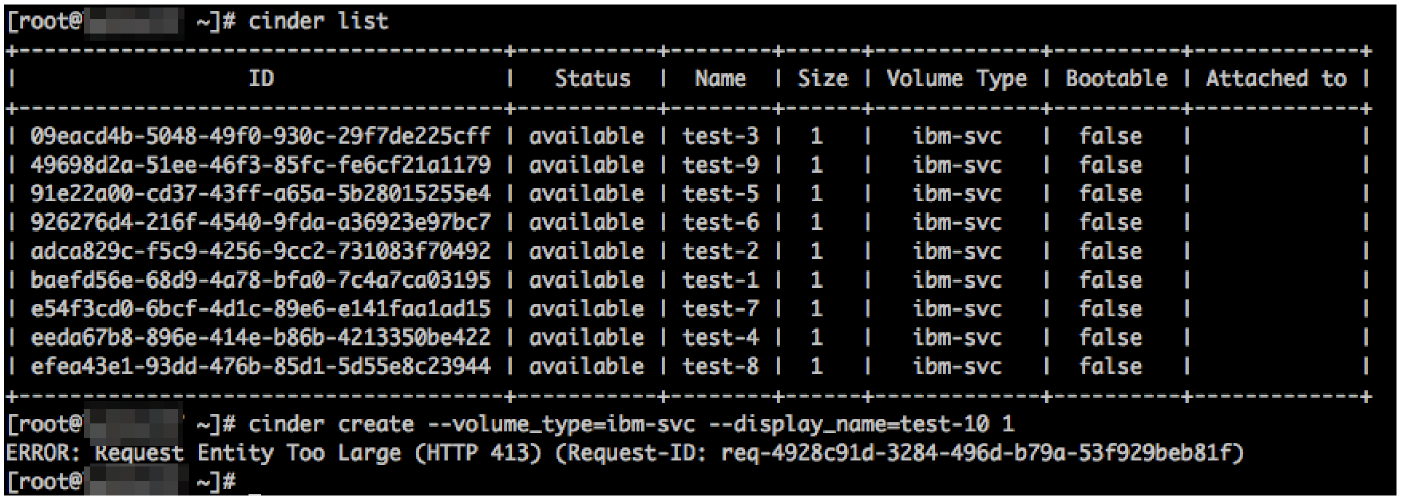


```
[root@[redacted] ~]# cinder list
+--------------------------------------+-----------+------+------+-------------+----------+-------------+
|                  ID                  |   Status  | Name | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+------+------+-------------+----------+-------------+
| be1fb37c-2ab3-4df2-bead-a60d80124502 | available | test |  2   |   ibm-svc   |  false   |             |
+--------------------------------------+-----------+------+------+-------------+----------+-------------+
[root@[redacted] ~]# cinder delete be1fb37c-2ab3-4df2-bead-a60d80124502
```

BEWARE THE QUOTA LIMIT!

There's some great installation and troubleshooting information at https://community.emc.com/docs/DOC-37248. 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 but after you build a few, you're likely to hit a problem as shown in Figure 12:

Figure 12. OVER THE QUOTA LIMIT



```
[root@[redacted] ~]# cinder list
+--------------------------------------+-----------+--------+------+-------------+----------+-------------+
|                  ID                  |   Status  |  Name  | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+--------+------+-------------+----------+-------------+
| 09eacd4b-5048-49f0-930c-29f7de225cff | available | test-3 |  1   |   ibm-svc   |  false   |             |
| 49698d2a-51ee-46f3-85fc-fe6cf21a1179 | available | test-9 |  1   |   ibm-svc   |  false   |             |
| 91e22a00-cd37-43ff-a65a-5b28015255e4 | available | test-5 |  1   |   ibm-svc   |  false   |             |
| 926276d4-216f-4540-9fda-a36923e97bc7 | available | test-6 |  1   |   ibm-svc   |  false   |             |
| adca829c-f5c9-4256-9cc2-731083f70492 | available | test-2 |  1   |   ibm-svc   |  false   |             |
| baefd56e-68d9-4a78-bfa0-7c4a7ca03195 | available | test-1 |  1   |   ibm-svc   |  false   |             |
| e54f3cd0-6bcf-4d1c-89e6-e141faa1ad15 | available | test-7 |  1   |   ibm-svc   |  false   |             |
| eeda67b8-896e-414e-b86b-4213350be422 | available | test-4 |  1   |   ibm-svc   |  false   |             |
| efea43e1-93dd-476b-85d1-5d55e8c23944 | available | test-8 |  1   |   ibm-svc   |  false   |             |
+--------------------------------------+-----------+--------+------+-------------+----------+-------------+
[root@[redacted] ~]# cinder create --volume_type=ibm-svc --display_name=test-10 1
ERROR: Request Entity Too Large (HTTP 413) (Request-ID: req-4928c91d-3284-496d-b79a-53f929beb81f)
[root@[redacted] ~]# _
```

If you do run into this limitation, check the information at the above link (and on the OpenStack web site) about resetting quotas. You'll need to use "openstack user list" to get the GUID of your admin user, then "cinder quota show" command to display the current settings.

INTEGRATE CINDER WITH VIPR CONTROLLER

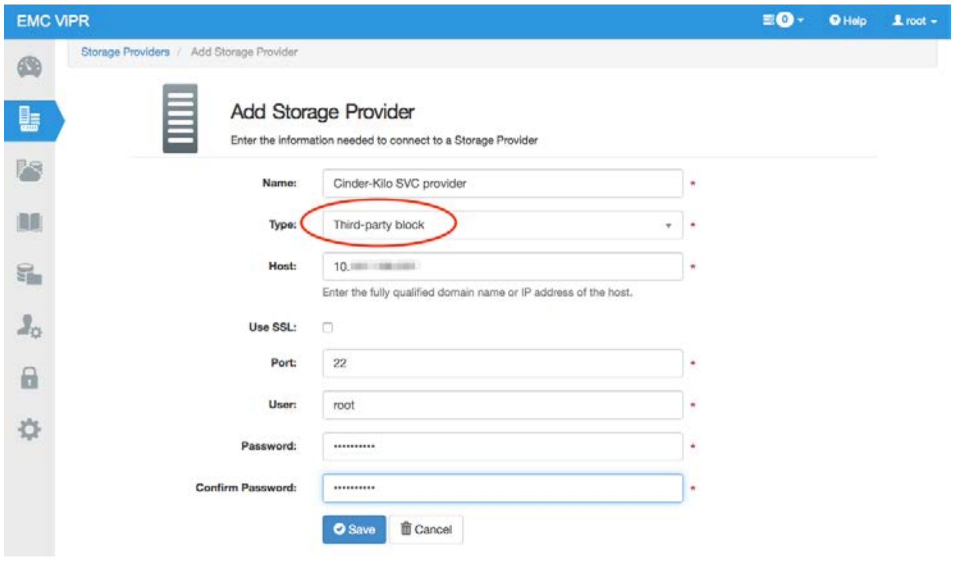
Before you can use ViPR Controller to provision storage to your hosts, you need to set up a few things so that the entities can communicate. This paper assumes you've already done the following:

- o Installed and licensed ViPR Controller
- o Discovered a Fabric Manager and at least one network
- o Verified that your target host and array Fibre Channel ports were discovered in or added to the network(s)
- o Verified link connectivity between your host and switch(es)

CONFIGURE A THIRD-PARTY PROVIDER

In ViPR Controller, under Tenant Settings | Projects, create a project ("Project 1") and then open Physical Assets | Storage Providers. Add the Cinder host using "Third-party block" as the type.

