Bright Cluster Manager 8.1

OpenStack Deployment Manual

Revision: 8511be6

Date: Tue May 20 2025

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of Novell, Inc. PGI is a registered trademark of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, PBS Pro, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. Bright Computing, Inc. shall not be liable for technical or editorial errors or omissions which may occur in this document. Bright Computing, Inc. shall not be liable for any damages resulting from the use of this document.

Limitation of Liability and Damages Pertaining to Bright Computing, Inc.

The Bright Cluster Manager product principally consists of free software that is licensed by the Linux authors free of charge. Bright Computing, Inc. shall have no liability nor will Bright Computing, Inc. provide any warranty for the Bright Cluster Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the Bright Cluster Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the Bright Cluster Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

Table of Contents

Preface

Welcome to the *OpenStack Deployment Manual* for Bright Cluster Manager 8.1.

0.1 About This Manual

This manual is aimed at helping cluster administrators install, understand, configure, and manage basic OpenStack capabilities easily using Bright Cluster Manager. The administrator is expected to be reasonably familiar with the *Administrator Manual*.

0.2 About The Manuals In General

Regularly updated versions of the Bright Cluster Manager 8.1 manuals are available on updated clusters by default at /cm/shared/docs/cm. The latest updates are always online at http://support.brightcomputing.com/manuals.

- The *Installation Manual* describes installation procedures for a basic cluster.
- The *Administrator Manual* describes the general management of the cluster.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to program with Bright Cluster Manager.
- The *OpenStack Deployment Manual* describes how to deploy OpenStack with Bright Cluster Manager.
- The *Big Data Deployment Manual* describes how to deploy Big Data with Bright Cluster Manager.
- The *Machine Learning Manual* describes how to install and configure machine learning capabilities with Bright Cluster Manager.

If the manuals are downloaded and kept in one local directory, then in most pdf viewers, clicking on a cross-reference in one manual that refers to a section in another manual opens and displays that section in the second manual. Navigating back and forth between documents is usually possible with keystrokes or mouse clicks.

For example: <Alt>-<Backarrow> in Acrobat Reader, or clicking on the bottom leftmost navigation button of xpdf, both navigate back to the previous document.

The manuals constantly evolve to keep up with the development of the Bright Cluster Manager environment and the addition of new hardware and/or applications. The manuals also regularly incorporate customer feedback. Administrator and user input is greatly valued at Bright Computing. So any comments, suggestions or corrections will be very gratefully accepted at manuals@brightcomputing.com.

There is also a feedback form available via Bright View, via the Account icon, , following the clickpath:

Account→Help→Feedback

0.3 Getting Administrator-Level Support

If the reseller from whom Bright Cluster Manager was bought offers direct support, then the reseller should be contacted.

Otherwise the primary means of support is via the website `https://support.brightcomputing.com`. This allows the administrator to submit a support request via a web form, and opens up a trouble ticket. It is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket goes via standard e-mail. Section 13.2 of the *Administrator Manual* has more details on working with support.

0.4 Getting Professional Services

Bright Computing normally differentiates between professional services (customer asks Bright Computing to do something or asks Bright Computing to provide some service) and support (customer has a question or problem that requires an answer or resolution). Professional services can be provided after consulting with the reseller, or the Bright account manager.

1

Quickstart Installation Guide For OpenStack

This quickstart chapter describes, step-by-step, a basic and quick installation of OpenStack for Bright Cluster Manager on a cluster that is already running Bright Cluster Manager. Unlike in the main installation chapter (Chapter 3), the quickstart gives very little explanation of the steps, and is more of a recipe approach. Following these steps should allow a moderately experienced cluster administrator to get an operational OpenStack cluster up and running in a fairly standard configuration as quickly as possible. This would be without even having to read the introductory Chapter 2 of this manual, let alone any of the rest of the manual.

In this quickstart Chapter 1, the sections 1.1-1.3 are about what needs to be done to quickly get OpenStack up. The last section of this chapter, section 1.4 then covers tasks to check OpenStack-related functions of the cluster are working as expected.

1.1 Hardware Specifications

The hardware specifications suggested in this quickstart are a minimum configuration. Less powerful hardware is not guaranteed to work with Bright OpenStack.

The minimum number of nodes required to create an OpenStack cluster is 3:

- one head node
- one controller/network node
- and one hypervisor node.

Page 17 has a more extensive explanation of the required number of nodes.

The minimal hardware specifications for these node types are indicated by the following table:

Node Type	CPUs	RAM/GB	Hard Drive/GB	NICs
Head	4	8	40	2 *
Controller	4	8	80	2 *
Hypervisor	4	8	80	1 **

* 2 NICs, one of them connected to the switch where the other compute nodes will be connected and the other is connected to the external world through which it can access the Internet.

** 1 NIC connected to the switch where the other compute nodes will be connected.

1.2 Prerequisites

The starting point of the quickstart installation for Bright OpenStack requires an up and running Bright Cluster Manager. A quickstart on how to set up Bright Cluster Manager is given in Chapter 1 of the *Installation Manual* (`http://support.brightcomputing.com/manuals/8.1/installation-manual.pdf`)

Bright OpenStack is supported for RHEL7 and derivatives only.

The head node must have access to the base distribution repositories and to the Bright repositories. This is because `cm-openstack-setup`—a utility used in section 1.3—must be able to install packages from these repositories. The head node must therefore be connected to the internet, or it must be able to access a local mirror of both repositories.

1.3 Installing Bright OpenStack Using `cm-openstack-setup`

The `cm-openstack-setup` script is run from the head node and deploys an OpenStack instance. An example session is shown next. This example is based on using `node001` as the controller node, and `node002` as the hypervisor node:

```
[root@bright81 ~]# cm-openstack-setup
Please wait
Connecting to CMDaemon
```

If all is well, then a deployment screen is seen. The steps are then:

1. Select the `Deploy` option from the deployment screen (figure 1.1):

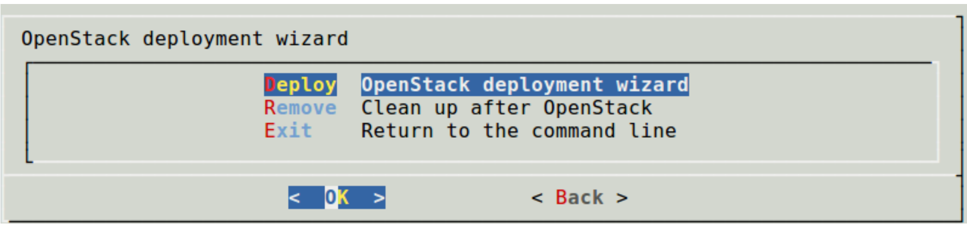

Figure 1.1: Deployment Screen

2. Select `node001` as the controller node.(figure 1.2):

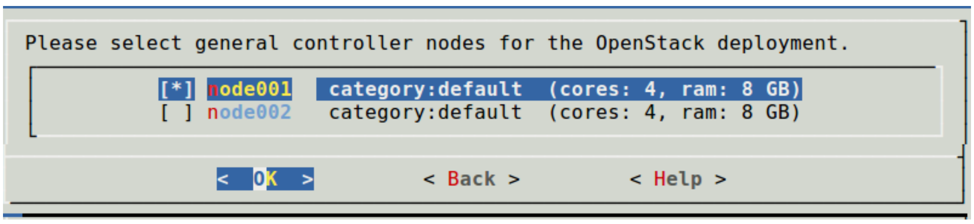

Figure 1.2: Setting the controller nodes

3. Set a password for the `admin` user (figure 1.3):

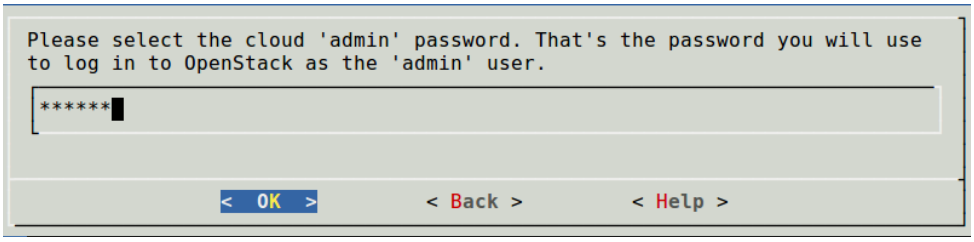

Figure 1.3: Setting The `admin` Password

The `admin` user is an OpenStack user who is to be created and who is to be given administrator privileges in the OpenStack instance that is being created by the wizard. The `admin` user can login to OpenStack Horizon (an administrative dashboard) when OpenStack is running.

4. Set OpenStack users to be stored in Keystone's MySQL (figure 1.4):

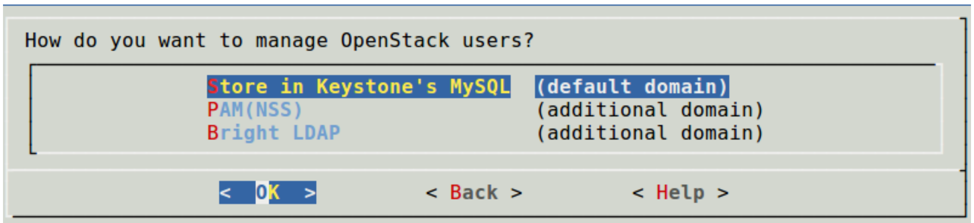

Figure 1.4: Configuring OpenStack users to be stored within Keystone's MySQL database

5. Set `/cm/shared` for Glance (images) storage (figure 1.5):

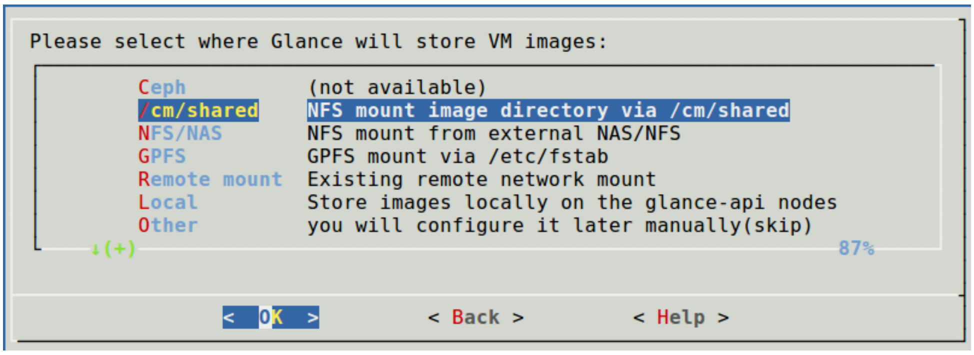

Figure 1.5: Configuring Glance (image) storage

6. Set NFS for Cinder (volume) storage (figure 1.6):

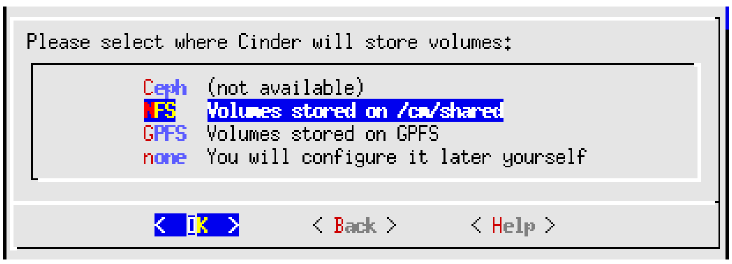

Figure 1.6: Configuring Cinder (volume) storage

7. Select `node002` as the hypervisor node (figure 1.7):

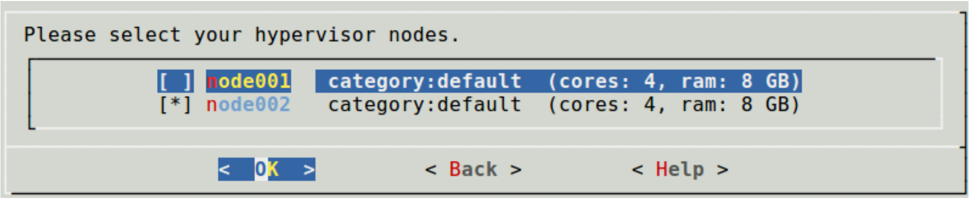

Figure 1.7: Configuring the hypervisor nodes

8. Set `/cm/shared` for Nova (virtual machines) storage (figure 1.8):

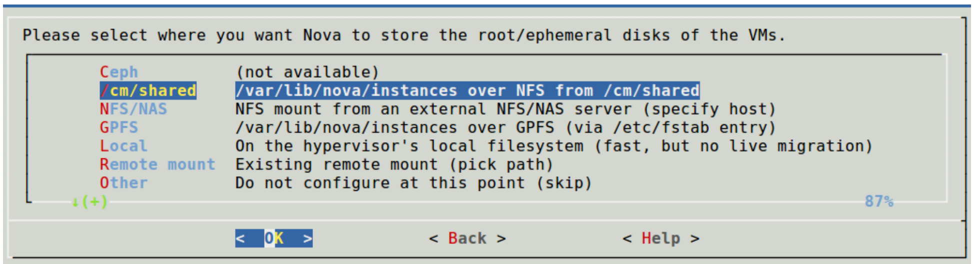

Figure 1.8: Configuring the Nova virtual machine disk storage

9. Set VXLAN as the network overlay technology (figure 1.9):

Figure 1.9: Setting VXLAN as the network overlay technology

10. Select the `<Create new>` option to create a new network for virtual networks in the OpenStack cluster (figure 1.10):

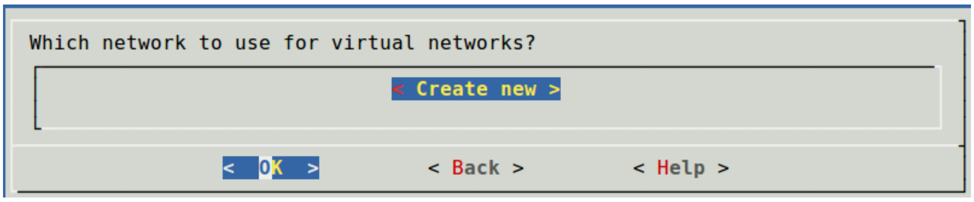

Figure 1.10: Configuring the creation of a new network for virtual networks

The default values for the new network can be accepted.

11. The OpenStack controller node can also be a network node. The controller node `node001` is selected to be a network node as well for this example (figure 1.11):

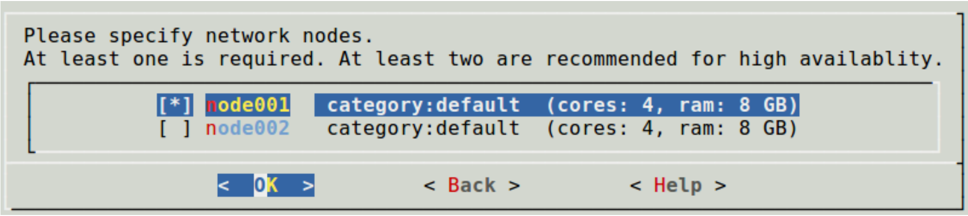

Figure 1.11: Setting the network nodes

12. Floating IP addresses and sNAT should be selected for the external network (figure 1.12):

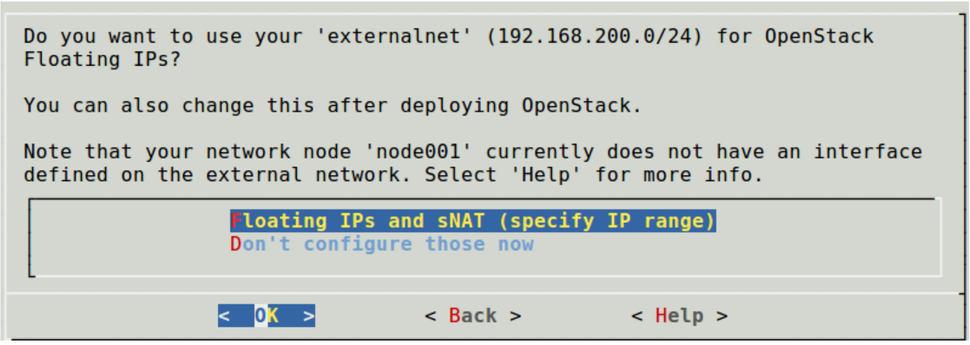

Figure 1.12: Configuring floating IP addresses to be used on the external network

13. The IP address range can then be set up. Many ranges are possible. However, for this example, the range 192.168.200.100-192.168.200.200 is chosen (figure 1.13):

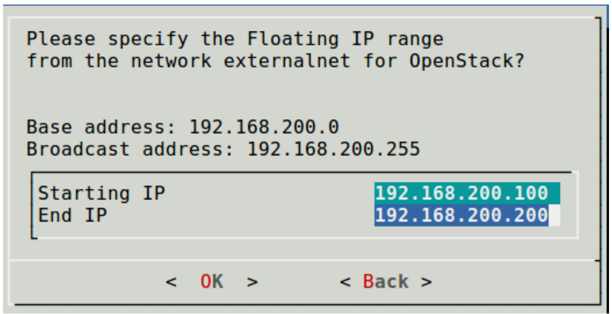

Figure 1.13: Configuring floating IP address range to be used on the external network

14. For the network for virtual networks, `vxlanhostnet`, that was set up in figure 1.10, the hypervisor node should have an interface that connects to it. A shared interface can be set up, and will be an alias for the bridged interface (figure 1.14):

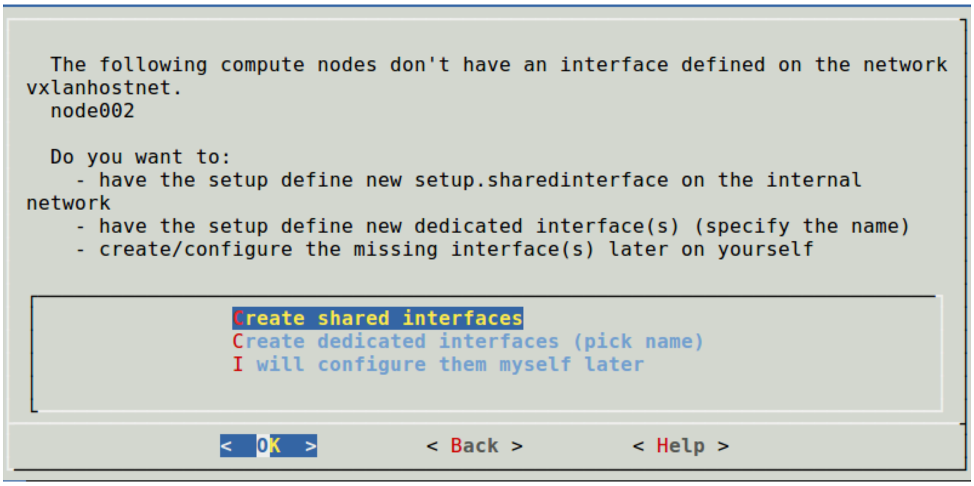

Figure 1.14: Configuring the shared interface on the hypervisor (compute) node

15. Similarly, for `vxlanhostnet`, the network node should also have an interface that connects to it. A shared interface can be set up, and as before will be an alias for the bridged interface (figure 1.15):

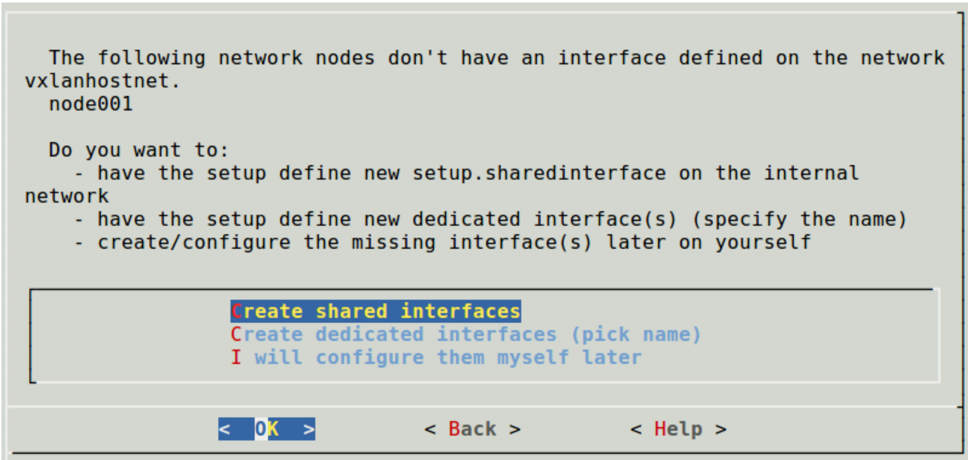

Figure 1.15: Configuring the shared interface on the network node to the virtual networks

16. The head node and the network/controller node are both connected to the external network of the cluster. For the external network that the network/controller node is attached to, a dedicated interface is created (figure 1.16). A name is set for the new interface, for example `eth1`.