Figure 13. ADDING THE STORAGE PROVIDER



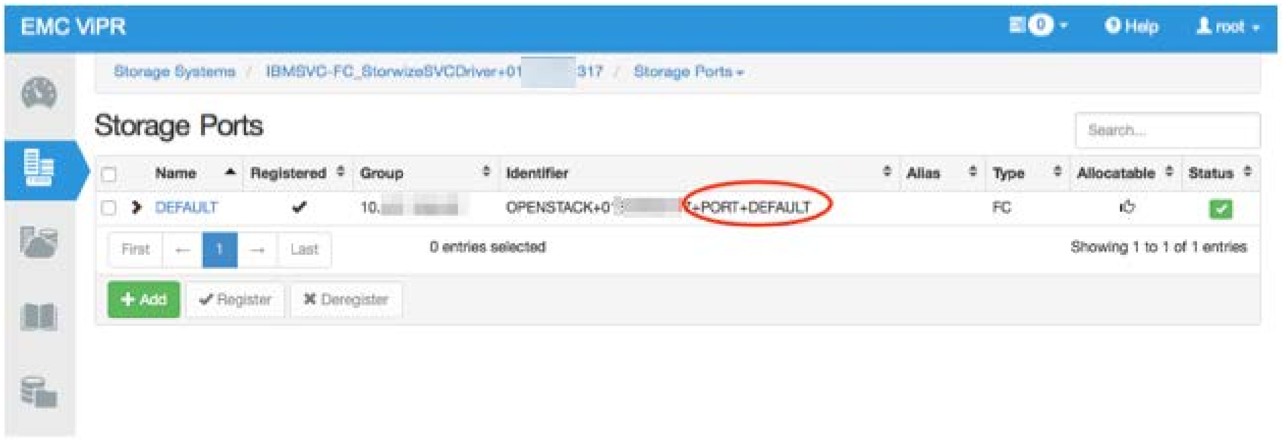
Add Storage Provider

Enter the information needed to connect to a Storage Provider

Name:	Cinder-Kilo SVC provider
Type:	Third-party block
Host:	10.[redacted]
	Enter the fully qualified domain name or IP address of the host.
Use SSL:	☐
Port:	22
User:	root
Password:	••••••••••
Confirm Password:	••••••••••

Once that’s added, you should see the target array added. But if you look under “Ports”, something interesting appears:

Figure 14. WHAT’S THIS DEFAULT PORT?


EMC ViPR
Help
root
Storage Systems / IBMSVC-FC_StorwizeSVCDriver+01[redacted]317 / Storage Ports
Search...

Storage Ports

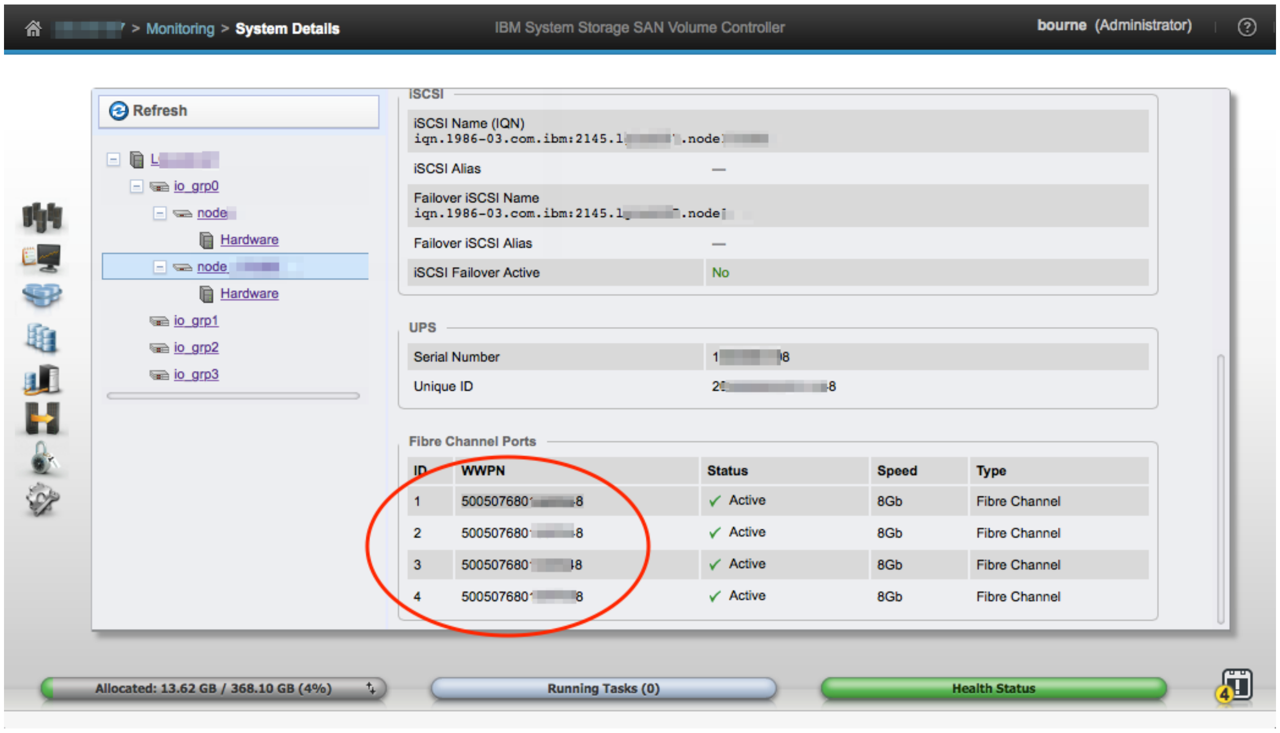
	Name	Registered	Group	Identifier	Alias	Type	Allocatable	Status
☐	DEFAULT	✓	10.[redacted]	OPENSTACK+0[redacted]+PORT+DEFAULT		FC		✓

First
1
Last
0 entries selected
Showing 1 to 1 of 1 entries
Add
Register
Deregister

Note here that the port type is “FC”. If you see “IP” here, that probably means you need to restart the Cinder services (or you didn’t put “FC” in the back-end name), so ViPR Controller doesn’t realize the port is in fact Fibre Channel.