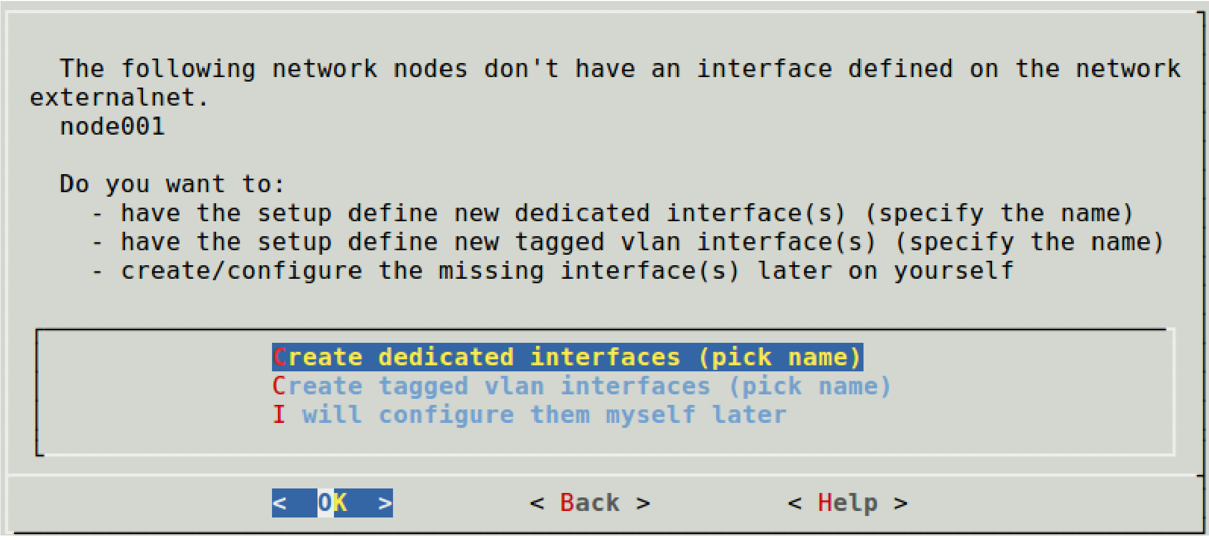

Figure 1.16: Configuring the dedicated interface on the network node to the external network

17. The `Save config & deploy` option (figure 1.17) saves a YAML configuration file of the settings:

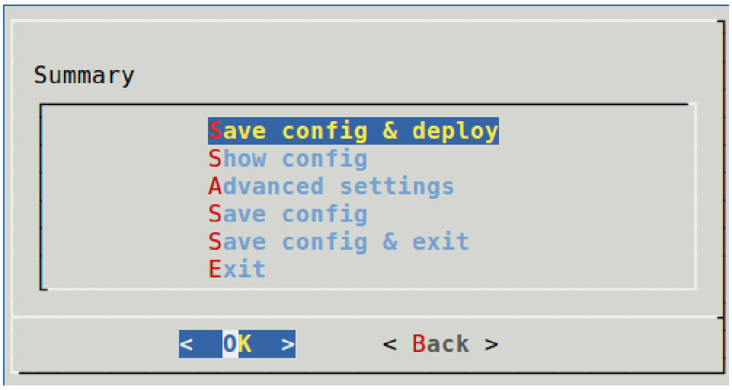

Figure 1.17: Saving and deploying the YAML configuration file

Deployment can then begin.

A deployment can take some time. Progress is displayed throughout the deployment procedure, and the session should end with something like:

```
Took:     35:48 min.
Progress: 100/100
################### Finished execution for 'Bright OpenStack', status: completed

Bright OpenStack finished!
```

1.4 Testing OpenStack Deployment

The example tasks that follow can be used to check if OpenStack has been successfully deployed and is behaving as expected. All the commands are run from Bright head node, and are a handy set of relatively common OpenStack-related actions. The commands in this testing section mostly avoid using the Bright Cluster Manager interface so that the direct OpenStack behavior is visible rather than Bright Cluster Manager behavior. If a command does not work in a similar way to what is shown, then the behavior should be investigated further.

Download a CirrOS image:

```
[root@bright81 ~]# wget -P /tmp/images http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-\
x86_64-disk.img
```

Create an OpenStack test project:

```
[root@bright81 ~]# source .openstackrc    #only needed if no login since deployment
[root@bright81 ~]# openstack project create brighttest
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 0e6cd466a0f849ff8743654940b5f8b8 |
| enabled     | True                             |
| id          | 4c522f2ce1ad4cd18d67de341d1481ff |
| is_domain   | False                            |
| name        | brighttest                       |
| parent_id   | 0e6cd466a0f849ff8743654940b5f8b8 |
+-------------+----------------------------------+
```

Create an OpenStack test user

```
[root@bright81 ~]# openstack user create --project brighttest --password Ch@ngeMe --enable\
 brightuser
+--------------------+----------------------------------+
| Field              | Value                            |
+--------------------+----------------------------------+
| default_project_id | 4c522f2ce1ad4cd18d67de341d1481ff |
| domain_id          | 0e6cd466a0f849ff8743654940b5f8b8 |
| enabled            | True                             |
| id                 | df27f5f7b7da457984616651c2aaed71 |
| name               | brightuser                       |
+--------------------+----------------------------------+
```

Assign an OpenStack role (not to be confused with a Bright role) for the test project and test user:

```
[root@bright81 ~]# openstack role add --project brighttest --user brightuser member
```

This step can be skipped for a user in the `default` project, since the user has the `member` role as a default profile setting anyway. The purpose of this step is to assign a profile setting to the user, otherwise the user cannot carry out any OpenStack functions.

Create the test user in Bright:

```
[root@bright81 ~]# cmsh
[bright81]% user add brightuser
[bright81->user*[brightuser*]]% set password Ch@ngeMe
[bright81->user*[brightuser*]]% commit
```

By default the authentication for the Bright user is separate from OpenStack authentication. User authentication integration with OpenStack is described in detail in section 5.1.

Check the autogenerated `.openstackrc` file for the test user:

```
[root@bright81 ~]# su - brightuser
Last login: Wed Feb 22 15:23:11 CET 2017 on pts/0
Creating ECDSA key for ssh
[brightuser@bright81 ~]$
[brightuser@bright81 ~]$ tail .openstackrc
export OS_PROJECT_DOMAIN_ID="0e6cd466a0f849ff8743654940b5f8b8"
export OS_USER_DOMAIN_ID="0e6cd466a0f849ff8743654940b5f8b8"
# Public Auth URL (used by users)
export OS_AUTH_URL="http://10.2.62.216:5000/v3"

# For keystone v3
export OS_IDENTITY_API_VERSION=3 # for the 'openstack' utility to work
export OS_CACERT="/etc/keystone/ssl/certs/ca.pem"
# END AUTOGENERATED SECTION   -- DO NOT REMOVE
[brightuser@bright81 ~]$
```

Set up the account so that `.openstackrc` is sourced on login: This can be done by editing the `.bashrc` file for the account that is being logged into. The account is `brightuser` in this case, and it can be edited so that the line

```
. /home/brightuser/.openstackrc
```

is placed at the end of `.bashrc`.

A login as `brightuser` then automatically sets up the environment so that the commands of the `openstack` client agent work.

By default, the environment variable `OS_PASSWORD` is not set in `.openstackrc`. This can be set to be sourced during login by placing the line:

```
export OS_PASSWORD="Ch@ngeMe"
```

outside the autogenerated section in the file `.openstackrc`.

The autogenerated section of the `.openstackrc` file is the section within the tags:

```
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
```

and

```
# END AUTOGENERATED SECTION   -- DO NOT REMOVE
```

The `.openstackrc` and `.openstackrc_password` files are described further on page 82.

Create a key pair to be used by the test user: From this point onward, using root privileges is not required to carry out OpenStack tasks. The user has the required privileges to carry out the actions that follow due to the `member` role assignment earlier on.

```
[brightuser@bright81 ~]$ . .openstackrc   #if the file has not yet been sourced
[brightuser@bright81 ~]$ openstack keypair create --public-key\
 ~/.ssh/id_ecdsa.pub brightuser-key
+-------------+-------------------------------------------------+
| Field       | Value                                           |
+-------------+-------------------------------------------------+
| fingerprint | c6:50:f6:9b:c8:ac:7f:5c:e7:ff:54:b7:f7:8e:ec:fd |
| name        | brightuser-key                                  |
| user_id     | 822c81781cf743c3962eae34e97e3cfe                |
+-------------+-------------------------------------------------+
```

Create an OpenStack network:

```
[brightuser@bright81 ~]$ openstack network create brightnet
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2018-06-07T13:08:17Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 71267504-e4d5-43b9-b66d-b8c3a1131d7e |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| mtu                       | 1450                                 |
| name                      | brightnet                            |
| port_security_enabled     | False                                |
| project_id                | cbd65ea4c62e4ec7b8491ce3194227be     |
| provider:network_type     | None                                 |
| provider:physical_network | None                                 |
| provider:segmentation_id  | None                                 |
| qos_policy_id             | None                                 |
| revision_number           | 1                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2018-06-07T13:08:17Z                 |
+---------------------------+--------------------------------------+
```

Create a subnet for the network:

```
[brightuser@bright81 ~]$ openstack subnet create --subnet-range 192.168.100.0/24\
 --network brightnet brightsubnet
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| allocation_pools        | 192.168.100.2-192.168.100.254        |
| cidr                    | 192.168.100.0/24                     |
| created_at              | 2018-06-07T13:08:19Z                 |
| description             |                                      |
| dns_nameservers         |                                      |
| enable_dhcp             | True                                 |
| gateway_ip              | 192.168.100.1                        |
| host_routes             |                                      |
| id                      | e0bccea2-9b35-42e8-9357-7a83b2dcba1e |
| ip_version              | 4                                    |
| ipv6_address_mode       | None                                 |
| ipv6_ra_mode            | None                                 |
| name                    | brightsubnet                         |
| network_id              | 71267504-e4d5-43b9-b66d-b8c3a1131d7e |
```

```
| project_id              | cbd65ea4c62e4ec7b8491ce3194227be     |
| revision_number         | 0                                    |
| segment_id              | None                                 |
| service_types           |                                      |
| subnetpool_id           | None                                 |
| tags                    |                                      |
| updated_at              | 2018-06-07T13:08:19Z                 |
| use_default_subnet_pool | None                                 |
+-------------------------+--------------------------------------+
```

Create a router:

```
[brightuser@bright81 ~]$ openstack router create brightrouter
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2018-06-07T13:08:21Z                 |
| description             |                                      |
| distributed             | False                                |
| external_gateway_info   | None                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| id                      | 4f927596-59c7-46e6-a293-47fb4c7682ac |
| name                    | brightrouter                         |
| project_id              | cbd65ea4c62e4ec7b8491ce3194227be     |
| revision_number         | None                                 |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    |                                      |
| updated_at              | 2018-06-07T13:08:21Z                 |
+-------------------------+--------------------------------------+
```

Attach the router to the bright-external-flat-externalnet: (this is the flat network which is bridged with the interface to the outside)

```
[brightuser@bright81 ~]$ openstack router set brightrouter --external-gateway\
 bright-external-flat-externalnet
```

Attach the router to the subnet created earlier:

```
[brightuser@bright81 ~]$ openstack router add subnet brightrouter brightsubnet
```

Import the CirrOS image from the downloaded CirrOS cloud image:

```
[brightuser@bright81 ~]$ openstack image create --disk-format qcow2 --container-format bare \
 --file /tmp/images/cirros-0.4.0-x86_64-disk.img  cirros-0.4.0-x86_64
+------------------+------------------------------------------------------------------+
| Field            | Value                                                            |
+------------------+------------------------------------------------------------------+
| checksum         | 443b7623e27ecf03dc9e01ee93f67afe                                 |
| container_format | bare                                                             |
| created_at       | 2018-06-07T13:08:34Z                                             |
```

```
| disk_format       | qcow2                                                            |
| file              | /v2/images/3f4d7b8e-d827-4a98-a9aa-ec571856a3bc/file             |
| id                | 3f4d7b8e-d827-4a98-a9aa-ec571856a3bc                             |
| min_disk          | 0                                                                |
| min_ram           | 0                                                                |
| name              | cirros-0.4.0-x86_64                                              |
| owner             | cbd65ea4c62e4ec7b8491ce3194227be                                 |
| properties        | direct_url='file:///cm/shared/apps/openstack/glance-images/      |
|                                               3f4d7b8e-d827-4a98-a9aa-ec571856a3bc'    |
| protected         | False                                                            |
| schema            | /v2/schemas/image                                                |
| size              | 12716032                                                         |
| status            | active                                                           |
| tags              |                                                                  |
| updated_at        | 2018-06-07T13:08:34Z                                             |
| virtual_size      | None                                                             |
| visibility        | shared                                                           |
+-------------------+------------------------------------------------------------------+
```

Create a CirrOS VM on the network `brightnet`**:**

```
[brightuser@bright81 ~]$ openstack network list
+---------------------+-----------------------------------+---------------------+
| ID                  | Name                              | Subnets             |
+---------------------+-----------------------------------+---------------------+
| 11c2e29b-68ef-45... | bright-external-flat-externalnet | 5a21577a-bdb1-4a... |
| 71267504-e4d5-43... | brightnet                         | e0bccea2-9b35-42... |
+---------------------+-----------------------------------+---------------------+

[brightuser@bright81 ~]$ networkidbrightnet=$(openstack network list -c ID --name brightnet\
 -f value)
[brightuser@bright81 ~]$ echo $networkidbrightnet
[brightuser@bright81 ~]$ 71267504-e4d5-43b9-b66d-b8c3a1131d7e
[brightuser@bright81 ~]$ openstack server create --image cirros-0.4.0-x86_64 --flavor m1.xs\
mall --key-name brightuser-key --nic net-id=$networkidbrightnet cirrosvm
+-----------------------------+-----------------------------------------------------------+
| Field                       | Value                                                     |
+-----------------------------+-----------------------------------------------------------+
| OS-DCF:diskConfig           | MANUAL                                                    |
| OS-EXT-AZ:availability_zone |                                                           |
| OS-EXT-STS:power_state      | NOSTATE                                                   |
| OS-EXT-STS:task_state       | scheduling                                                |
| OS-EXT-STS:vm_state         | building                                                  |
| OS-SRV-USG:launched_at      | None                                                      |
| OS-SRV-USG:terminated_at    | None                                                      |
| accessIPv4                  |                                                           |
| accessIPv6                  |                                                           |
| addresses                   |                                                           |
| adminPass                   | amDodTX2u8DU                                              |
| config_drive                |                                                           |
| created                     | 2018-06-07T13:09:59Z                                      |
| flavor                      | m1.xsmall (81b7d8db-2bc6-4745-beb9-5d6f84f15419)          |
| hostId                      |                                                           |
| id                          | 9488aa29-f765-478a-a742-16cfe7481c57                      |
| image                       | cirros-0.4.0-x86_64 (3f4d7b8e-d827-4a98-a9aa-ec571856a3bc) |
| key_name                    | brightuser-key                                            |
```

```
| name                         | cirrosvm                                                    |
| progress                     | 0                                                           |
| project_id                   | cbd65ea4c62e4ec7b8491ce3194227be                            |
| properties                   |                                                             |
| security_groups              | name='default'                                              |
| status                       | BUILD                                                       |
| updated                      | 2018-06-07T13:09:59Z                                        |
| user_id                      | 822c81781cf743c3962eae34e97e3cfe                            |
| volumes_attached             |                                                             |
+------------------------------+-------------------------------------------------------------+
```

Create a floating IP:

```
[brightuser@bright81 ~]$ openstack floating ip create bright-external-flat-externalnet
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2018-06-07T13:10:31Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 192.168.200.12                       |
| floating_network_id | 11c2e29b-68ef-4512-8fa9-35fe14596c1b |
| id                  | e4710e3b-8b91-4560-aca2-9bb4c282b3d5 |
| name                | 192.168.200.12                       |
| port_id             | None                                 |
| project_id          | cbd65ea4c62e4ec7b8491ce3194227be     |
| revision_number     | 0                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| updated_at          | 2018-06-07T13:10:31Z                 |
+---------------------+--------------------------------------+
```

Attach the floating IP to the CirrOS VM:

```
[brightuser@bright81 ~]$ floatingipflatexternalnet=$(openstack floating ip list -c\
 "Floating IP Address" -f value)
[brightuser@bright81 ~]$ echo $floatingipflatexternalnet
192.168.200.12
[brightuser@bright81 ~]$ openstack server add floating ip cirrosvm $floatingipflatexternalnet
```

Enable ssh port 22 in the default security group:

```
[brightuser@bright81 ~]$ openstack security group rule create --dst-port 22 default
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| id                    | 9f10223b-4cdf-4e7b-aa72-879c85710bb8 |
| ip_protocol           | tcp                                  |
| ip_range              | 0.0.0.0/0                            |
| parent_group_id       | 5affac60-34b8-4217-8670-c82a8c8e2d88 |
| port_range            | 22:22                                |
| remote_security_group |                                      |
+-----------------------+--------------------------------------+
```

Test ssh access to the CirrOS VM:

```
[brightuser@bright81 ~]$ ssh cirros@$floatingipflatexternalnet
Warning: Permanently added '192.168.200.12' (RSA) to the list of known hosts.
$ hostname
cirrosvm
```

2

Introduction

OpenStack is an open source implementation of cloud services. It is currently (2017) undergoing rapid development, and its roadmap is promising.

An implementation of OpenStack, based on the OpenStack Pike release (`https://www.openstack.org/software/Pike/`) is integrated into the Bright Cluster Manager 8.1 for OpenStack edition. It is supported for RHEL7 onwards.

The implementation of OpenStack is usable and stable for regular use in common configurations. In a complex and rapidly-evolving product such as OpenStack, the number of possible unusual configuration changes is vast. As a result, the experience of Bright Computing is that Bright Cluster Manager can sometimes run into OpenStack issues while implementing the less common OpenStack configurations.

As one of the supporting organizations of OpenStack, Bright Computing is committed towards working together with OpenStack developers to help Bright customers resolve any such issue. The end result after resolving the issue means that there is a selection pressure that helps evolve that aspect of OpenStack, so that it becomes convenient and stable for regular use. This process benefits all participants in the OpenStack software ecosystem.

OpenStack consists of subsystems, developed as upstream software projects[1]. A software project provides capabilities to OpenStack via the implementation of a backend service, and thereby provides an OpenStack service. The OpenStack service can thus be implemented by interchangeable backends, which projects can provide.

For example, the OpenStack Cinder project provides block storage capabilities to OpenStack via the implementation of, for example, NFS or Ceph block storage. The OpenStack's block storage service can therefore be implemented by the interchangable backends of the NFS or Ceph projects. Indeed, the entire Cinder project itself can be replaced by a Cinder rewrite from scratch. As far as the user is concerned the end result is the same.