The OpenStack API doesn’t provide the storage WWPN in a Fibre Channel connected setup, and so ViPR Controller isn’t going to be able to perform any export operations. In order to get the WWPN in there, you need to log into the ViPR Controller console and run a few commands, but first, you need a valid WWPN from the array. You can get that from the ViPR Controller network screen or from the array itself. The IBM SVC console provides the information in the Monitoring | System Details screen:

Figure 15. PORT DISPLAY AT THE IBM SVC CONSOLE



iSCSI

iSCSI Name (IQN) iqn.1986-03.com.ibm:2145.1[redacted].node[redacted]	
iSCSI Alias	—
Failover iSCSI Name iqn.1986-03.com.ibm:2145.1[redacted].node[redacted]	
Failover iSCSI Alias	—
iSCSI Failover Active	No

UPS

Serial Number	1[redacted]8
Unique ID	2[redacted]8

Fibre Channel Ports

ID	WWPN	Status	Speed	Type
1	500507680[redacted]8	✓ Active	8Gb	Fibre Channel
2	500507680[redacted]8	✓ Active	8Gb	Fibre Channel
3	500507680[redacted]8	✓ Active	8Gb	Fibre Channel
4	500507680[redacted]8	✓ Active	8Gb	Fibre Channel

Select one of the PWWNs that you know is attached to your network, then SSH into the ViPR Controller host console and run the following:

```
# cd /opt/storageos/cli/bin
# ./viprcli authenticate –hostname <your-ViPR-host-IP> -u root –d /tmp
# ./viprcli -hostname <your-ViPR-host-IP> storagesystem list
```

This command shows the attached storage systems. You're interested in the last three digits of the serial number for the array you want to connect (in the example screenshot below, it's "317", so we'll use that here).

```
# ./viprcli -hostname <your-ViPR-host-IP> storageport list -t openstack -sn 317
```

Feed those three digits into the storageport command (-t is the type, here "openstack" and –sn is the serial number). The output shows you the port network ID, which is the same as the identifier in the ViPR Controller display. That's the value you're trying to change – note the "DEFAULT" here.

```
# ./viprcli -hostname <your-ViPR-host-IP> storageport update -t openstack -sn 317 -tt FC -pn
DEFAULT -pnwid "your-PWWN"
```

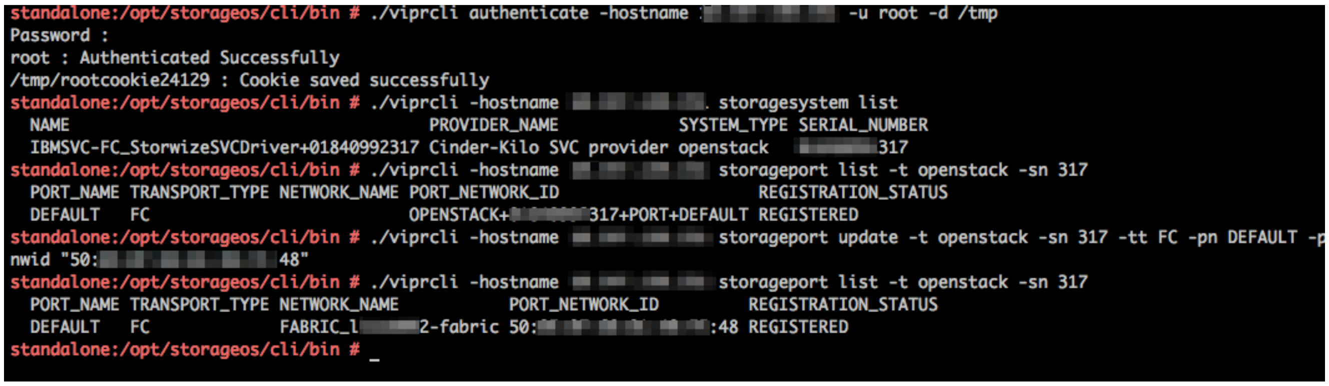
Update the storage port with the new data (-tt is transport type, here FC for Fibre Channel, -pn is the entry you're updating, here known as "DEFAULT", and the new WWN).

```
# ./viprcli -hostname <your-ViPR-host-IP> storageport list -t openstack -sn 317
```

Running the storageport command again reveals that the valid WWPN you provided is now listed under PORT_NETWORK_ID.

The example output below shows the relationship among these commands:

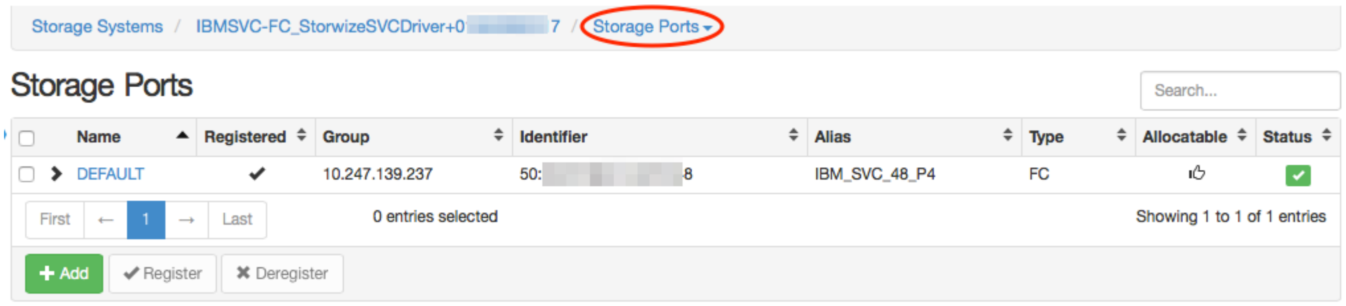
Figure 16. VIPRCLI COMMANDS TO ADD THE WWPN



```
standalone:/opt/storageos/cli/bin # ./viprcli authenticate -hostname [redacted] -u root -d /tmp
Password :
root : Authenticated Successfully
/tmp/rootcookie24129 : Cookie saved successfully
standalone:/opt/storageos/cli/bin # ./viprcli -hostname [redacted]. storagesystem list
  NAME                                    PROVIDER_NAME            SYSTEM_TYPE SERIAL_NUMBER
  IBMSVC-FC_StorwizeSVCDriver+01840992317 Cinder-Kilo SVC provider openstack   [redacted]317
standalone:/opt/storageos/cli/bin # ./viprcli -hostname [redacted] storageport list -t openstack -sn 317
  PORT_NAME TRANSPORT_TYPE NETWORK_NAME PORT_NETWORK_ID                       REGISTRATION_STATUS
  DEFAULT   FC                          OPENSTACK+[redacted]317+PORT+DEFAULT REGISTERED
standalone:/opt/storageos/cli/bin # ./viprcli -hostname [redacted] storageport update -t openstack -sn 317 -tt FC -pn DEFAULT -p
nwid "50:[redacted]48"
standalone:/opt/storageos/cli/bin # ./viprcli -hostname [redacted] storageport list -t openstack -sn 317
  PORT_NAME TRANSPORT_TYPE NETWORK_NAME              PORT_NETWORK_ID        REGISTRATION_STATUS
  DEFAULT   FC             FABRIC_l[redacted]2-fabric 50:[redacted]:48 REGISTERED
standalone:/opt/storageos/cli/bin # _
```