An analogy to OpenStack interchangable subsystem backends provided by projects, is operating system interchangeable subsystem backends , as provided by distributions packages:

An operating system distribution consists of subsystems, maintained as packages and their dependencies. Some subsystems provide capabilities to the operating system via the implementation of a backend service. The service can often be implemented by interchangeable backends for the subsystem.

A specific example for an operating system distribution would be the mailserver subsystem that provides mail delivery capabilities to the operating system via the implementation of, for example, Postfix or Sendmail. The mailserver package and dependencies can therefore be implemented by the interchangeable backends of the Postfix or Sendmail software. As far as the e-mail user is concerned, the end result is the same.

The project that implements the backend can also change, if the external functionality of the project remains the same.

Some of the more common OpenStack projects are listed in the following table:

[1]The term projects must not be confused with the term used in OpenStack elsewhere, where projects, or sometimes tenants, are used to refer to a group of users

Service	OpenStack Project	Managed By Bright
Compute	Nova	✓
Object Storage	Swift	depends*
Block Storage	Cinder	✓
Networking	Neutron	✓
Dashboard	Horizon	✓
Identity Service	Keystone	✓
Orchestration	Heat	✓
Telemetry	Ceilometer	×
Database Service	Trove	×
Image Service	Glance	✓

* Bright Cluster Manager does not manage the OpenStack reference implementation for Swift object storage, but does manage a replacement, the API-compatible Ceph RADOS Gateway implementation.

Not all of these projects are integrated, or needed by Bright Cluster Manager for a working OpenStack system. For example, Bright Cluster Manager already has an extensive monitoring system and therefore does not for now implement `Ceilometer`, while `Trove` is ignored for now until it becomes more popular.

Projects that are not yet integrated can in principle be added by administrators on top of what is deployed by Bright Cluster Manager, even though this is not currently supported or tested by Bright Computing. Integration of the more popular of such projects, and greater integration in general, is planned in future versions of Bright Cluster Manager.

This manual explains the installation, configuration, and some basic use examples of the OpenStack projects that have so far been integrated with Bright Cluster Manager.

3

OpenStack Installation

OpenStackRHEL7 And Derivatives Only

Bright OpenStack is supported for RHEL7 and derivatives only.

To Use Ceph, It Must Be Installed Before Deploying OpenStack

If OpenStack is to access Ceph for storage purposes, for any combination of block storage (Cinder), image storage (Glance), ephemeral storage (Nova), or object storage (RADOS Gateway), then the Ceph components must first be installed with `cm-ceph-setup` (Chapter 4) before starting the OpenStack installation procedure covered here.

Hardware Requirement For Running OpenStack

The optimum hardware requirements for OpenStack depend on the intended use. A rule of thumb is that the number of cores on the compute nodes determines the number of virtual machines.

OpenStack itself can run entirely on one physical machine for limited demonstration purposes.

However, if running OpenStack with Bright Cluster Manager, then a standard reference architecture used by Bright Computing consists of the following three types of nodes:

- A head node.
- Several regular nodes that can be used as hypervisor hosts. Regular nodes (Bright Cluster Manager terminology) are also commonly called compute nodes, and are typically multicore. Running guest VMs is therefore a suitable use for regular nodes.
- 3 nodes that combine OpenStack controller and OpenStack network node functionality.

For a standard reference configuration, minimal hardware specifications for useful demonstration purposes are:

- **Head node**: 8GB RAM, 4 cores and two network interfaces. In a standard configuration the head node does not run OpenStack services, other than the OpenStack-associated Haproxy service.
- **Regular nodes:** 2GB RAM per core. Each regular node has a network interface.
  - In larger clusters, it may be a good idea to separate the OpenStack controller functionality from networking functionality. If a regular node is configured as a controller, then it must have at least 8GB RAM.
- **3 OpenStack controller/network nodes**: 8GB RAM and two network interfaces. 3 nodes is the minimum needed to provide OpenStack high availability via Galera cluster for OpenStack databases.

  The controller/network nodes can be separated from each other, but it is usually convenient to keep them together. Bright Cluster Manager OpenStack edition therefore uses combined controller/network nodes by default.

The database for the controller nodes cannot run with two OpenStack controllers. If the administrator would like use something other than the standard reference controller configuration of 3 controllers, then it is possible to run with just one OpenStack controller, without OpenStack database high availability.

3 controllers allows one to be rebooted while the other two provide quorum. However, rebooting two at the same time in such a configuration risks data loss.

More than three controllers are also allowed, in a high-availability configuration.

Setting the `datanode` property of controller nodes to `yes` (page 164 of the *Administrator Manual*) is recommended, so that a FULL install of a controller requires some effort to carry out.

The OpenStack controller/network nodes provide:

- OpenStack API endpoint services for Nova, Cinder, Keystone, Neutron, Glance, and Heat.
- Horizon Dashboard. This is a Django-based web service.
- RabbitMQ nodes, deployed as a RabbitMQ cluster. This is used in the OpenStack backend for internal communication within an OpenStack service. For example, such as between `nova-api`, `nova-conductor`, `nova-scheduler`, `nova-compute`, or such as between `neutron-server` and the Neutron L2 agents.
- If Ceph is used, then the Ceph monitor nodes can also be used as the controller nodes, in order to provide high availability for the Ceph monitor node data. In this case, more than 8GB of memory is needed for the controller nodes.

An *ethernet fabric* is used as a terminology to talk about treating the network architecture as being based on a giant flat logical OSI Layer 2-style network connected to a single switch, with point-to-point routing, rather than the traditional OSI 2/3 mixture with a hierarchy of access, distribution, and core routers.

The reference architecture networking runs on an ethernet fabric for the:

- internal network of the cluster, which is also the OpenStack management network.
- V(X)LAN network of the cluster, which is used by OpenStack virtual networks.

If Ceph is also deployed, then an ethernet fabric is assumed for:

- The public Ceph network.
- The Ceph replication network.
- An optional external network that is used to access virtual machines in OpenStack via Floating IPs.

Hard drive requirements for minimal systems can remain as for those required for a regular Bright Cluster Manager cluster. For production systems, these minimal requirements are however unlikely to work for very long. Storage requirements should therefore be considered with care according to the use case. If necessary, Bright Computing can provide advice on this.

Running OpenStack under Bright Cluster Manager with fewer resources than suggested in the preceding is possible but may cause issues. While such issues can be resolved, they are usually not worth the time spent analyzing them, due to the great number of possible configurations. It is better to run with ample resources, and then analyze the resource consumption in the configuration that is used, to see what issues to be aware of when scaling up to a production system.

Running a Bright Cluster Manager OpenStack cluster that varies greatly from the reference cluster is also possible. If necessary, Bright Computing can provide advice on this.

Ways Of Installing OpenStack

The version of OpenStack that is integrated with Bright Cluster Manager can be installed in the following two ways:

- Using the web-based `Setup wizard` menu option (section 3.1), accessed via the OpenStack resource, if OpenStack has not already been installed. This is the recommended installation method.
- Using the text-based `cm-openstack-setup` utility (section 3.2). The utility is a part of the standard `cluster-tools` package.

The priorities that the package manager uses are expected to be at their default settings, in order for the installation to work.

By default, deploying OpenStack installs the following projects: Keystone, Nova, Cinder, Glance, Neutron, Heat and Horizon (the dashboard).

If Ceph is used, then Bright also deploys RADOS Gateway as a Swift-API-compatible object storage system. Using RADOS Gateway instead of the reference Swift object storage is regarded in the OpenStack community as good practice, and is indeed the only object storage system that Bright Cluster Manager manages for OpenStack. Alternative backend storage is possible at the same time as object storage, which means, for example, that block and image storage are options that can be used in a cluster at the same time as object storage.

3.1 Installation Of OpenStack From Bright View

Using Bright View is the preferred way to install OpenStack. A prerequisite for running it is that the head node should be able to connect to the distribution repositories, or alternatively the head node should have OpenStack RPMs preinstalled on it. Preinstalled OpenStack RPMs can be configured as part of the head node installation from the ISO, if the ISO that is used the Bright Cluster Manager OpenStack edition.

Some suggestions and background notes These are given here to help the administrator understand what the setup configuration does, and to help simplify deployment. Looking at these notes after a dry-run with the wizard will probably be helpful.

- A VXLAN (Virtual Extensible LAN) network is similar to a VLAN network in function, but has features that make it more suited to cloud computing.
  - If VXLANs are to be used, then the wizard is able to help create a VXLAN *overlay network* for OpenStack *tenant networks*.

    An OpenStack tenant network is a network used by a group of users allocated to a particular virtual cluster.

    A VXLAN overlay network is a Layer 2 network "overlaid" on top of a Layer 3 network. The VXLAN overlay network is a virtual LAN that runs its frames encapsulated within UDP packets over the regular TCP/IP network infrastructure. It is very similar to VLAN technology, but with some design features that make it more useful for cloud computing needs. One major improvement is that around 16 million VXLANs can be made to run over the underlying Layer 3 network. This is in contrast to the 4,000 or so VLANs that can be made to run over their underlying Layer 2 network, if the switch port supports that level of simultaneous capability.

    By default, if the VXLAN network and VXLAN network object do not exist, then the wizard helps the administrator create a `vxlanhostnet` network and network object (section 3.1.10). The network is attached to, and the object is associated with, all non-head nodes taking part in the OpenStack deployment. If a `vxlanhostnet` network is pre-created beforehand, then the wizard can guide the administrator to associate a network object with it, and ensure that all the non-head nodes participating in the OpenStack deployment are attached and associated accordingly.

- The VXLAN network runs over an IP network. It should therefore have its own IP range, and each node on that network should have an IP address. By default, a network range of 10.161.0.0/16 is suggested in the VXLAN configuration screen (section 3.1.10, figure 3.12).
- The VXLAN network can run over a dedicated physical network, but it can also run over an alias interface on top of an existing internal network interface. The choice is up to the administrator.
- It is possible to deploy OpenStack without VXLAN overlay networks if user instances are given access to the internal network. Care must then be taken to avoid IP addressing conflicts.

• When allowing for Floating IPs and/or enabling outbound connectivity from the virtual machines (VMs) to the external network via the network node, the network node can be pre-configured manually according to how it is connected to the internal and external networks. Otherwise, if the node is not pre-configured manually, the wizard then carries out a basic configuration on the network node that

  - configures one physical interface of the network node to be connected to the internal network, so that the network node can route packets for nodes on the internal network.
  - configures the other physical interface of the network node to be connected to the external network so that the network node can route packets from external nodes.

The wizard asks the user several questions on the details of how OpenStack is to be deployed. From the answers, it generates an YAML document with the intended configuration. Then, in the back-end, largely hidden from the user, it runs the text-based `cm-openstack-setup` script (section 3.2) with this configuration on the active head node. In other words, the wizard can be regarded as a GUI front end to the `cm-openstack-setup` utility.

The practicalities of executing the wizard: The explanations given by the wizard during its execution steps are intended to be verbose enough so that the administrator can follow what is happening.

The wizard is accessed via the OpenStack resource in the navigation pane of Bright View (figure 3.1). Launching the wizard is only allowed if the Bright Cluster Manager license (Chapter 4 of the *Installation Manual*) entitles the license holder to use OpenStack.

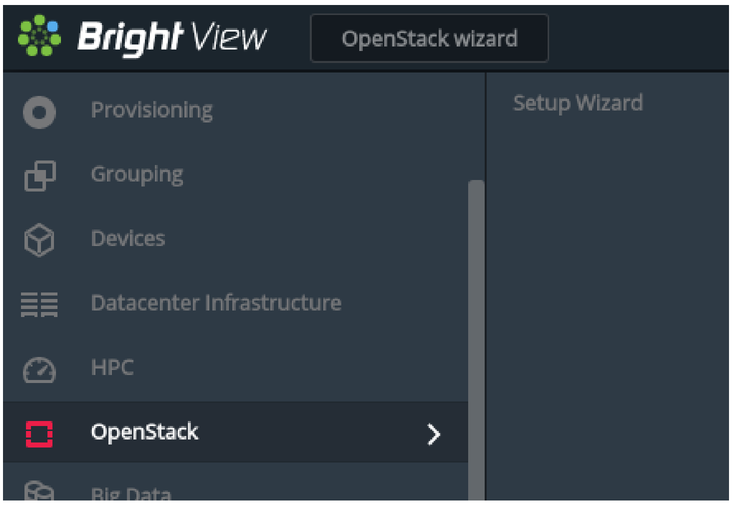

Figure 3.1: The `Setup wizard` Menu Option In Bright View's OpenStack Resource

The wizard runs through the screens in sections 3.1.1-3.1.14, described next.

3.1.1 OpenStack Setup Wizard Overview

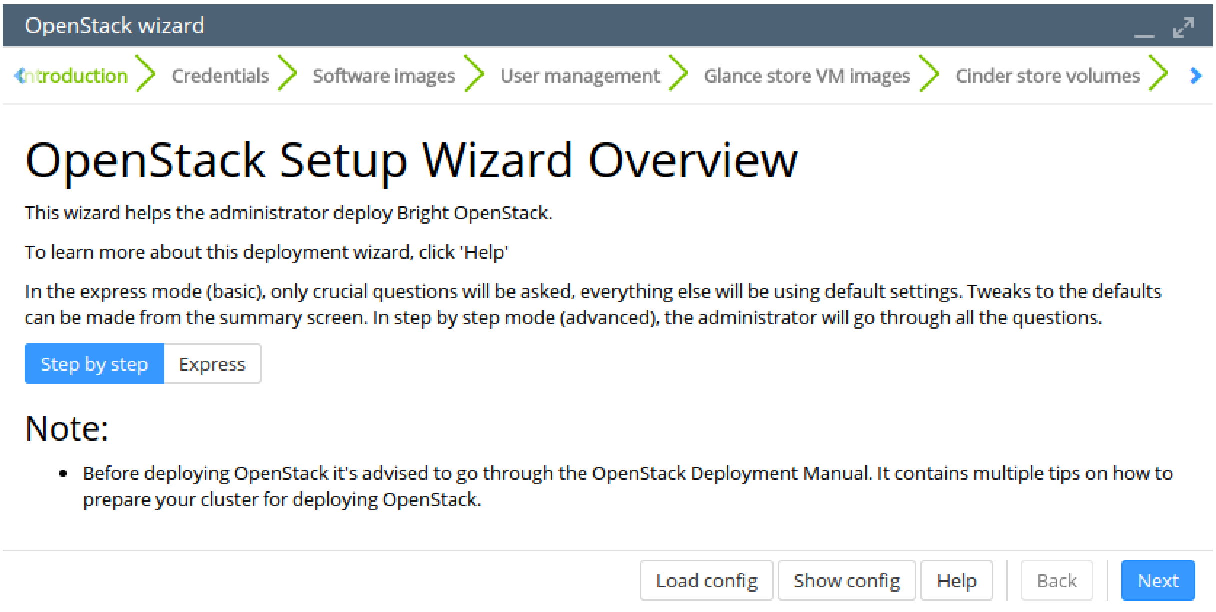

Figure 3.2: OpenStack Setup Wizard Overview Screen

The main overview screen (figure 3.2) gives an overview of how the wizard runs.

The main overview screen also asks for input on if the wizard should run in step-by-step mode, or in express mode.

- Step-by-step mode asks for many explicit configuration options, and can be used by the administrator to become familiar with the configuration options.
- Express mode asks for very few configuration options, and uses mostly default settings. It can be used by an administrator that would like to try out a relatively standard configuration.

During the wizard procedure, buttons are available at the bottom of the screen. Among other options, in the main overview screen, the buttons allow a previously-saved configuration to be loaded, or allow the current configuration to be saved. The configurations are loaded or saved in a YAML format.

On clicking the `Next` button:

- If the express mode has been chosen, then the wizard goes through the credentials screen (section 3.1.2), after which it skips ahead to the `Summary` screen (section 3.1.14).
- Otherwise, if the step-by-step mode has been chosen, then each time the `Next` button is clicked, the wizard goes to the next screen in the series of in-between steps. Each screen allows options to be configured.

  The steps are described in the following sections 3.1.2-3.1.14.

3.1.2 OpenStack admin User Screen

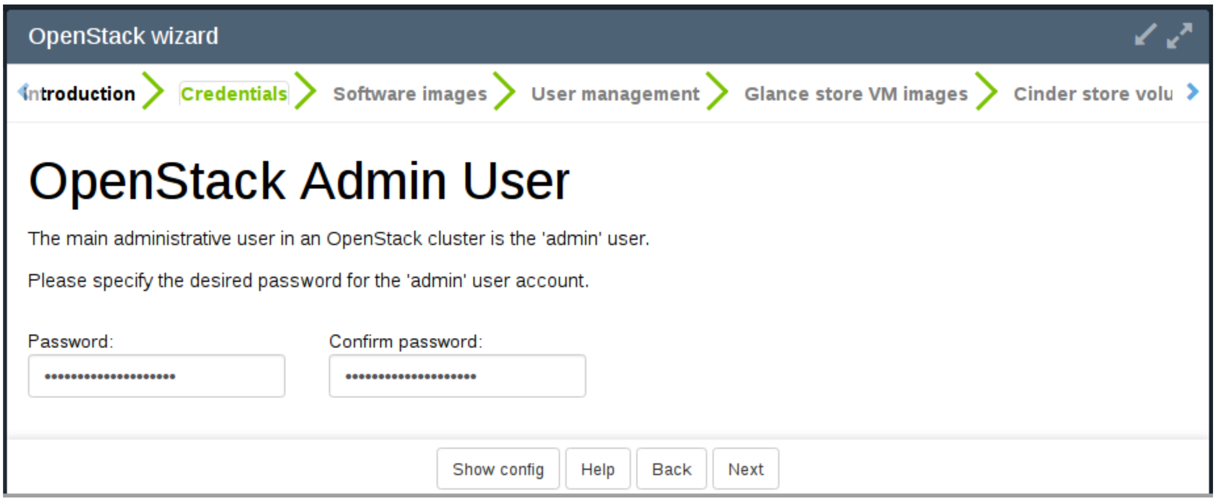

Figure 3.3: OpenStack admin User Screen

The OpenStack credentials screen (figure 3.3) allows the administrator to set the password for the OpenStack admin user. The admin user is how the administrator logs in to the Dashboard URL to manage OpenStack when it is finally up and running. If express mode has been chosen, then the Next button has the wizard skip ahead to the Summary screen (section 3.1.14).

3.1.3 OpenStack Software Image Selection

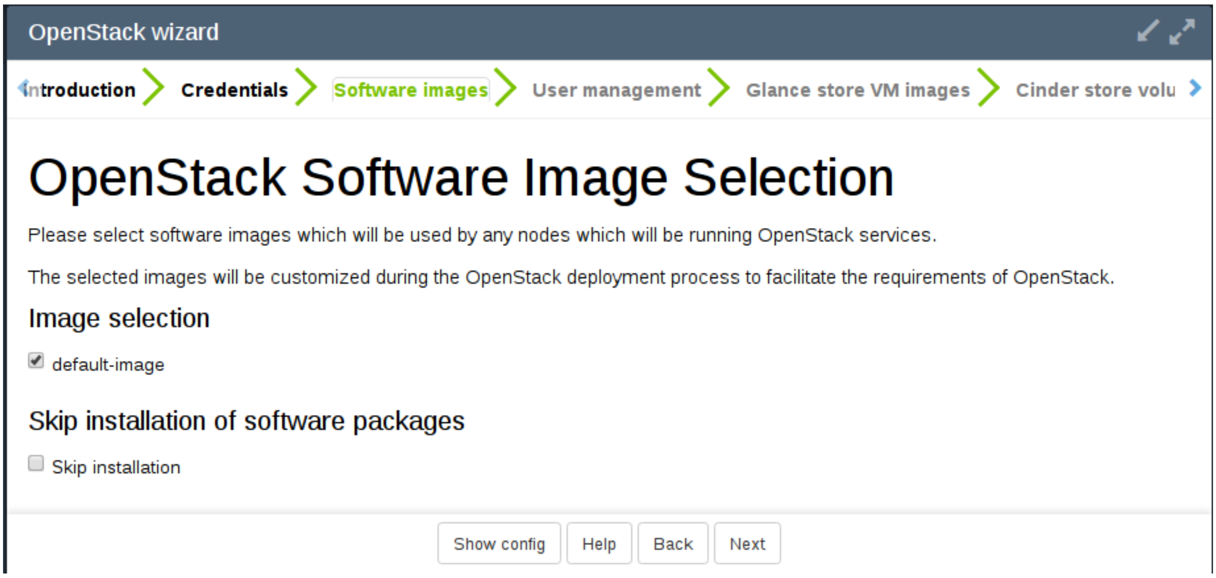

Figure 3.4: OpenStack Software Image Selection Screen

The OpenStack software image selection screen (figure 3.4) lets the administrator select the software image that is to be modified and used on the nodes that run OpenStack.