The ViPR Controller console also shows the new WWPN:

Figure 17. THERE'S THE WWPN!



Storage Systems / IBMSVC-FC_StorwizeSVCDriver+0[redacted]7 / Storage Ports

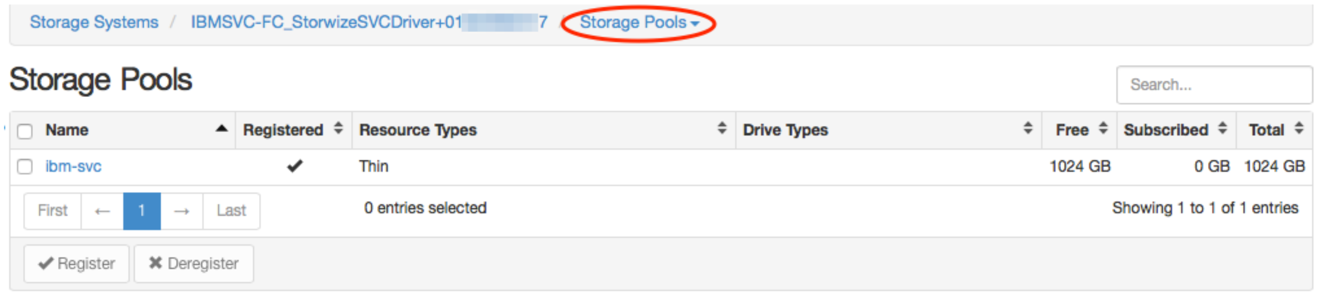
Storage Ports

	Name	Registered	Group	Identifier	Alias	Type	Allocatable	Status
☐	DEFAULT	✓	10.247.139.237	50:[redacted]8	IBM_SVC_48_P4	FC	👍	✓

0 entries selected — Showing 1 to 1 of 1 entries

You should also see the storage pool(s) from your array:

Figure 18. AND STORAGE, TOO…



Storage Systems / IBMSVC-FC_StorwizeSVCDriver+01[redacted]7 / Storage Pools

Storage Pools

	Name	Registered	Resource Types	Drive Types	Free	Subscribed	Total
☐	ibm-svc	✓	Thin		1024 GB	0 GB	1024 GB

0 entries selected — Showing 1 to 1 of 1 entries

Wait a second. Where did that 1024 GB figure 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.

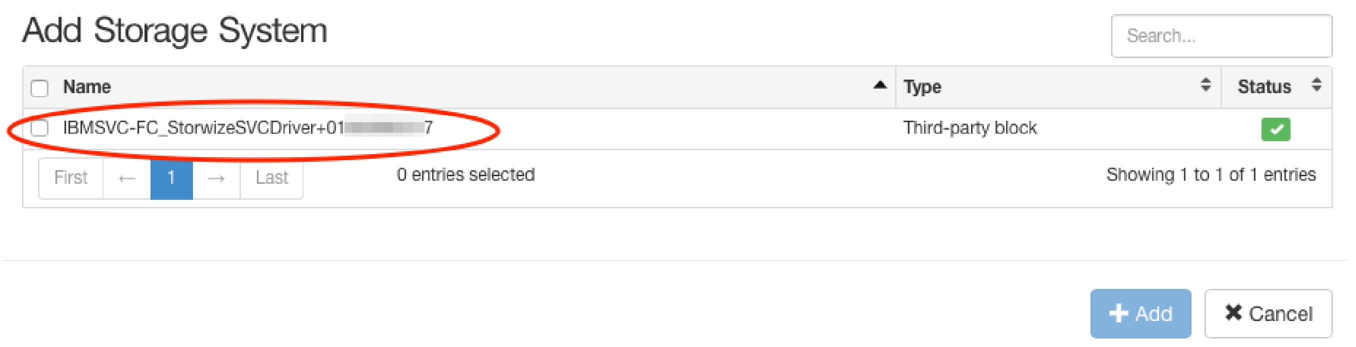
Now that ViPR Controller knows about your array and host, you can build your virtual array and virtual block storage pool.

BUILD A VIRTUAL ARRAY

Best practice is to create a separate virtual array for Cinder use. Although it is possible to combine Cinder into a virtual array with other storage, odd results may occur.

In ViPR Controller, under Virtual Assets | Virtual Arrays, create a new virtual array to serve out the third-party storage. Select "Add Storage System" and check the IBM SVC driver from your array (if multipath is to be used, make sure all of your storage ports show up correctly; our example only uses one port at the IBM SVC array, which is definitely not recommended in a production environment):