The administrator can clone the default-image before running the wizard and modifying the image, in order to keep an unmodified default-image as a backup.

The administrator should take care not to move a node with OpenStack roles to another category that contains a different image without OpenStack roles. OpenStack nodes behave quite differently from non-OpenStack nodes.

3.1.4 User Management

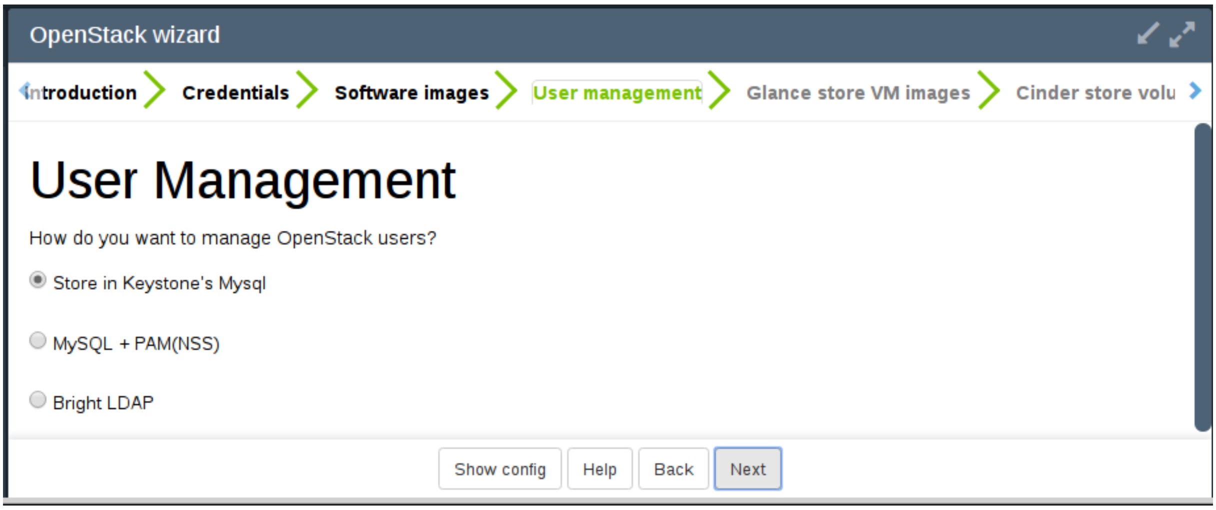

Figure 3.5: OpenStack User Management Screen

The `User Management` screen (figure 3.5) allows the administrator to select how OpenStack users are to be managed. Choices available are:

- Store in a MySQL database managed by Keystone, and by default isolate users from the non-OpenStack part of the cluster.

  Thus, in this case, the OpenStack users are managed by Keystone, and isolated from the LDAP users managed by Bright Cluster Manager.

- Store in a MySQL database managed by Keystone, and use PAM (NSS). Further details on this can be found in the background note on page 76.

- Use Bright Cluster Manager LDAPS authentication. Further details on this can be found in the background note on page 76.

Keystone can also be set to authenticate directly with an external LDAP or AD server, but this requires manual configuration in Bright Cluster Manager. In `cmsh` this configuration can be done as follows:

Example

```
[root@bright81 ~]# cmsh
[bright81]% openstack settings default
[bright81->openstack[default]->settings]% authentication
[bright81->...->settings->authentication]% set custompublicauthhost <external authentication server>
```

3.1.5 Glance VM Image Storage

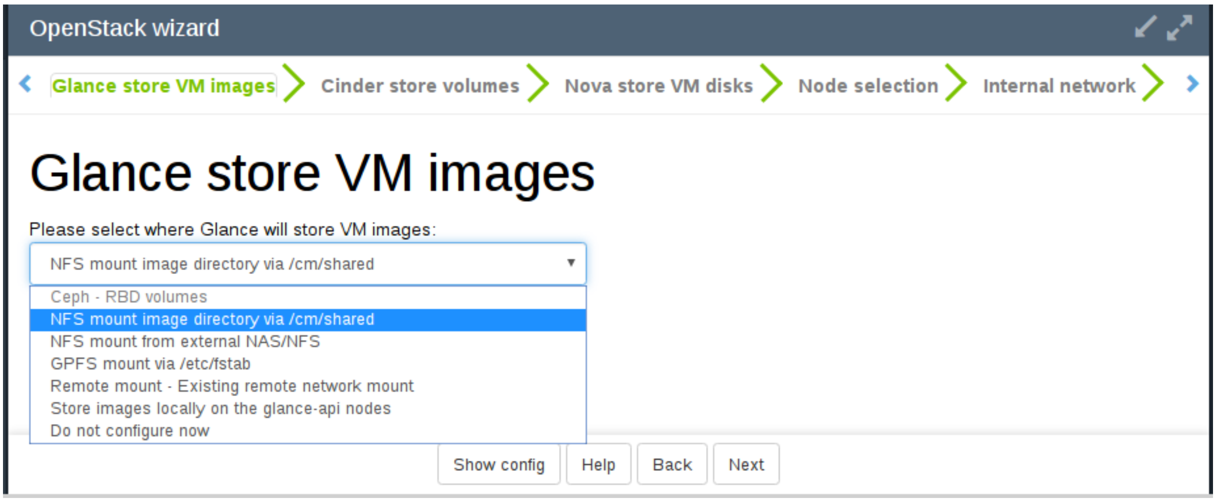

Figure 3.6: OpenStack Glance VM Image Storage Screen

The `Glance VM Image Storage` screen (figure 3.6) allows the administrator to select where virtual machine images are stored. Choices are:

- As Ceph-RBD volumes
- Within an NFS image directory, using the internal NFS. This is using a directory under `/cm/shared`
- Within an NFS image directory, using an external NAS/NFS. The share location, mount point and mount options are prompted for if this choice is selected.
- Within a GPFS image directory, mounted via `/etc/fstab`. The share location, mount point, and mount options are prompted for if this choice is selected.
- Using a remote mount from another network file system. The mount point is prompted for if this choice is selected.
- As images stored locally on the glance-api nodes.

3.1.6 Cinder Volume Storage

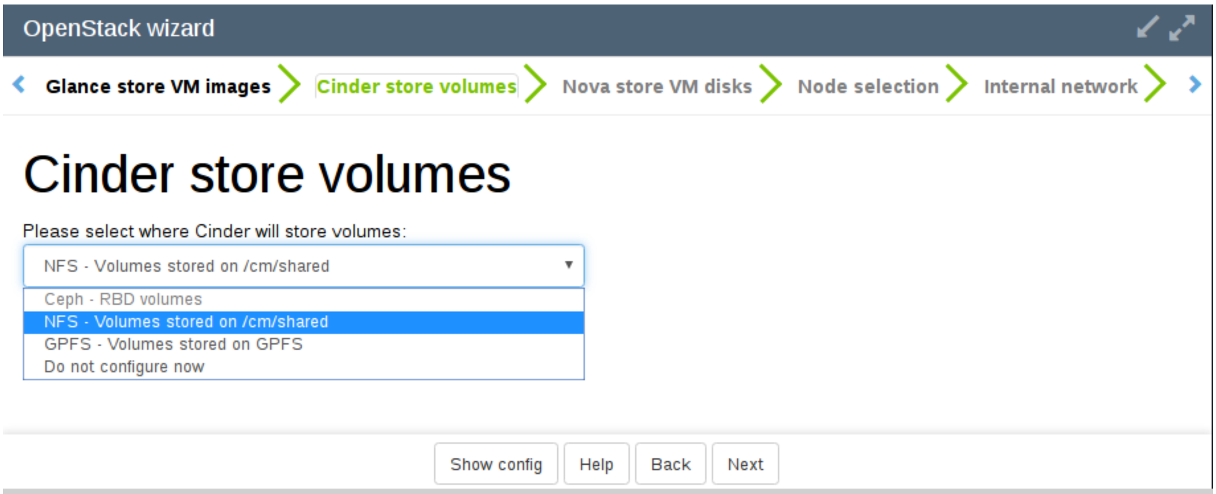

Figure 3.7: OpenStack Cinder Volume Storage Screen

The `Cinder Volume Storage` screen (figure 3.7) allows the administrator to choose how Cinder volumes are to be stored. Options are:

- As Ceph-RBD volumes
- Within an NFS directory, using the internal NFS. This is using a directory under `/cm/shared`
- Within a GPFS volume. The mount point is prompted for if this choice is selected.

3.1.7 Nova VM Disks Storage

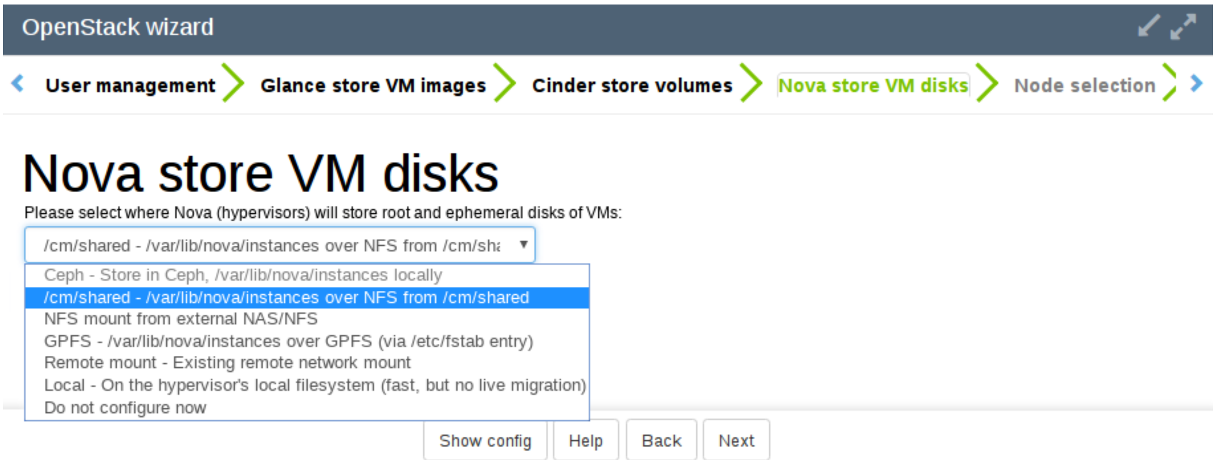

Figure 3.8: OpenStack Nova VM Disks Storage Screen

The `Nova VM Disks Storage` screen (figure 3.8) allows the administrator to choose how Nova hypervisors store the root and ephemeral disks of VMs. Options are:

- Ceph: Stored locally under `/var/lib/nova/instances`

- An NFS directory, using the internal NFS. This is using a directory served from /cm/shared as /var/lib/nova/instances.
- An NFS directory, using an external NAS/NFS. The share location, and mount options are prompted for if this choice is selected.
- A GPFS directory, mounted via /etc/fstab. The directory is served as /var/lib/nova/instances
- A remote mount from another network file system. The mount point is prompted for if this choice is selected.
- A local filesystem on the hypervisor itself, under /var/lib/nova. This is fast, but does not support live migration.

3.1.8 OpenStack Nodes Selection

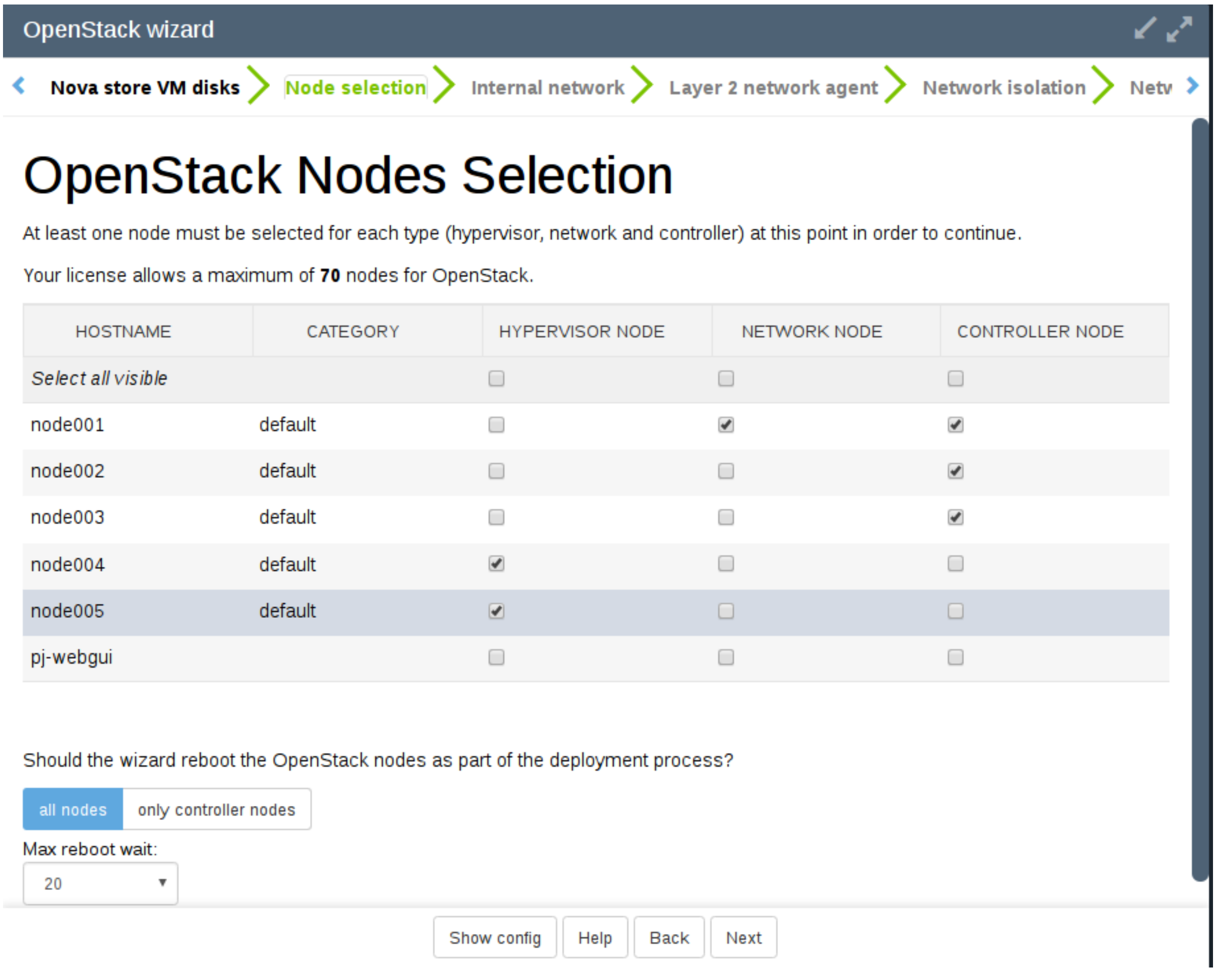

Figure 3.9: OpenStack Nodes Selection

The OpenStack Nodes Selection screen allows the administrator to toggle whether a node takes on the function type of hypervisor node, network node, or controller node.

- A hypervisor node hosts virtual nodes. Typically a hypervisor node has many cores. The more hypervisors there are, the more VMs can be run.

- A network node runs DHCP and legacy routing services. At least one is required, and two are recommended for high availability DHCP and routing for production systems. In the reference architecture (page 17) the set of network nodes is the same as the set of controller nodes. This means that in the reference architecture case each of the controller nodes is running on a machine which is also running a network node within that same machine, which means the resulting hybrid machine can be called a controller/network node. There are therefore 3 controller/network nodes in the reference architecture.

- A controller node runs RabbitMQ services. At least one is required, and three are recommended for high-availability production systems.

Each of these three function types must exist at least once in the cluster. Each node can have multiple functions types, and each function type can be allocated to many nodes. Combining hypervisor nodes with controller nodes is however usually not recommended, due to the high CPU load from controller services.

Within the `OpenStack Nodes Selection` screen, clicking on the `HOSTNAME` column makes it possible to filter the list of nodes that is displayed (figure 3.10) so that it is easier to tick the correct checkboxes in large clusters.

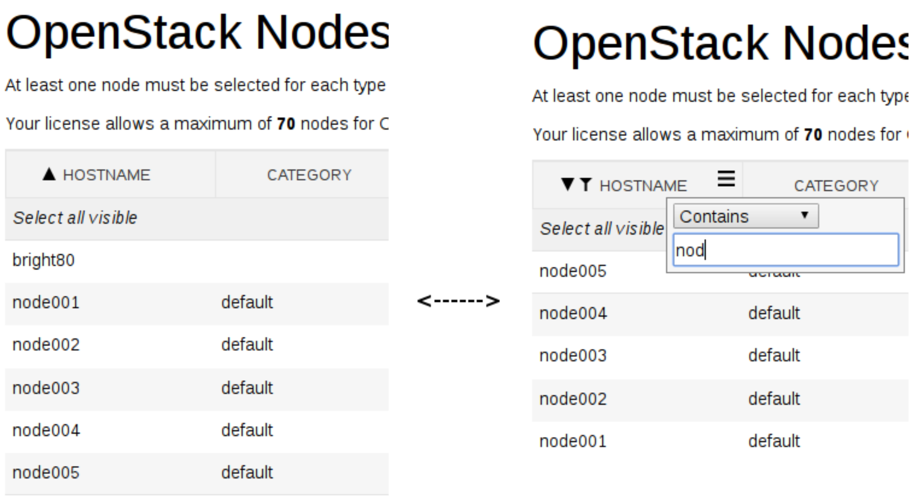

Figure 3.10: OpenStack Nodes Selection Filtering

When the OpenStack installation wizard completes, and configuration is deployed, the OpenStack nodes are all set to reboot by default. However, the `OpenStack Nodes Selection` screen also allows the rebooting of just the controller nodes, which is often sufficient.

When a node reboots, it can take some time to be provisioned. The time to wait for reboot is configurable in the `OpenStack Nodes Selection` screen.