Figure 19. ADDING THE STORAGE SYSTEM



Add Storage System

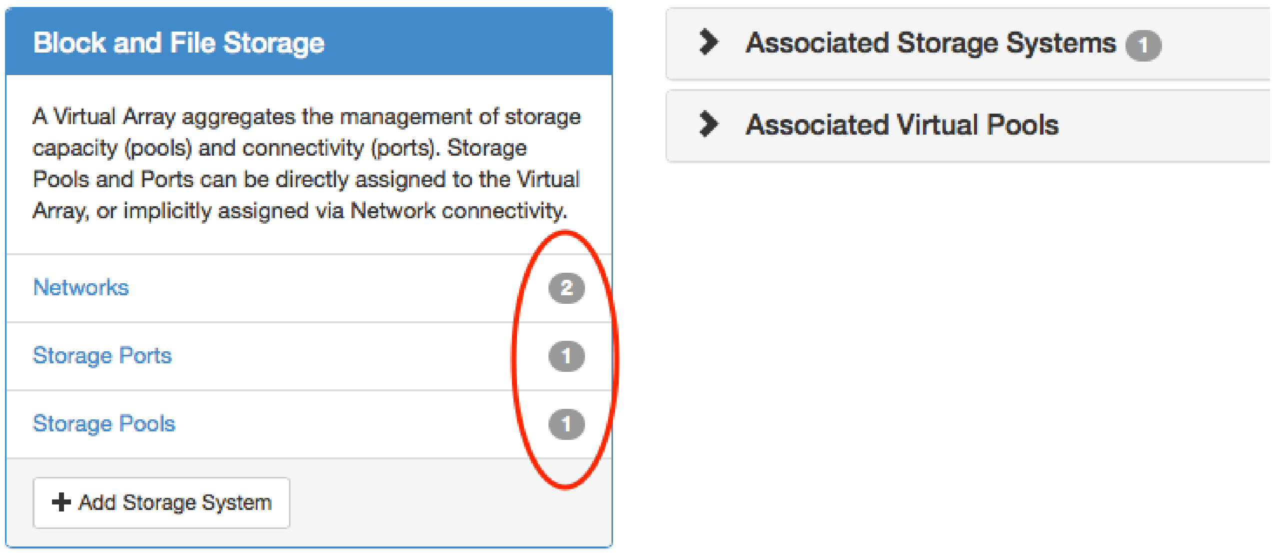
Search...

Name	Type	Status
IBMSVC-FC_StorwizeSVCDriver+01[redacted]7	Third-party block	✓

First ← 1 → Last   0 entries selected   Showing 1 to 1 of 1 entries

You should see the port(s) and pool(s) associated with your array:

Figure 20. VERIFY THE PORT AND POOL COUNTS



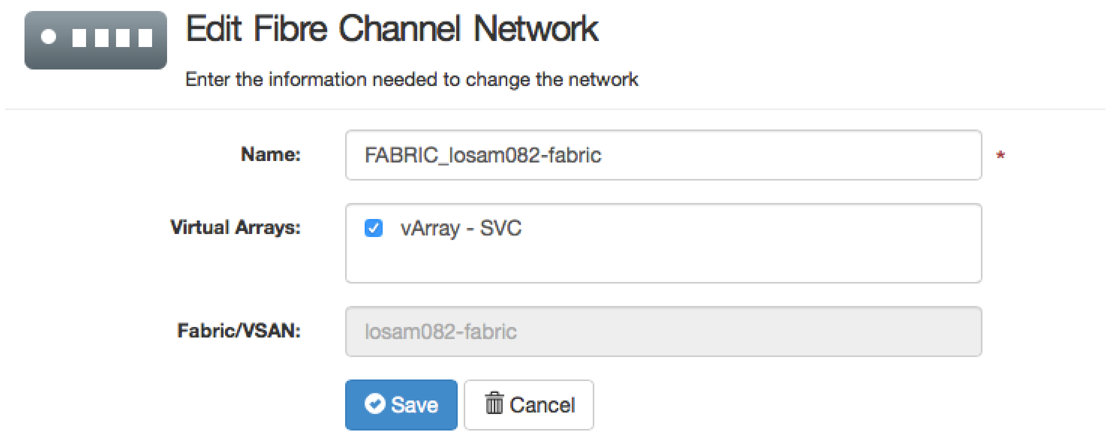
Block and File Storage

A Virtual Array aggregates the management of storage capacity (pools) and connectivity (ports). Storage Pools and Ports can be directly assigned to the Virtual Array, or implicitly assigned via Network connectivity.

Networks	2
Storage Ports	1
Storage Pools	1

Next, under Physical Assets | Networks, open your Fibre Channel network (the target host and array should already be there), check the box for your virtual array, and save the changes:

Figure 21. ADD THE VIRTUAL ARRAY TO THE NETWORK


Edit Fibre Channel Network

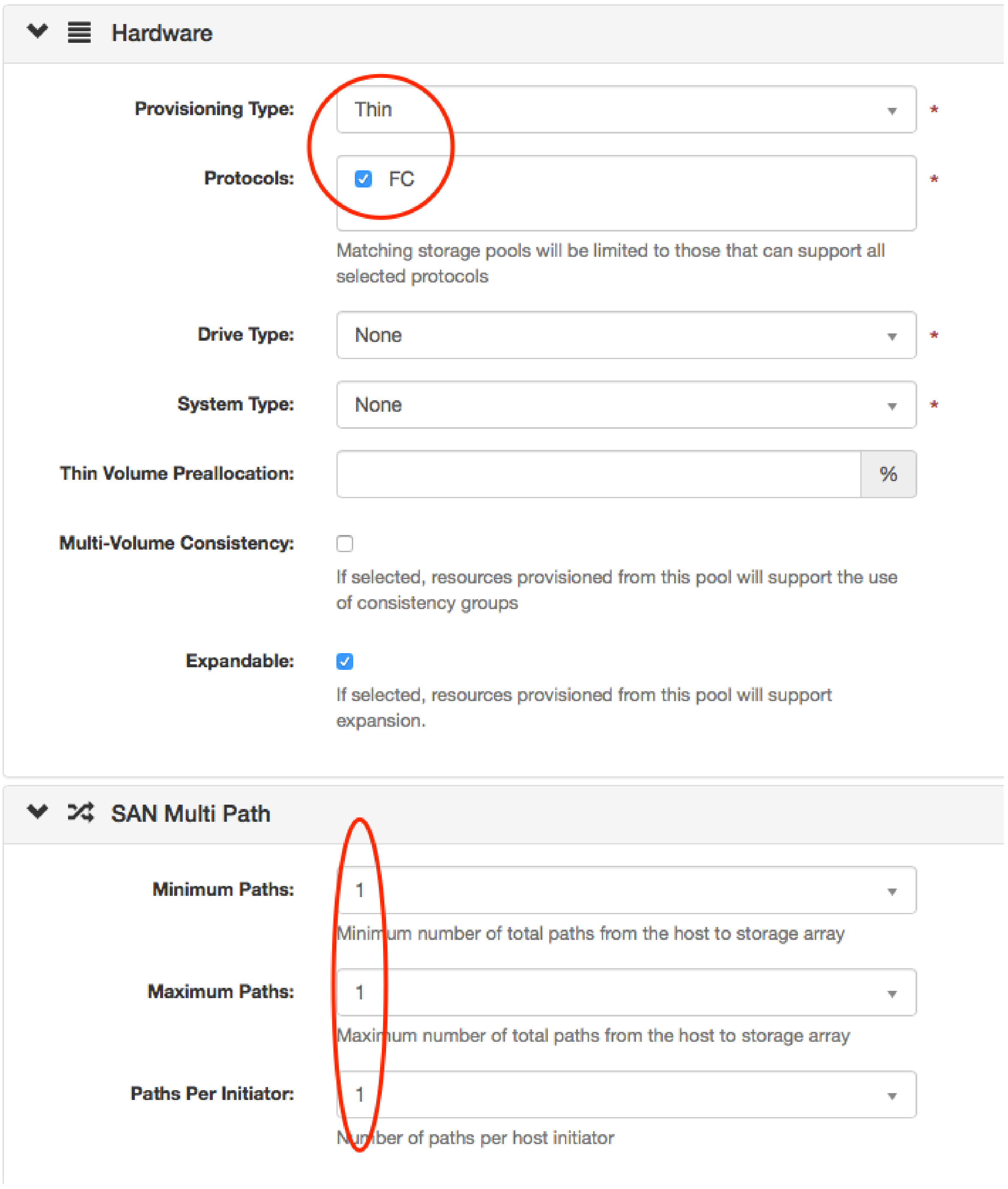
Enter the information needed to change the network

Name:	FABRIC_losam082-fabric *
Virtual Arrays:	✓ vArray - SVC
Fabric/VSAN:	losam082-fabric

BUILD A VIRTUAL POOL

The final configuration step from the ViPR Controller side is to build a virtual pool from your Cinder-backed storage. Wait a few minutes for ViPR Controller to rescan, and then create a block virtual pool using the new array. If you plan to use snapshots, then make sure you configure the snapshot settings in the pool. Critical settings include making it a thin pool and setting multipath correctly; if you have more than one port, then the virtual array should have been configured with all of the available ports. In this example we use 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 22. VIRTUAL POOL SETTINGS FOR CINDER USE


Hardware

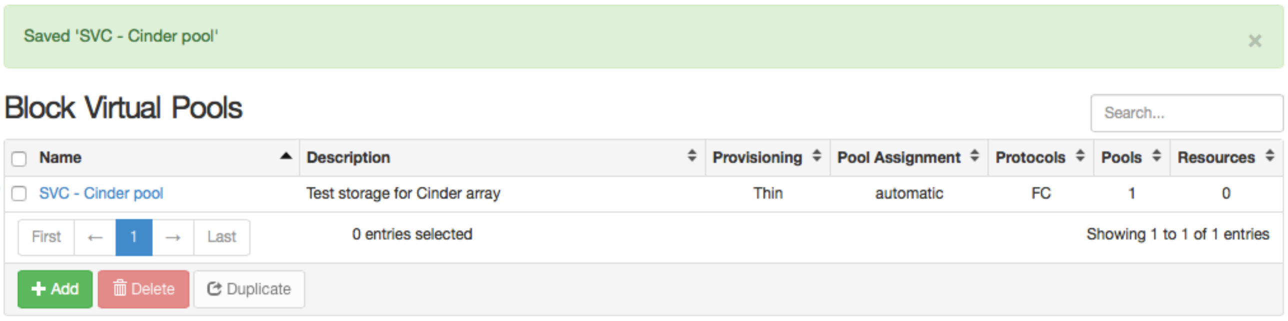
Provisioning Type:	Thin *
Protocols:	☑ FC *
	Matching storage pools will be limited to those that can support all selected protocols
Drive Type:	None *
System Type:	None *
Thin Volume Preallocation:	%
Multi-Volume Consistency:	☐
	If selected, resources provisioned from this pool will support the use of consistency groups
Expandable:	☑
	If selected, resources provisioned from this pool will support expansion.

SAN Multi Path

Minimum Paths:	1
	Minimum number of total paths from the host to storage array
Maximum Paths:	1
	Maximum number of total paths from the host to storage array
Paths Per Initiator:	1
	Number of paths per host initiator

Your virtual pool appears as shown below:

Figure 23. VERIFYING THE VIRTUAL POOL


Saved 'SVC - Cinder pool'

Block Virtual Pools

Name	Description	Provisioning	Pool Assignment	Protocols	Pools	Resources
SVC - Cinder pool	Test storage for Cinder array	Thin	automatic	FC	1	0