3.1.9 OpenStack Internal Network Selection Screen

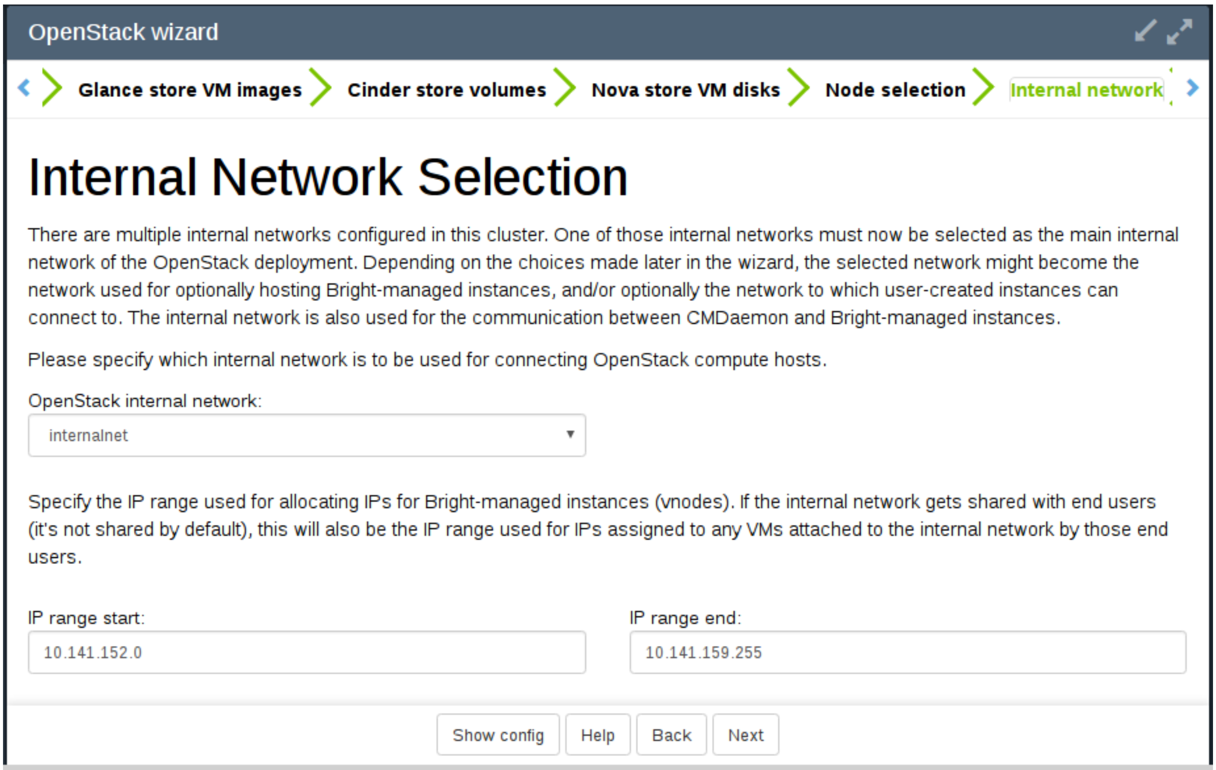

Figure 3.11: OpenStack Internal Network Selection

The `OpenStack Internal Network Selection` screen allows the administrator to set the main internal network of the OpenStack nodes. This network is the network that is used to host Bright-managed instances and is also the network that user-created instances can connect to.

By default for a default Bright Cluster Manager installation, `internalnet` is used. A subset of the network is configured for OpenStack use by setting appropriate IP ranges.

3.1.10 OpenStack Network Isolation And VLAN/VXLAN Configuration

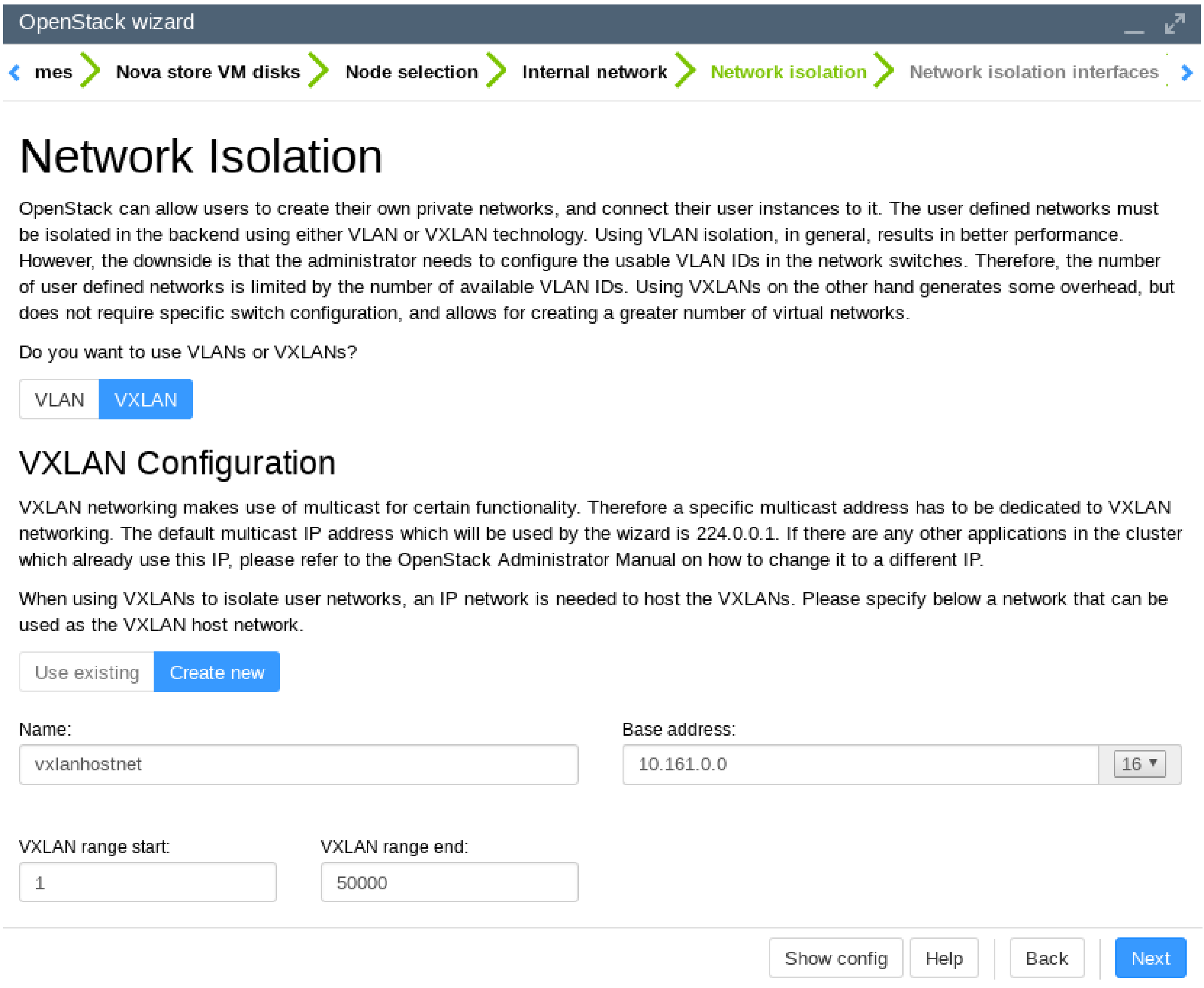

Figure 3.12: OpenStack Network Isolation And VXLAN Configuration Screen

The OpenStack Network Isolation And VXLAN Configuration screen allows the administrator to decide on the network isolation technology that is to be used for the private network of OpenStack user instances. The options, selectable by buttons, are either VLANs or VXLANS. Accordingly, VLAN screen options or closely similar VXLAN screen options, are then displayed. VXLANs are recommended by default due to their greater ease of use.

VLAN Screen Options

The VLAN range defines the number of user IP networks that can exist at the same time. This must match the VLAN ID configuration on the switch, and can be up to around 4000.

In the VLAN configuration screen options a network must be selected by:

- either choosing an existing network that has already been configured in Bright Cluster Manager, but not `internalnet`
- or it requires specifying the following, in order to create the network:
  - A new network `Name`: default: `vlanhostnet`
  - VLAN Range start: default: `5`
  - VLAN Range end: default: `100`

VXLAN Screen Options

The VXLAN range defines the number of user IP networks that can exist at the same time. While the range can be set to be around 16 million, it is best to keep it to a more reasonable size, such as 50,000, since a larger range slows down Neutron significantly.

An IP network is needed to host the VXLANs and allow the tunneling of traffic between VXLAN endpoints. This requires

- either choosing an existing network that has already been configured in Bright Cluster Manager, but not `internalnet`
- or it requires specifying the following, in order to create the network:
  - A new network `Name`: default: `vxlanhostnet`
  - `Base address`: default: `10.161.0.0`
  - `Netmask bits`: default: `16`

In the VXLAN configuration options the following extra options are suggested, with overrideable defaults as listed:

- `VXLAN Range start`: default: `1`
- `VXLAN Range end`: default: `50000`

VXLAN networking uses a multicast address to handle broadcast traffic in a virtual network. The default multicast IP address that is set, `224.0.0.1`, is unlikely to be used by another application. However, if there is a conflict, then the address can be changed using the CMDaemon `OpenStackVXLANGroup` directive (Appendix C, page 632 of the *Administrator Manual*).

3.1.11 OpenStack Network Isolation interface For Network And Hypervisor Nodes

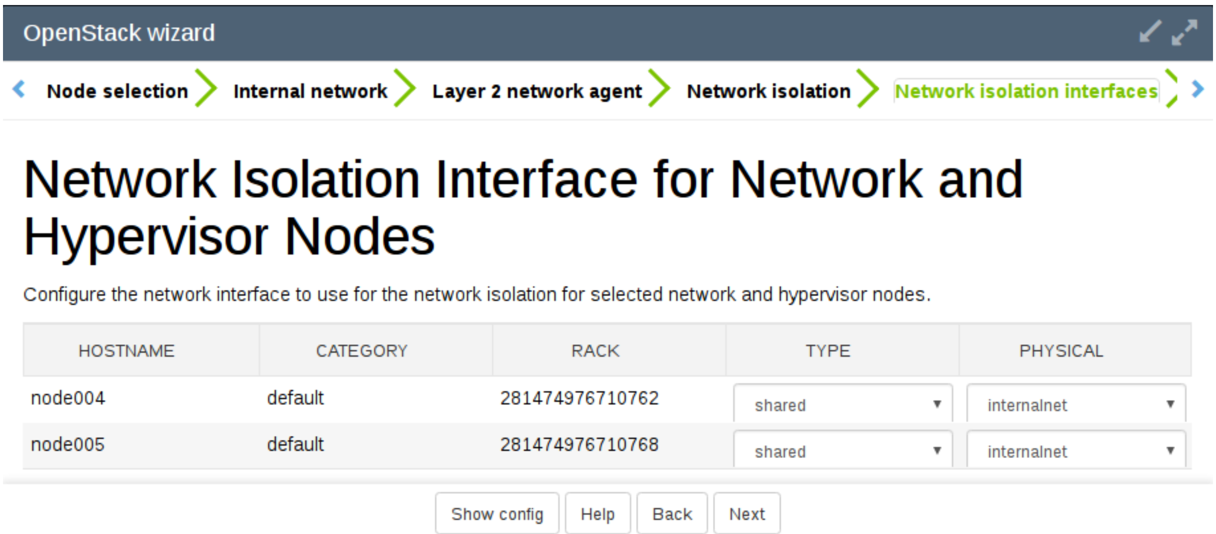

Figure 3.13: OpenStack Network Isolation interface For Network And Hypervisor Nodes Screen

The `Network Isolation interface For Network And Hypervisor Nodes` screen (figure 3.13) sets the network that will be used for the network nodes and hypervisor nodes. These are classed according to whether the network will be shared or dedicated, and the nodes can be text-filtered by column, which is useful when dealing with a large number of nodes.

3.1.12 OpenStack Inbound External Traffic

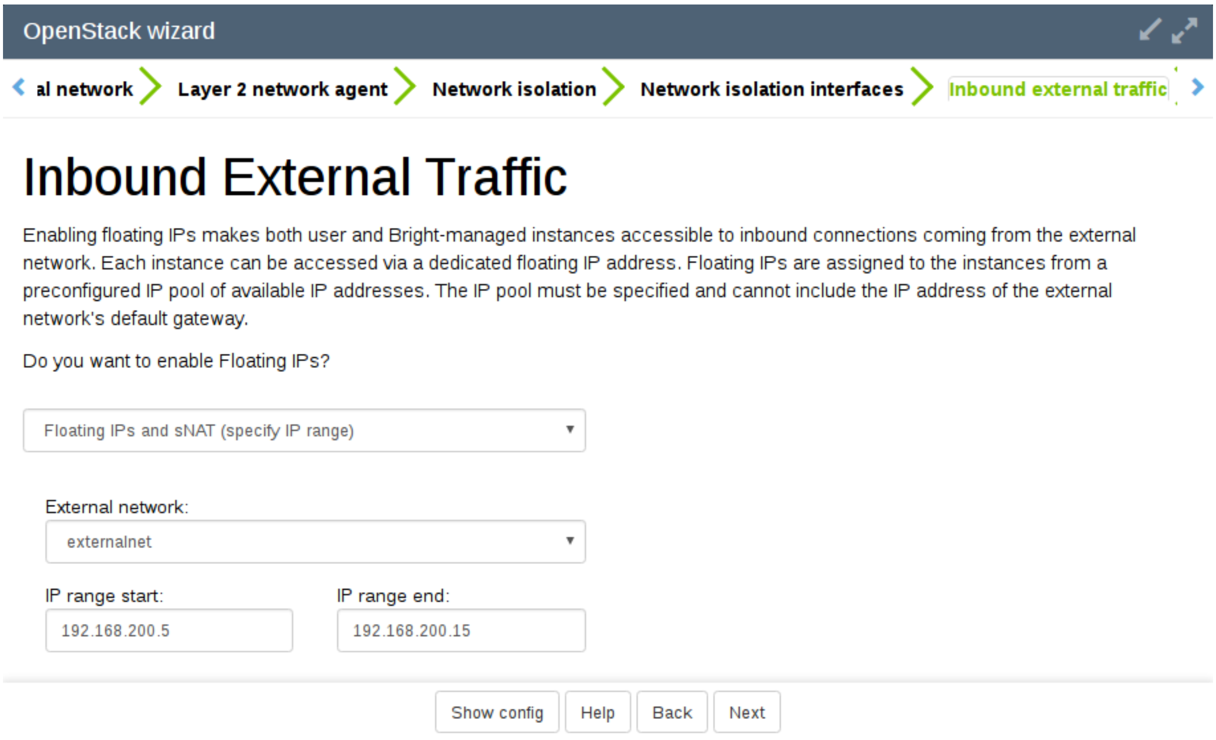

Figure 3.14: OpenStack Inbound External Traffic Screen

The `OpenStack Inbound External Traffic` screen (figure 3.14) allows the administrator to set floating IP addresses. A floating IP address is an address on the external network that is associated with an OpenStack instance. The addresses "float" because they are assigned from an available pool of addresses, to the instance, when the instance requests an address.

3.1.13 OpenStack External Network Interface For Network Node

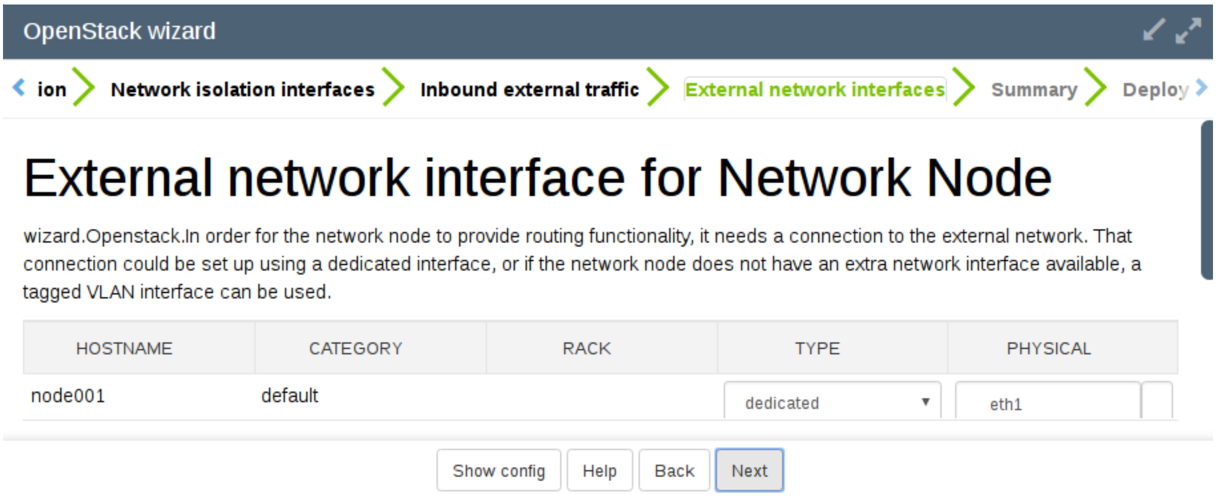

Figure 3.15: OpenStack External Network Interface For Network Node Screen

The `OpenStack External Network Interface For Network Node` screen (figure 3.15) allows the administrator to provide routing between the external network and the network nodes. It can be set up on a dedicated interface. If no spare interface is available on the network node, then if the switch supports it, a tagged VLAN interface can be configured instead.

The nodes can be text-filtered by column, which is useful when dealing with a large number of nodes.

3.1.14 Summary

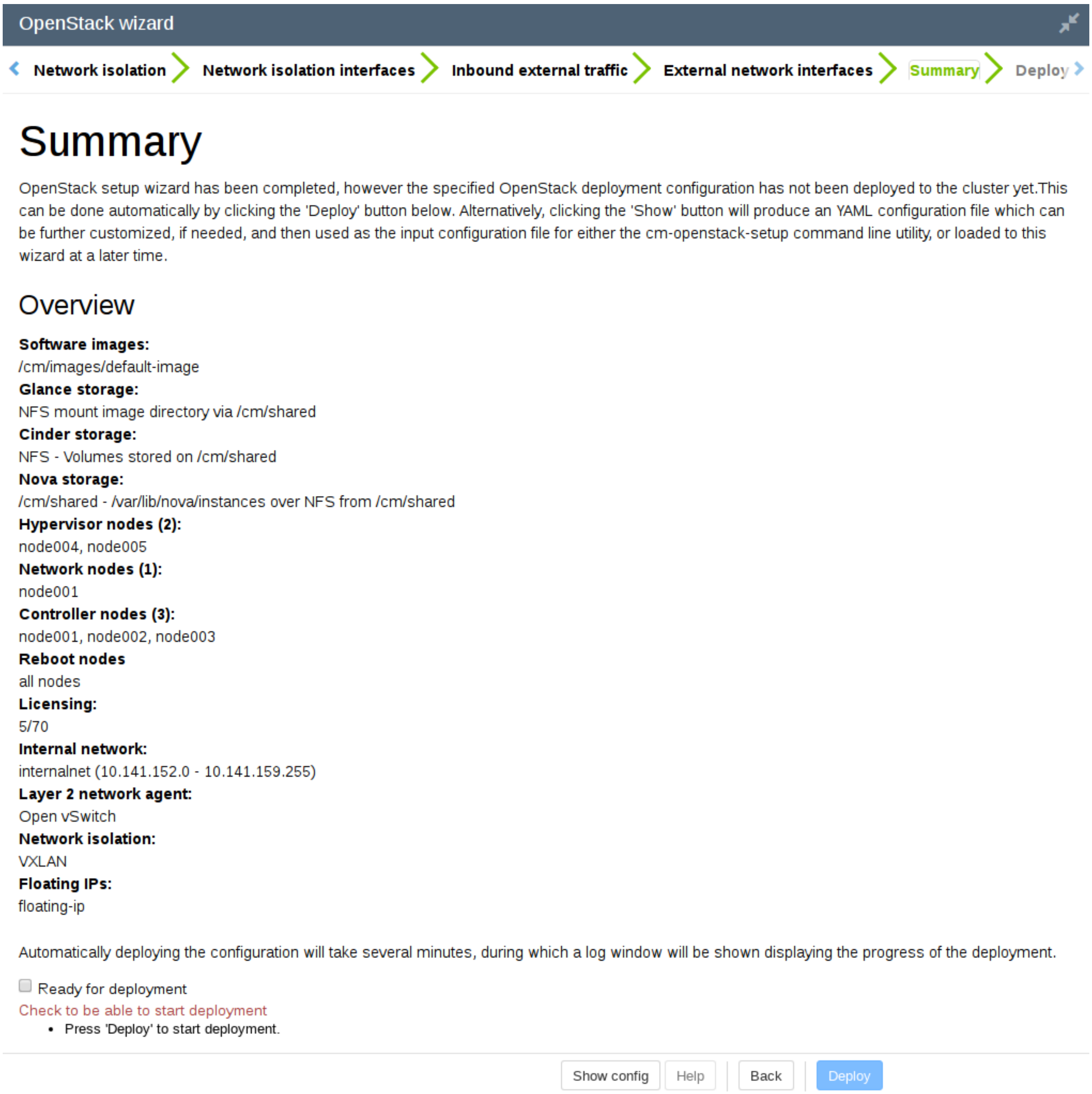

Figure 3.16: Summary Screen

Viewing And Saving The Configuration

The summary screen (figure 3.16) gives a summary of the configuration. The configuration can be changed in Bright View by clicking on a value displayed in the summary screen. Clicking opens the screen in the wizard that is associated with setting that value.