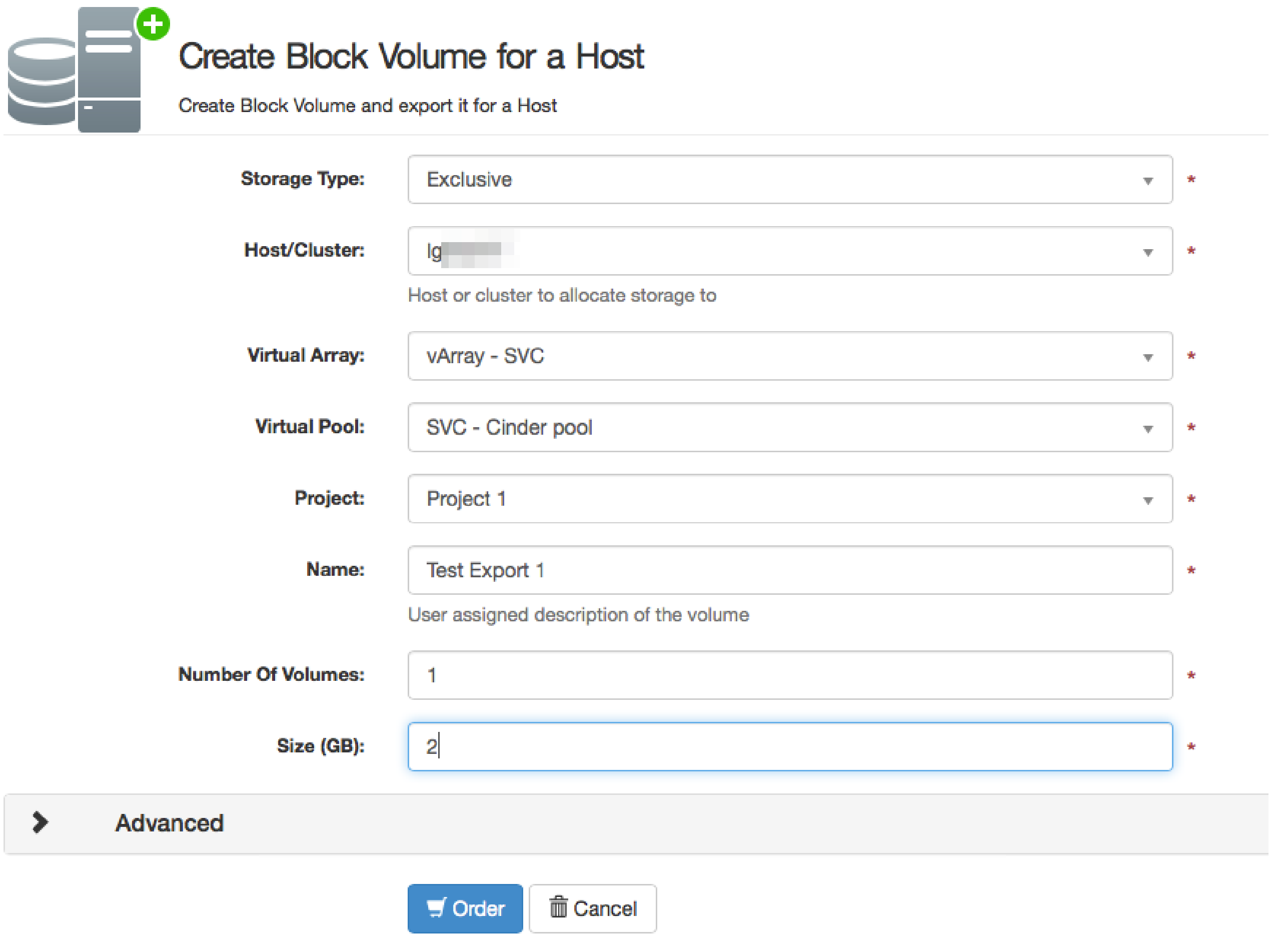
0 entries selected

Showing 1 to 1 of 1 entries

TEST HOST PROVISIONING

Now that the ViPR Controller configuration is complete, it's time to actually provision some storage to a host. In the Service Catalog, select Block Storage Services | Create Block volume for a Host, then fill in the details.

Figure 24. TESTING THE INTEGRATION SETUP

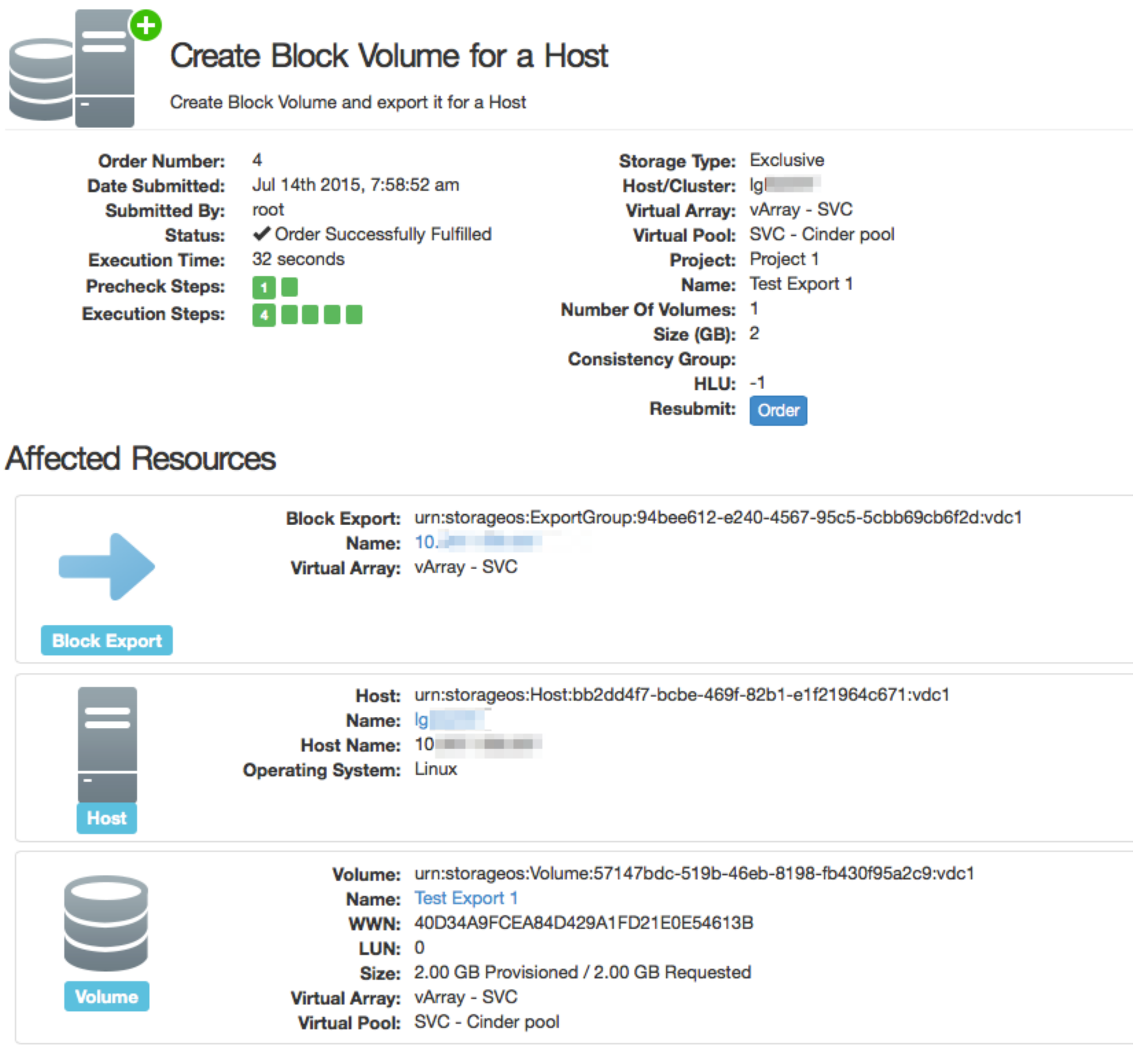

Create Block Volume for a Host

Create Block Volume and export it for a Host

Storage Type:	Exclusive *
Host/Cluster:	lg[illegible] *
	Host or cluster to allocate storage to
Virtual Array:	vArray - SVC *
Virtual Pool:	SVC - Cinder pool *
Project:	Project 1 *
Name:	Test Export 1 *
	User assigned description of the volume
Number Of Volumes:	1 *
Size (GB):	2 *

Successful completion of the order looks like this:

Figure 25. SUCCESSFUL PROVISIONING



Create Block Volume for a Host

Create Block Volume and export it for a Host

Order Number:	4	**Storage Type:**	Exclusive
Date Submitted:	Jul 14th 2015, 7:58:52 am	**Host/Cluster:**	lg[redacted]
Submitted By:	root	**Virtual Array:**	vArray - SVC
Status:	✓ Order Successfully Fulfilled	**Virtual Pool:**	SVC - Cinder pool
Execution Time:	32 seconds	**Project:**	Project 1
Precheck Steps:	1 🟩	**Name:**	Test Export 1
Execution Steps:	4 🟩🟩🟩🟩	**Number Of Volumes:**	1
		Size (GB):	2
		Consistency Group:	
		HLU:	-1
		Resubmit:	Order