The full configuration is kept in an YAML file, which can be viewed by clicking on the Show Config button. The resulting editable text is shown in figure 3.17.

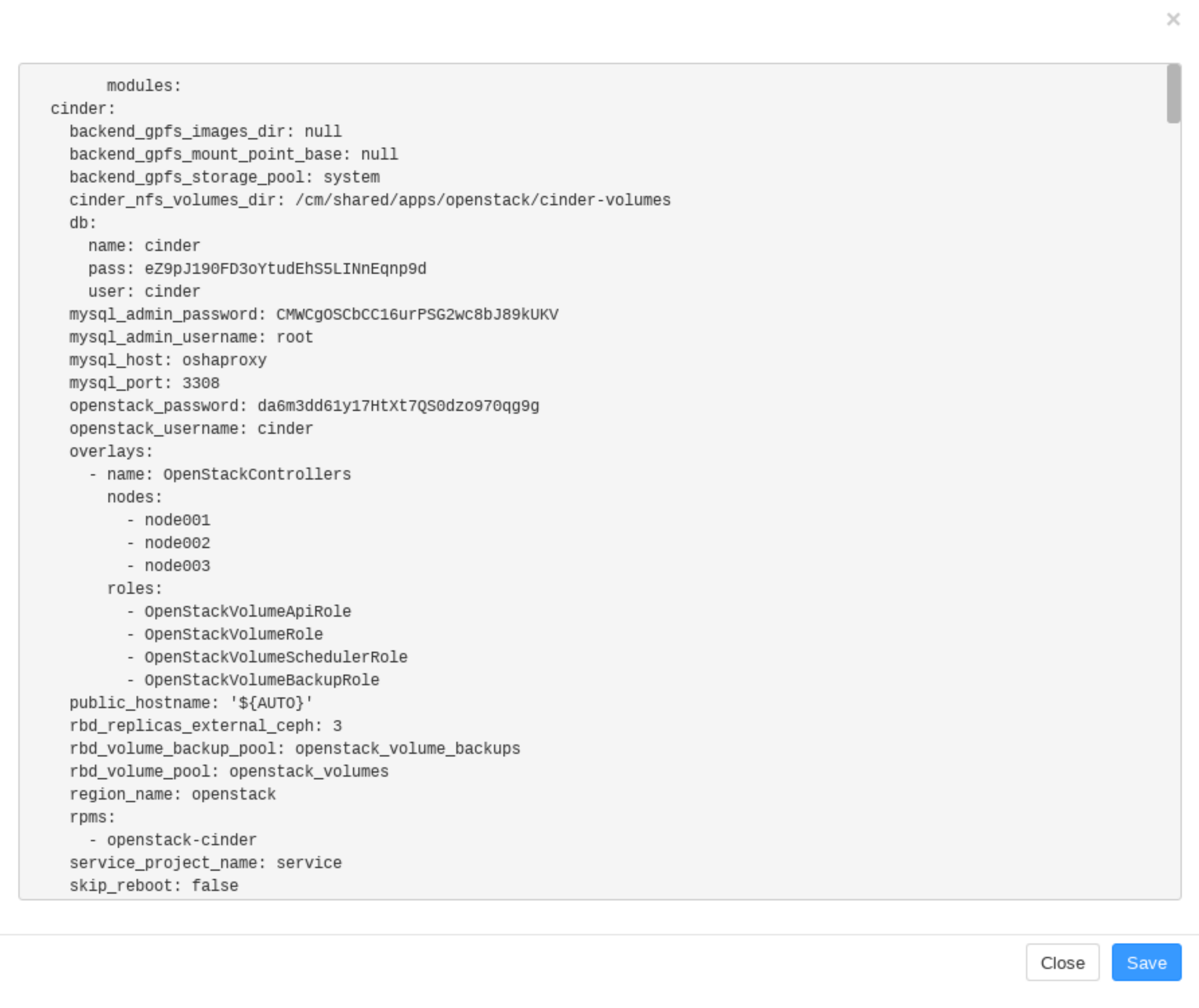

Figure 3.17: OpenStack Configuration Screen

The configuration can be saved with the Save button of figure 3.17.

Using A Saved Configuration And Deploying The Configuration

Using a saved YAML file is possible.

- The YAML file can be used as the configuration starting point for the text-based cm-openstack-setup utility (section 3.2), if run as:

  [root@bright81~]# cm-openstack-setup -c <*YAML file*>

- Alternatively, the YAML file can be deployed as the configuration by launching the Bright View wizard, and then clicking on the Load config button of the first screen (figure 3.2). After loading the configuration, a Deploy button appears.

Clicking the Deploy button that appears in figure 3.2 after loading the YAML file, or clicking the Deploy button of figure 3.16, sets up OpenStack in the background. The direct background progress is hidden from the administrator, and relies on the text-based cm-openstack-setup script (section 3.2). Some log excerpts from the script can be displayed within a Show Log section (figure 3.18).

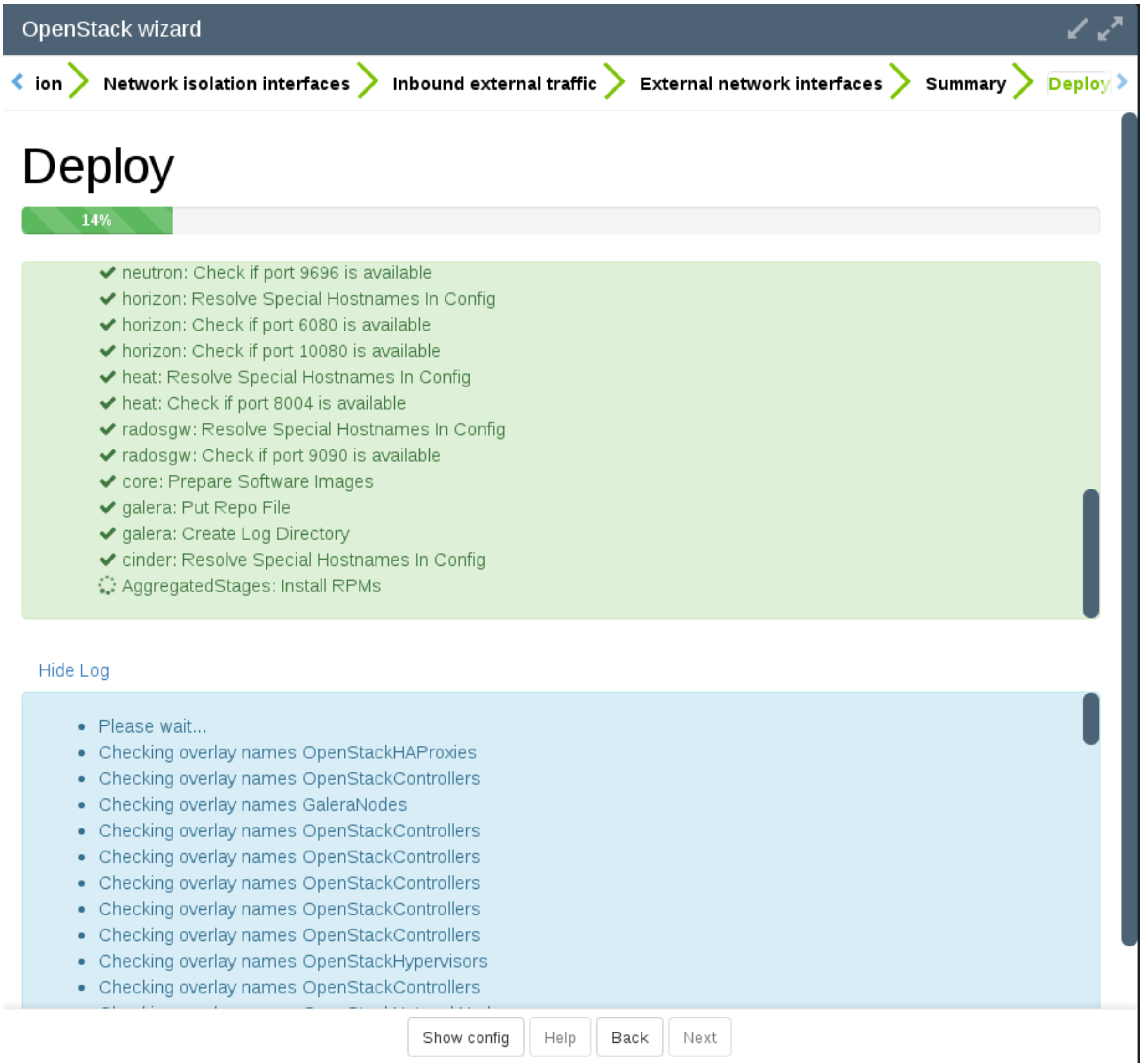

Figure 3.18: OpenStack Deployment Progress Screen

At the end of its run, the cluster has OpenStack set up and running in an integrated manner with Bright Cluster Manager.

The administrator can now configure the cluster to suit the particular site requirements.

3.2 Installation Of OpenStack From The Shell

The Bright View OpenStack installation (section 3.1) actually uses the `cm-openstack-setup` utility during deployment, only the utility is normally invisible. The installation can also be done directly with `cm-openstack-setup`. The `cm-openstack-setup` utility is a less-preferred alternative to the installation of OpenStack from Bright View.

The `cm-openstack-setup` utility is a part of the standard `cluster-tools` package. Details on its use are given in its manual page (`man (8) cm-openstack-setup`). When run, the regular nodes that are to run OpenStack instances are rebooted by default at the end of the dialogs, in order to deploy them.

A prerequisite for running `cm-openstack-setup` is that the head node should be connected to the distribution repositories.

A sample `cm-openstack-setup` wizard session is described next, starting from section 3.2.1. The session runs on a cluster consisting of one head node and one regular node. The wizard can be interrupted gracefully with a `<ctrl-c>`.

3.2.1 Start Screen

Figure 3.19: Start Screen

The start screen (figure 3.19) lets the administrator:

- deploy Bright Cluster Manager OpenStack.
- remove Bright Cluster Manager's OpenStack if it is already on the cluster.
- exit the installation.

Removal removes OpenStack-related database entries, roles, networks, virtual nodes, and interfaces. Images and categories related to OpenStack are however not removed.

A shortcut to carry out a removal from the shell prompt is to run `cm-openstack-setup --remove`. The `preventremoval` setting can be set to `no` for this to work:

Example

```
[root@bright81 ~]# cmsh
[bright81]% openstack
[bright81->openstack[default]]% set preventremoval no; commit; quit
[root@bright81 ~]# cm-openstack-setup --remove
Please wait...
Connecting to CMDaemon
###### WARNING: Setup will attempt to remove the following objects:
...
```

3.2.2 Controller Node Selection

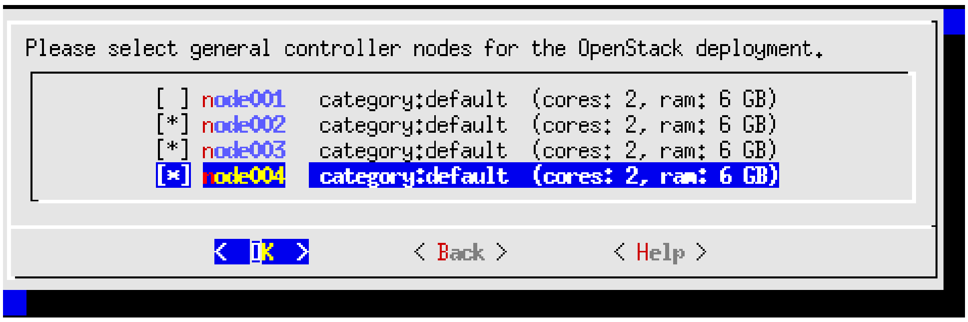

Figure 3.20: Controller Nodes Selection

The controller nodes selection screen (figure 3.20) allows the selection of nodes on which the following services are to run:

- the OpenStack database service
- Heat (orchestration)
- Nova (compute)
- Neutron (networking)
- Swift (object storage)—not deployed by default in Bright Cluster Manager OpenStack edition
- Cinder (block storage)
- Keystone (identity)
- Glance (image service)

Each controller node is required to have a minimum of 2 cores, and a minimum of 8GB of RAM.

3.2.3 Setting The Cloud `admin` Password

Figure 3.21: Cloud `admin` Password Screen

The OpenStack cloud `admin` password screen (figure 3.21) prompts for a password to be entered, and then re-entered, for the soon-to-be-created `admin` user of OpenStack. The `admin` user is mandatory. The password can be changed after deployment.

3.2.4 User Management Configuration Of OpenStack Users

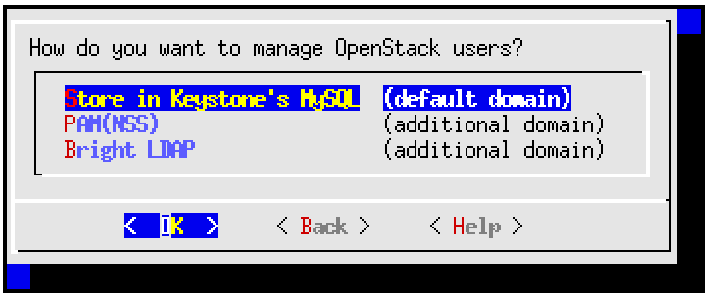

Figure 3.22: User Management Configuration Of OpenStack Users Screen

The user management configuration of OpenStack users screen (figure 3.22) allows the administrator to choose how OpenStack users are to be managed. Options are:

- Managing via Keystone MySQL (default domain)
- Managing via PAM(NSS)

- Using LDAPS as provided by Bright Cluster Manager

Managing via Keystone's MySQL means that OpenStack users, in the default OpenStack domain, are independent of the pre-existing Bright Cluster Manager users.

Managing via PAM(NSS) additionally allows Keystone to use PAM as an identity backend for additional domains. For external identity authentication, PAM(NSS) can be run in a read-only mode.

Managing via Bright Cluster Manager's LDAPS means that OpenStack users, stored in the default OpenStack domain, and independent of the pre-existing Bright Cluster Manager users, are used, and Bright Cluster Manager users are also visible to Keystone, via a read-only access.

3.2.5 Storage Options, Including Ceph

This section (3.2.5) covers the Ncurses `cm-openstack-setup` wizard configuration of storage options, including Ceph.

Glance VM Image Storage

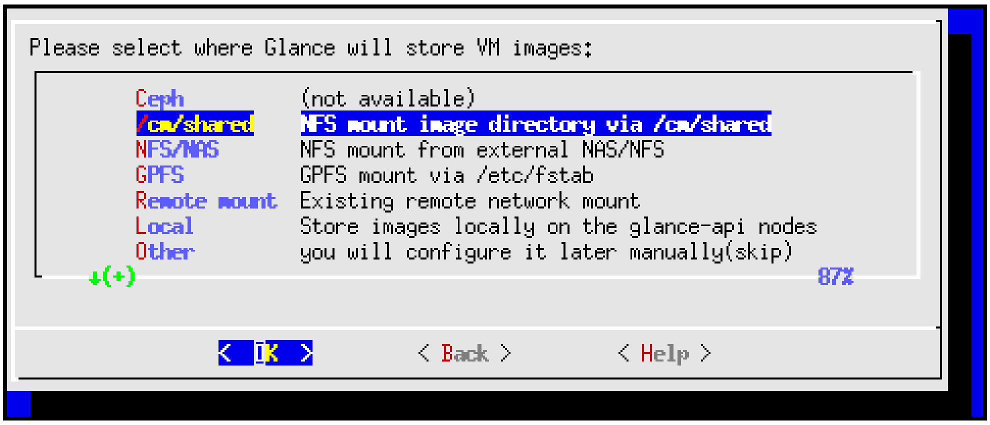

Figure 3.23: Image Storage Options

The image storage screen (figure 3.23) can be used to set the virtual machine storage used.

The storage options are:

- `Ceph` - This is only available as an image storage option, if set up as in Chapter 4.
- `/cm/shared` - The standard Bright Cluster Manager shared NFS directory
- `NFS/NAS` - An external NAS NFS directory
- `GPFS` - A GPFS mount as defined in the `/etc/fstab` configuration.
- `Remote mount` - An existing remote network mount
- `Local` - Images are stored locally on Glance API nodes.
- `Other` - to be configured later (skips this screen)
- `More` - Other backends that are not listed in this menu

Cinder Volume Storage

Figure 3.24: Volume Storage Options

The OpenStack Cinder volume storage screen (figure 3.24) allows the setting of persistent block volume read and write storage.

The storage options are:

- `Ceph` - This is only available as a volume storage option, if set up as in Chapter 4. If set, it uses Ceph's RBD volume driver, and configures a "volume backup" driver to use Ceph.
- `NFS` - Storage is done on `/cm/shared` using the Cinder reference driver. This is not recommended for large-scale production use.
- `None` - to be configured later (skips this screen)

Root And Ephemeral Device Storage With Ceph

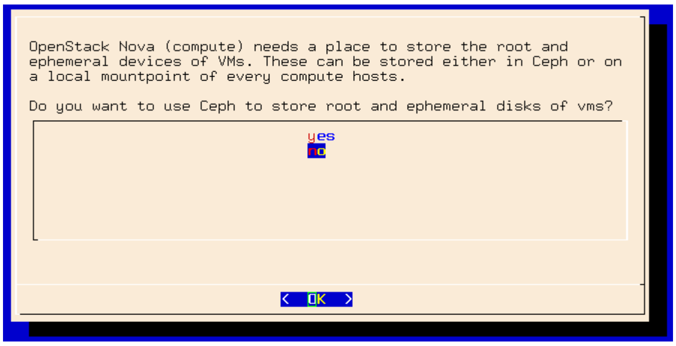

Figure 3.25: Root And Ephemeral Device Storage

Data storage to a root or ephemeral device with Ceph can be enabled by the administrator by using the OpenStack root and ephemeral device storage screen (figure 3.25).

Ceph Object Gateway (Ceph RADOS Gateway)

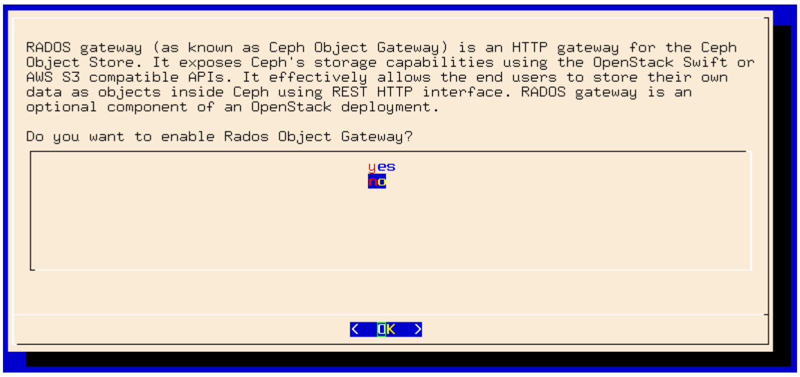

Figure 3.26: Root And Ephemeral Device Storage With Ceph

The Ceph RADOS gateway screen (figure 3.26) lets the administrator set the Ceph RADOS gateway service to run when deployment completes.

3.2.6 Hypervisor Nodes Selection For OpenStack

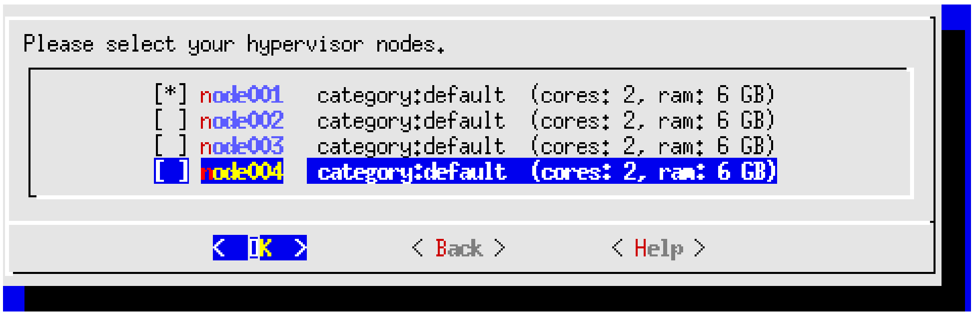

Figure 3.27: Hypervisors To Be Used For OpenStack

The hypervisor nodes selection screen (figure 3.27) lets the administrator set the nodes that will be hypervisors. These are the machines that host the compute nodes, and which are assigned the `OpenStackNovaCompute` role. The set of nodes can be changed on a cluster later on, by managing the node list of the `OpenStackHyperVisors` configuration overlay.

3.2.7 VM Root/Ephemeral Disk Storage

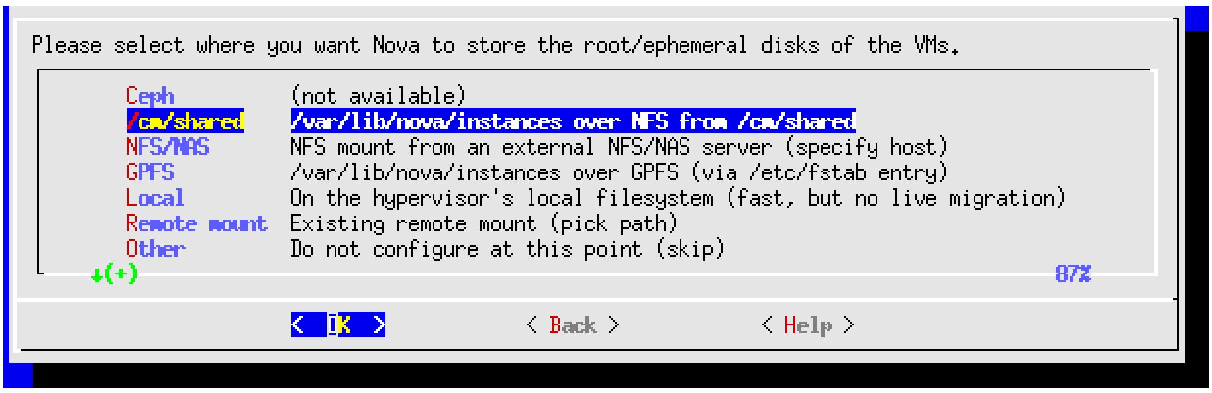

Figure 3.28: Setting Root/Ephemeral VM Disk Storage Location

The VM root/ephemeral disk storage screen (figure 3.28) allows the administrator to tell Nova where to store the root/ephemeral disks. The options are

- `Ceph`: This option is available if Ceph has been configured. By default, `/var/lib/nova/instances` is used.
- `/cm/shared`: The disks can be stored on the hypervisor nodes under the NFS shared directory in `/var/lib/nova/instances`
- `NFS/NAS`: An external NFS/NAS host can be used
- `GPFS`: The disks can be stored on the hypervisor nodes via a GPFS directory specified for `/var/lib/nova/instances` in `/etc/fstab`
- `Local`: The disks can be stored on the local filesystem of the hypervisor. This is avoids network lag, but also does not permit migration.
- `Remote mount`: A path to an existing remount mount point.
- `Other`: Skip (configure later maybe)
- `More`: Suggests alternatives

3.2.8 Network Overlay Technology Used For OpenStack

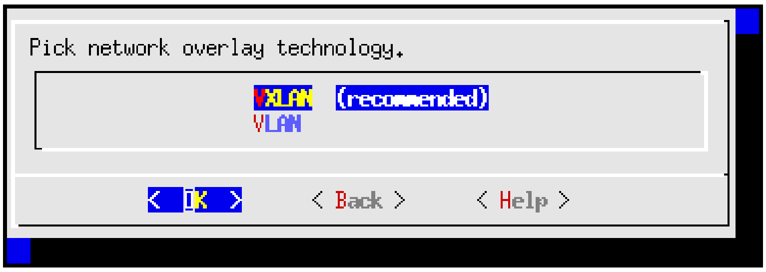

Figure 3.29: Network Overlay Used For OpenStack

The network overlay technology screen (figure 3.29) allows the administrator to choose what kind of network isolation type should be set for the user networks.

3.2.9 Setting The Virtual Network Name

Figure 3.30: Creating The Virtual Network

The virtual network is the hosting network for OpenStack end user networks. The virtual networks screen (figure 3.30) allows the administrator to configure a virtual network to host the end user networks. By default, if needed, the network to be created is named `vlanhostnet` for a VLAN network, and `vxlanhostnet` for a VXLAN network. An existing VXLAN or VLAN network can be selected.

3.2.10 Setting The Network Details For The Virtual Network

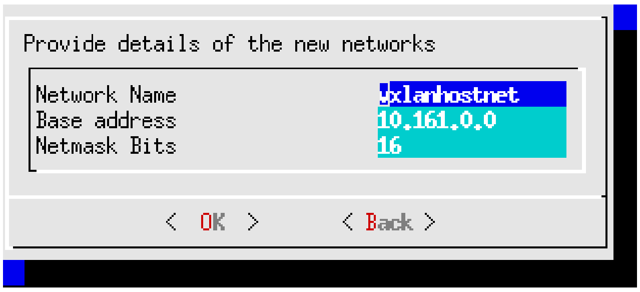

Figure 3.31: Setting The Network Details Of The Virtual Network

The virtual network is the hosting network for end user networks. If it does not have its details configured as yet, then the network details screen (figure 3.31) allows the administrator to set the base address and netmask bits for the virtual network.

3.2.11 Setting The Network Nodes

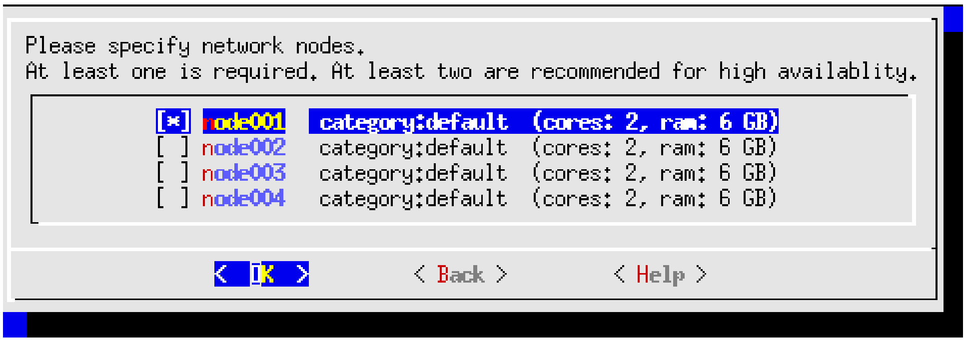

Figure 3.32: Setting The Network Nodes

The network node selection screen (figure 3.32) allows the administrator to set network nodes. The network nodes run OpenStack networking components from OpenStack neutron. A reasonable rule-of-thumb is to have 1 network node per 10 hypervisor nodes. Network nodes and compute nodes can be combined.

To use Floating IPs or sNAT, network nodes must be connected to the external network.

3.2.12 Floating IPs And sNAT

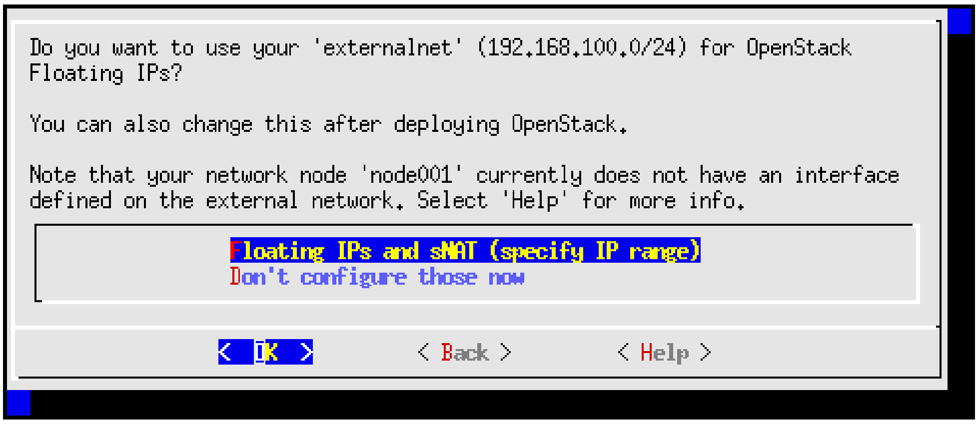

Figure 3.33: Floating IPs

The floating IPs screen (figure 3.33) lets the administrator allow floating IPs to be configured on the external network. This allows instances within OpenStack to be accessed from the external network. Floating IPs can also be configured after OpenStack has been set up.

A note is shown in the dialog if the network node does not have an external network interface. Creating the external network interface is possible at this point or later, using `cmsh` for example.

3.2.13 External Network Floating IP Range

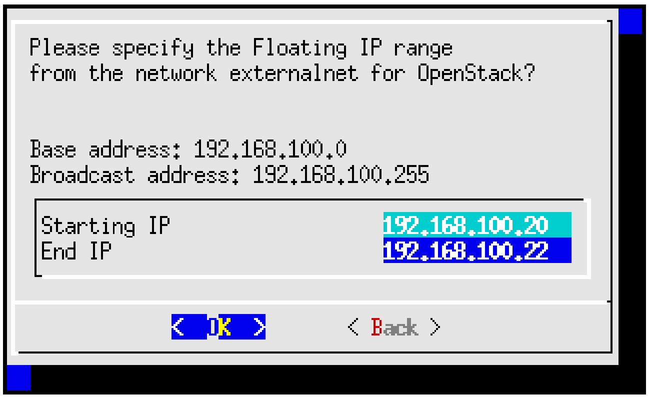

Figure 3.34: External Network: Floating IP Range

If floating IPs are to be configured by the wizard, then the floating IP range screen figure 3.34 allows the administrator to specify the floating IP address range on the external network.

3.2.14 External Network Interface Creation

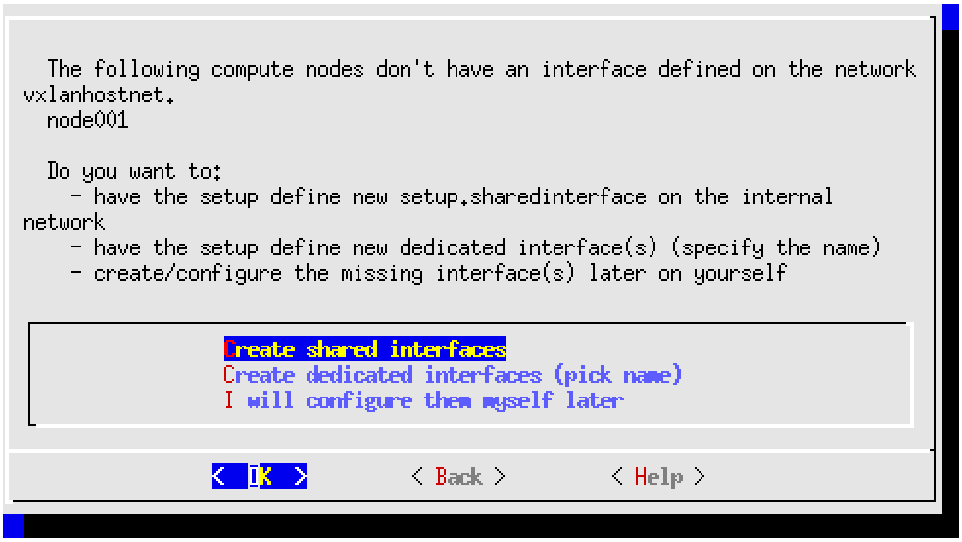

Figure 3.35: External Network: Interface Creation

If floating IPs are to be configured by the wizard, then the external network interface creation screen (figure 3.35) allows the administrator to create a network interface. The interface is created on each network node that is missing an interface to the external network.

The interface can be

- a shared interface: this uses the internal network for virtual networking
- a dedicated interface: this uses a dedicated network with its associated dedicated interface. The device must exist on the network node in order for the interface to be created.

The interface creation step can be skipped and carried out after OpenStack deployment, but OpenStack may not run properly because of this. Alternatively, if each network node has special needs, then each interface can be set up before running the wizard.

3.2.15 Saving The Configuration

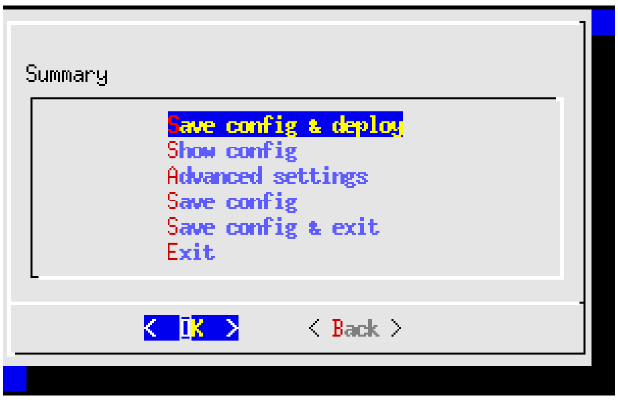

Figure 3.36: Viewing And Saving The Configuration

The screen for saving the configuration (figure 3.36) allows the administrator to view the configuration with the `Show` option. The configuration that has been prepared by the wizard can be seen with the `Show config` option, and using the `<Page Up>` and `<Page Down>` keys to scroll up and down.

The configuration options can also be saved with the various save options:

- `Save config & deploy`: Saves, and after saving carries out the text-based deployment stage of the installation.
- `Save`: Saves, and stays within the Ncurses dialog. The deployment can be carried out later from a saved configuration.
- `Save config & exit`: Saves, and then exits the Ncurses dialog. The deployment can be carried out later from a saved configuration.

Saving saves the configration as a YAML configuration file, by default `cm-openstack-setup.conf`, in the directory under which the wizard is running. This file can be used as the input configuration file for the `cm-openstack-setup` utility using the `-c` option.

Most administrators run `Save config & deploy`, and the deployment run takes place (section 3.2.16). Some administrators may however wish to modify some OpenStack component settings.

The OpenStack Components Advanced Settings Screens

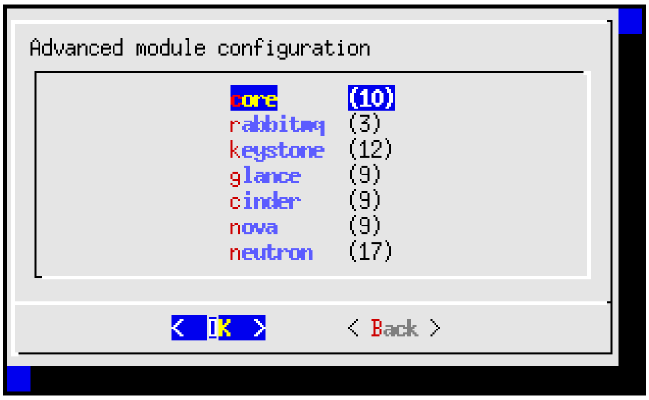

Figure 3.37: Advanced Options

The advanced settings screen (figure 3.37) allows an administrator to set up OpenStack components with some advanced options. For example, values for the passwords and ports used by various OpenStack services can be modified. These values can also be altered from within `cmsh` after deployment.

The components that can be dealt with in the advanced settings screen are `core`, `rabbitmq`, `keystone`, `glance`, `cinder`, `nova`, and `neutron` (figures 3.38– 3.44).