Affected Resources

Block Export

Block Export:	urn:storageos:ExportGroup:94bee612-e240-4567-95c5-5cbb69cb6f2d:vdc1
Name:	10.[redacted]
Virtual Array:	vArray - SVC

Host

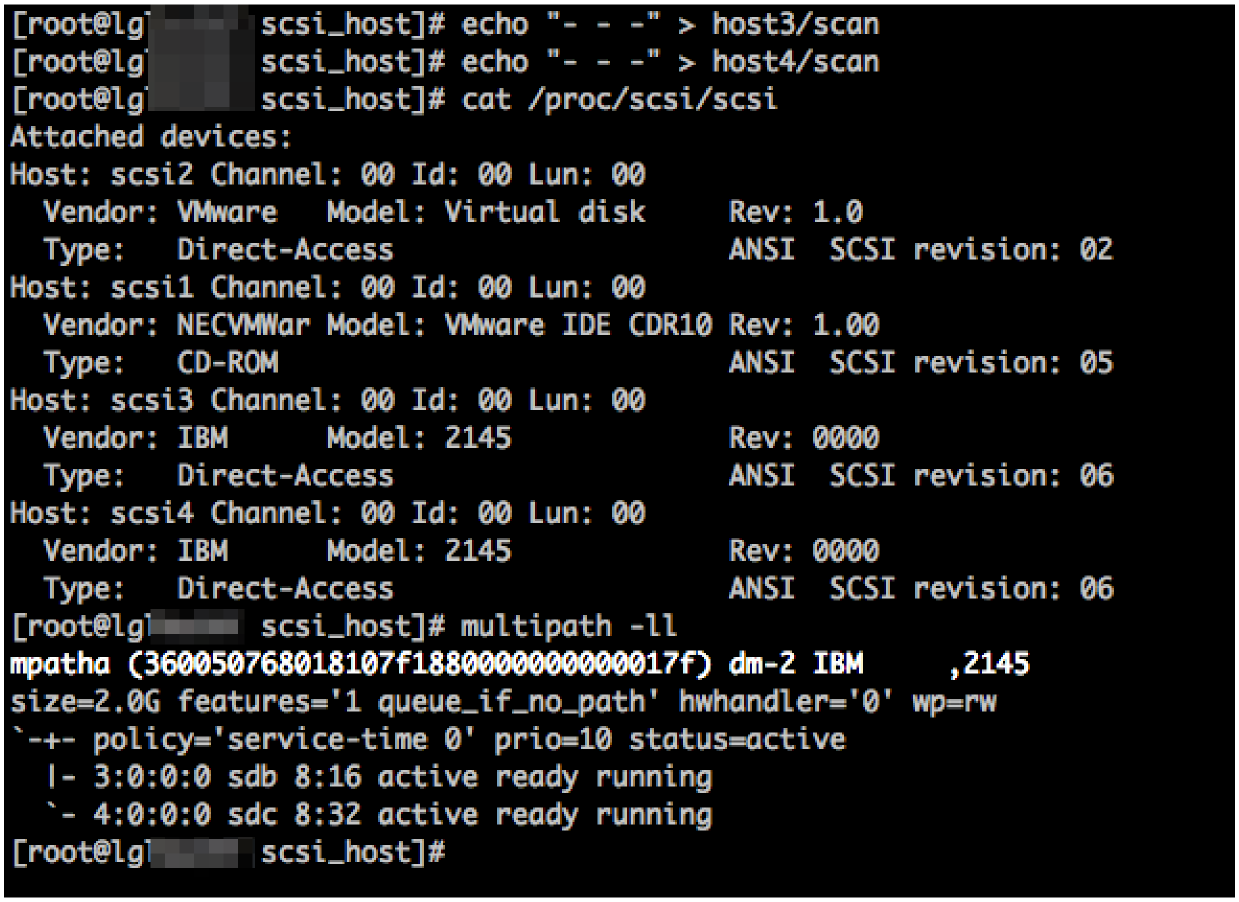
Host:	urn:storageos:Host:bb2dd4f7-bcbe-469f-82b1-e1f21964c671:vdc1
Name:	lg[redacted]
Host Name:	10[redacted]
Operating System:	Linux

Volume

Volume:	urn:storageos:Volume:57147bdc-519b-46eb-8198-fb430f95a2c9:vdc1
Name:	Test Export 1
WWN:	40D34A9FCEA84D429A1FD21E0E54613B
LUN:	0
Size:	2.00 GB Provisioned / 2.00 GB Requested
Virtual Array:	vArray - SVC
Virtual Pool:	SVC - Cinder pool

After rescanning the host’s HBAs (echo “- - -“ > /sys/class/scsi_host/hostN/scan) the host sees the storage as well:

Figure 26. STORAGE FROM THE HOST PERSPECTIVE



```
[root@lg[redacted] scsi_host]# echo "- - -" > host3/scan
[root@lg[redacted] scsi_host]# echo "- - -" > host4/scan
[root@lg[redacted] scsi_host]# cat /proc/scsi/scsi
Attached devices:
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: VMware   Model: Virtual disk     Rev: 1.0
  Type:   Direct-Access                    ANSI  SCSI revision: 02
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: NECVMWar Model: VMware IDE CDR10 Rev: 1.00
  Type:   CD-ROM                           ANSI  SCSI revision: 05
Host: scsi3 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM      Model: 2145             Rev: 0000
  Type:   Direct-Access                    ANSI  SCSI revision: 06
Host: scsi4 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM      Model: 2145             Rev: 0000
  Type:   Direct-Access                    ANSI  SCSI revision: 06
[root@lg[redacted] scsi_host]# multipath -ll
mpatha (360050768018107f1880000000000017f) dm-2 IBM     ,2145
size=2.0G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=10 status=active
  |- 3:0:0:0 sdb 8:16 active ready running
  `- 4:0:0:0 sdc 8:32 active ready running
[root@lg[redacted] scsi_host]#
```

Note that there's two paths to the array according to multipath, but there was only one configured in ViPR Controller. Why? In our setup, both HBAs are zoned to the same back-end IBM SVC port; this provides redundancy between the host and the local switch, but doesn't protect against a failure at the array port. ViPR Controller only knows about one array port and thus configured its view of the array for only a single path. In a production instance you would want to use at least two ports from the array and make sure that your virtual array sees all of the available paths, then set your virtual pool to use them.

CONCLUSION

This document has demonstrated how to build a single-host Cinder platform for ViPR Controller using Red Hat Linux 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. Also, internal reviewers on the ViPR Controller field, development, and engineering teams provided invaluable feedback.

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.