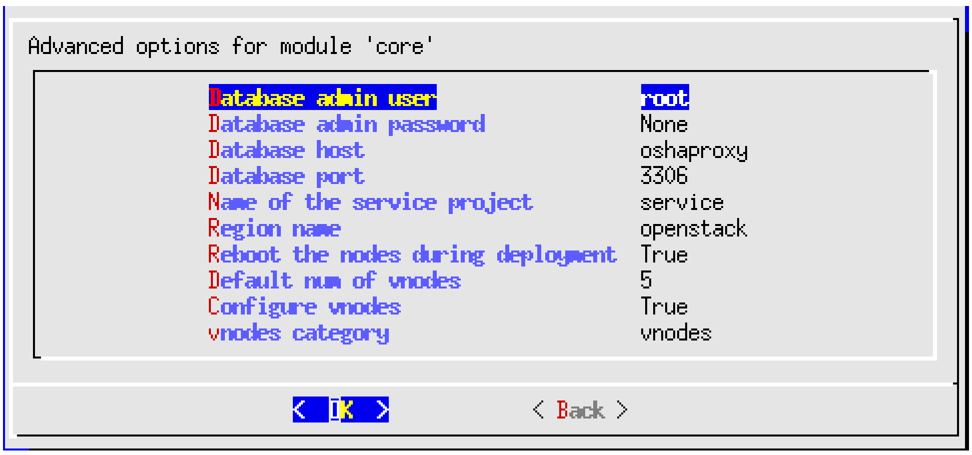

Figure 3.38: Advanced Options: Core

Figure 3.39: Advanced Options: RabbitMQ

Figure 3.40: Advanced Options: Keystone

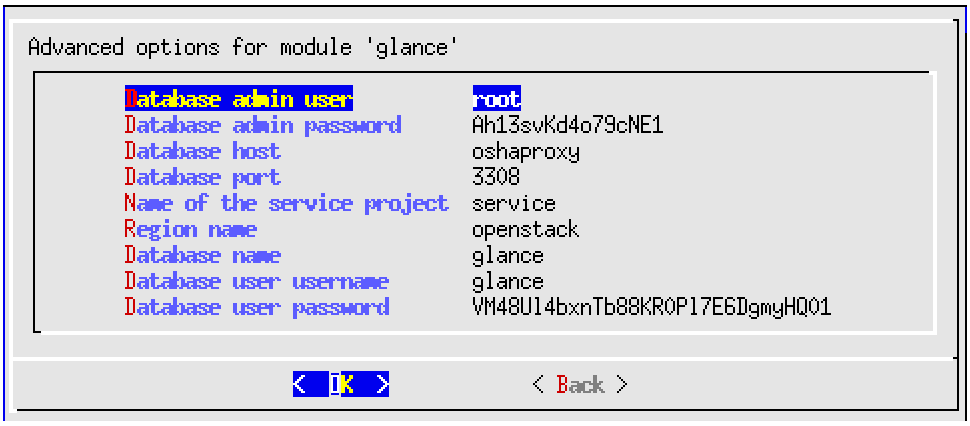

Figure 3.41: Advanced Options: Glance

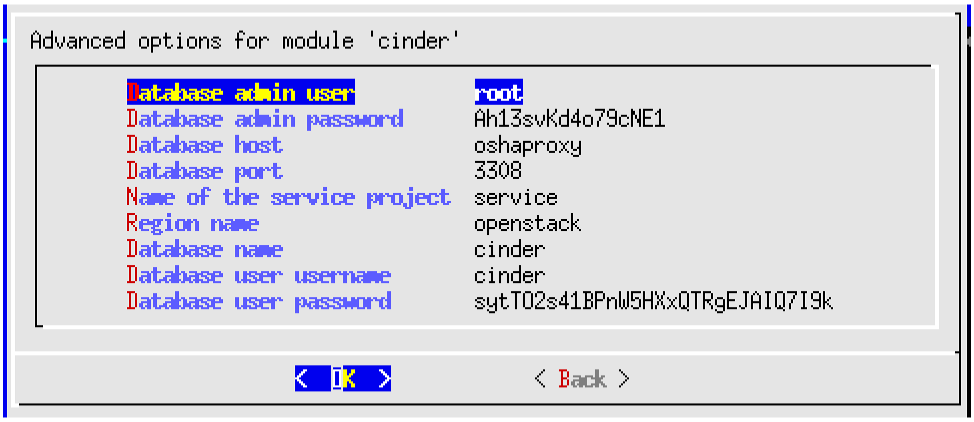

Figure 3.42: Advanced Options: Cinder

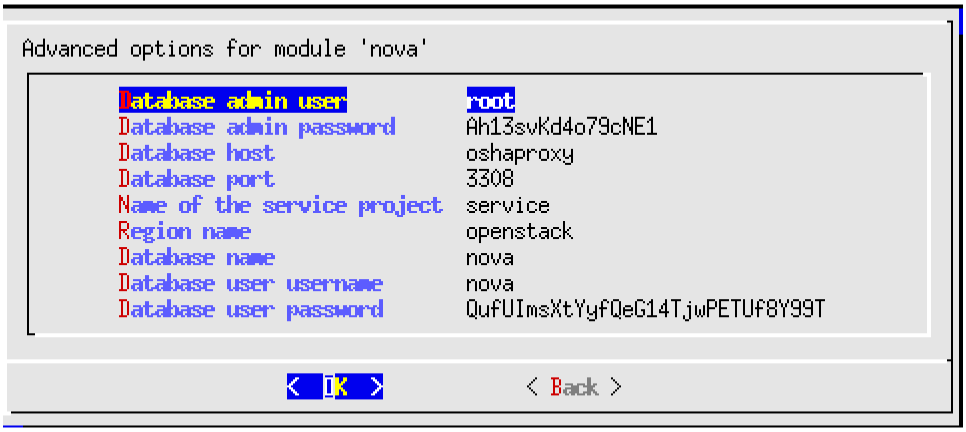

Figure 3.43: Advanced Options: Nova

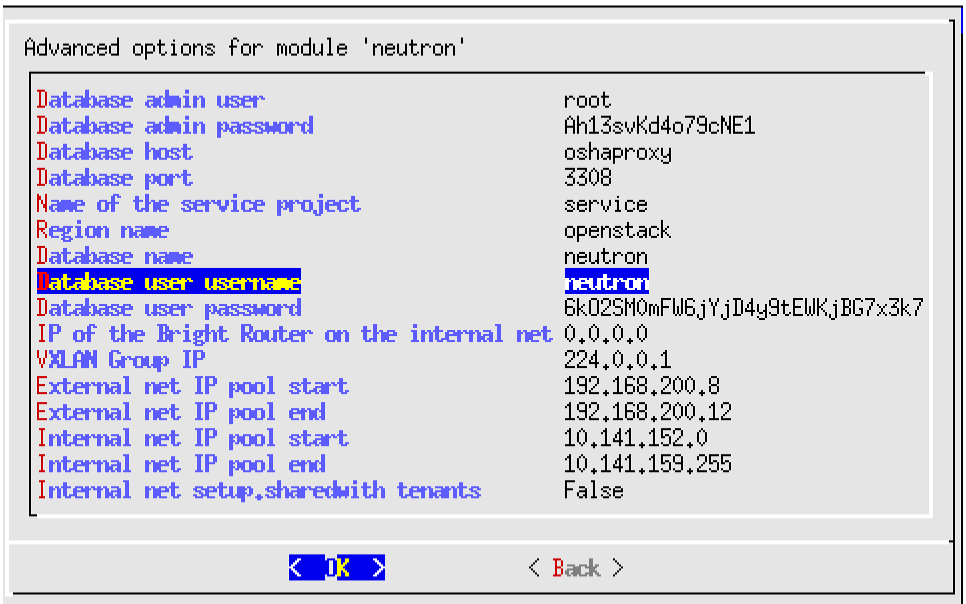

Figure 3.44: Advanced Options: Neutron

3.2.16 The Deployment Run—An Overview

The deployment displays a lengthy text run. An elided version follows:

```
Checking overlay names OpenStackControllers
Checking overlay names GaleraNodes
Checking overlay names OpenStackHypervisors
Checking overlay names OpenStackControllers
Checking overlay names OpenStackNetworkNodes
...
Executing 264 stages
################## Starting execution for 'Bright OpenStack'
  - core
  - galera
  - rabbitmq
  - keystone
  - glance
  - cinder
  - nova
  - neutron
  - horizon
  - heat
  - radosgw
## Progress: 0
#### stage: core: Resolve Special Hostnames In Config
#### stage: core: Precheck System
Checking system configuration
## Progress: 1
#### stage: core: Precheck OpenStack
#### stage: core: Check Networking
#### stage: core: Precheck License
...
Initializing Heat certificate
## Progress: 43
#### stage: heat: Init certificate in software image: '/cm/images/default-image'
#### stage: core: Set Deployment Phase
#### stage: AggregatedStages: Reboot Nodes
All affected nodes: ['node003', 'node002', 'node001', 'node005', 'node004']
All nodes to be rebooted: node003, node002, node001, node005, node004
Node has been rebooted node003
Node has been rebooted node002
Node has been rebooted node001
Node has been rebooted node005
Node has been rebooted node004
Press ctrl+c to abort waiting and continue with deployment
Waiting for nodes to start reboot
Going to wait up to 60 minutes for the nodes to come back up.
Waiting for 5 nodes to come back up
Waiting for 4 nodes to come back up
Waiting for 1 node to come back up
All 5 nodes came back up.
## Progress: 44
#### stage: core: Set Deployment Phase
#### stage: core: Determine Public Host Deployment
...
## Progress: 93
#### stage: heat: Add OpenStack User
```

```
#### stage: heat: Open Shorewall Port On Headnode
Opening port 8004 in Shorewall for OpenStack Heat
Restarting shorewall
#### stage: heat: Create Service And Endpoint
Creating service heat
Creating endpoint: 'http://oshaproxy:8004/v1/$(tenant_id)s'
Creating endpoint: 'http://oshaproxy:8004/v1/$(tenant_id)s'
Creating endpoint: 'http://10.2.61.198:8004/v1/$(tenant_id)s'
## Progress: 94
#### stage: heat: Create Api Objects
#### stage: heat: Assign Nodes Roles To Overlay
Assigning role OpenStackOrchestrationApiRole to overlay
Assigning role OpenStackOrchestrationRole to overlay
## Progress: 95
#### stage: core: Set Deployment Phase
#### stage: core: Add Image
#### stage: core: Add Image
## Progress: 96
#### stage: core: Add Image
#### stage: core: Add Image
#### stage: core: Add Image
## Progress: 97
#### stage: core: Get Image UUID
#### stage: core: Finalize OpenStack
## Progress: 98
#### stage: core: Configure Sec Groups
#### stage: keystone: Configure CMDaemon Post Deployment
#### stage: keystone: Enable keystone token flush timer
## Progress: 99
#### stage: nova: Wait For Service To Be Operational
Waiting for nova
#### stage: nova: Patch Flavors
## Progress: 100
#### stage: nova: Running: 'nova-manage --config-file /etc/nova/nova.conf cell_v2
 discover_hosts --verbose'

Took:     26:40 min.
Progress: 100/100
################## Finished execution for 'Bright OpenStack', status: completed

Bright OpenStack finished!
[root@bright81 ~]#
```

3.2.17 The State After Running `cm-openstack-setup`

At this point, the head node has OpenStack installed on it.

However, a regular node that has been configured with the OpenStack compute host role, ends up with OpenStack deployed on it only after the operating system running on the node is updated with the installed OpenStack software, and the newly-configured interfaces are set up according to the specified configuration.

For simplicity, the update is done on the regular nodes by a reboot action by default, as shown in the preceding output, in the text that follows "`All nodes to be rebooted`".

Trying to do it without a reboot by using `imageupdate` (section 5.6 of the *Administrator Manual*) is not recommended, because interfaces typically do change along with the updates, except for some specially configured cases. In the case of these special configurations, the setup wizard can be set to reboot only the controler node using Bright View (figure 3.4)

The administrator can further configure the cluster to suit requirements. Setting up a secondary node for high availability is discussed in section 3.3, while the rest of the manual describes other configurations.

3.3 Adding A Secondary Node To An Existing OpenStack Cluster For High Availability

On an existing OpenStack Bright cluster, the public endpoints point to the public IP address of the head node.

If a secondary head node is added to the cluster to provide high availability (Chapter 15 of the *Administrator Manual*), then some downtime is required. This is because after the secondary head node is synced from the primary and finalized, the public endpoints need to be changed to point to the shared public IP address, instead of the public IP address of the primary head node, and the OpenStack services then need to be restarted.

The endpoints can viewed and changed from `cmsh`, with a session similar to the following:

Example

```
[bright81->openstack[default]->endpoints]% list -f name:20,service:10,url:40,interface:10
name (key)           service    url                                      interface
-------------------- ---------- ---------------------------------------- ----------
volume:adminv1       cinder (8+ http://oshaproxy:8776/v1/$(tenant_id)s   Admin
volume:internalv1    cinder (8+ http://oshaproxy:8776/v1/$(tenant_id)s   Internal
volume:publicv1      cinder (8+ http://10.2.61.198:8776/v1/$(tenant_id)s Public
volume:adminv2       cinderv2 + http://oshaproxy:8776/v2/$(tenant_id)s   Admin
volume:internalv2    cinderv2 + http://oshaproxy:8776/v2/$(tenant_id)s   Internal
volume:publicv2      cinderv2 + http://10.2.61.198:8776/v2/$(tenant_id)s Public
volume:adminv3       cinderv3 + http://oshaproxy:8776/v3/$(tenant_id)s   Admin
volume:internalv3    cinderv3 + http://oshaproxy:8776/v3/$(tenant_id)s   Internal
volume:publicv3      cinderv3 + http://10.2.61.198:8776/v3/$(tenant_id)s Public
glance:admin         glance (6+ http://oshaproxy:9292                    Admin
glance:internal      glance (6+ http://oshaproxy:9292                    Internal
glance:public        glance (6+ http://10.2.61.198:9292                  Public
heat:admin           heat (831+ http://oshaproxy:8004/v1/$(tenant_id)s   Admin
heat:internal        heat (831+ http://oshaproxy:8004/v1/$(tenant_id)s   Internal
heat:public          heat (831+ http://10.2.61.198:8004/v1/$(tenant_id)s Public
keystone:admin       keystone + http://oshaproxy:35357/v3                Admin
keystone:internal    keystone + http://oshaproxy:5000/v3                 Internal
keystone:public      keystone + http://10.2.61.198:5000/v3               Public
networking:admin     neutron (+ http://oshaproxy:9696/                   Admin
networking:internal  neutron (+ http://oshaproxy:9696/                   Internal
networking:public    neutron (+ http://10.2.61.198:9696/                 Public
placement:admin      nova (38f+ http://oshaproxy:8778                    Admin
placement:internal   nova (38f+ http://oshaproxy:8778                    Internal
placement:public     nova (38f+ http://10.2.61.198:8778                  Public
compute:admin        nova (f7a+ http://oshaproxy:8774/v2/$(tenant_id)s   Admin
compute:internal     nova (f7a+ http://oshaproxy:8774/v2/$(tenant_id)s   Internal
compute:public       nova (f7a+ http://10.2.61.198:8774/v2/$(tenant_id)s Public
[bright81->openstack[default]->endpoints]% use volume:publicv1
[bright81...endpoints[volume:publicv1]]% set url "http://<shared external IP>:8776/v2/$(tenant_id)s"
[bright81->openstack[default]->endpoints*[volume:publicv1*]]% commit
```

In the preceding example, the `-f` option is used with the `list` command (page 28 of the *Administrator Manual*) to reduce the list output format to something easier to look over. Also in the example, when setting the `publicv1` URL, the text <*shared external IP*> should be replaced with the actual value of the shared public external IP address.

4

Ceph Installation

4.1 Ceph Introduction

Ceph, at the time of writing, is the recommended storage software for OpenStack for serious use. The Ceph RADOS Gateway is a drop-in replacement for Swift, with a compatible API. Ceph is the recommended backend driver for Cinder, Glance and Nova.

The current chapter discusses

- The concepts and required hardware for Ceph (section 4.1)
- Ceph installation and management (section 4.2)
- RADOS GW installation and management (section 4.4)

4.1.1 Ceph Object And Block Storage

Ceph is a distributed storage software. It is based on an object store layer called RADOS (Reliable Autonomic Distributed Object Store), which consists of Ceph components called OSDs (Object Storage Daemons) and MONs (Monitoring Servers). These components feature heavily in Ceph. OSDs deal with storing the objects to a device, while MONs deal with mapping the cluster. OSDs and MONs, together carry out object storage and block storage within the object store layer. The Ceph Manager daemon (MGR) runs alongside monitor daemons, to provide additional monitoring and interfaces to external monitoring and management systems. The stack diagram of figure 4.1 illustrates these concepts.

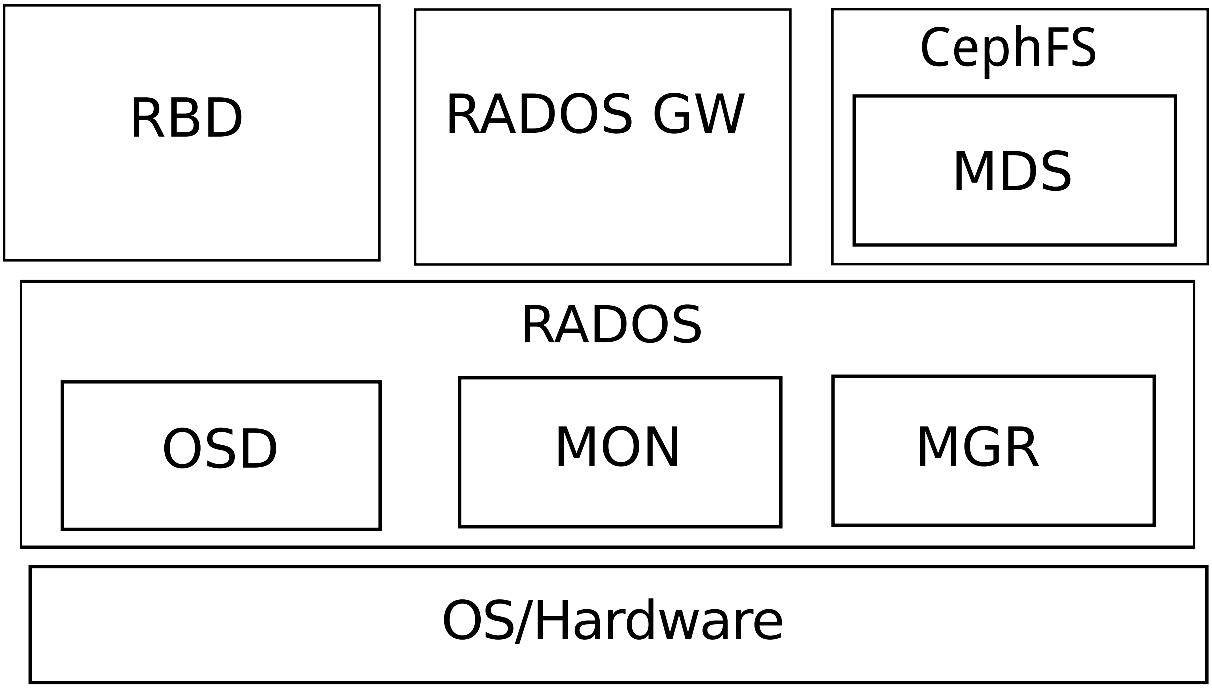

Figure 4.1: Ceph Concepts

On top of the object store layer are 3 kinds of access layers:

1. **Block device access:** RADOS Block Device (RBD) access can be carried out in two slightly different ways:
   (i) via a Linux kernel module based interface to RADOS. The module presents itself as a block device to the machine running that kernel. The machine can then use the RADOS storage, that is typically provided elsewhere.
   (ii) via the `librbd` library, used by virtual machines based on `qemu` or `KVM`. A block device that uses the library on the virtual machine then accesses the RADOS storage, which is typically located elsewhere.

2. **Gateway API access:** RADOS Gateway (RADOS GW) access provides an HTTP REST gateway to RADOS. Applications can talk to RADOS GW to access object storage in a high level manner, instead of talking to RADOS directly at a lower level. The RADOS GW API is compatible with the APIs of Swift and Amazon S3.

3. **Ceph Filesystem access:** CephFS provides a filesystem access layer. A component called MDS (Metadata Server) is used to manage the filesystem with RADOS. MDS is used in addition to the OSD and MON components used by the block and object storage forms when CephFS talks to RADOS.

4.1.2 Ceph Storage Backends

OSDs have a choice of two storage backends for managing their data. These are BlueStore and FileStore.

BlueStore

BlueStore is a special-purpose storage backend designed specifically for managing data on disk for Ceph OSD workloads. It is the default, and recommended, backend for the Ceph version 12.2.*x* series. This series is a long term stable release series, and is also known as the Luminous release series.

BlueStore consumes raw block devices or partitions. In contrast to the legacy FileStore approach, BlueStore avoids any intervening layers of abstraction that may limit performance or add complexity.

BlueStore does however, by its design, in contrast to FileStore, require at least an extra volume on the node the OSD runs on. As recommended for the initial Ceph Luminous release, Bright Cluster Manager uses `ceph-disk` to manage BlueStore devices.

FileStore

Prior to Luminous the only storage backend option was FileStore. FileStore is now the legacy approach to storing objects in Ceph. It relies on a standard file system, which is normally XFS. FileStore is well-tested and widely used in production. However it does suffer from many performance deficiencies due to its overall design and reliance on a traditional file system for storing object data.

Though it is technically possible to store Ceph data alongside other data when using FileStore, it is preferred that dedicated block devices (disks) are used.

Bright Cluster Manager supports both storage backends. In the following sections the storage backends are described in some more detail.

Additional information can be found at `http://docs.ceph.com/docs/luminous/rados/configuration/storage-devices/`.

4.1.3 Ceph Software Considerations Before Use

Recommended Filesystem For Legacy FileStore

BlueStore is the recommended storage backend for Ceph. BlueStore requires dedicated block devices (disks) that are fully managed by ceph-disk. The legacy FileStore backend, on the other hand, stores the data directly on a regular file system.

If using FileStore, then recommended file system is XFS, due to its stability, ability to handle extreme storage sizes, and its intrinsic ability to deal with the significant sizes of the extended attributes required by Ceph.

The nodes that run OSDs are typically regular nodes. Within the nodes, the storage devices used by FileStore OSDs automatically have their filesystems configured to be of the XFS type during the installation of Ceph with Bright Cluster Manager.

Use Of `datanode` For The Protection Of Ceph Data

OSD nodes store the actual data contents of the Ceph cluster. Ceph Monitor nodes also store some data content that is essential for the operation of the Ceph cluster. The devices of these nodes that store such content need protection from being wiped during the reprovisioning that takes place during a reboot of regular nodes.

The recommended way to protect storage devices from being wiped is to set the `datanode` property of their node to `yes` (page 164 of the *Administrator Manual*).

The `datanode` property is automatically set for Monitor and OSD nodes during installation of Ceph with Bright Cluster Manager.

Use Of Slurm On OSD Nodes

Ceph can be quite demanding of the network and I/O. Running Slurm jobs on an OSD node is therefore not recommended. In addition, if Slurm roles are to be assigned to nodes that have OSD roles, then the default ports 6817 and 6818 used by Slurm can conflict with the default range 6800-7300 used by the Ceph OSD daemons. If there is a need to run Slurm on an OSD node then it is necessary to arrange it so that the ports used do not conflict with each other. During installation, a warning is given when this conflict is present.

4.1.4 Hardware For Ceph Use

An absolute minimum installation: can be carried out on two nodes, where:

- 1 node, the head node, runs one Ceph Monitor and the first OSD.
- 1 node, the regular node, runs the second OSD.

This is however not recommended, or even supported by Bright Cluster Manager. Reasons for why this is not recommended are:

- If the Ceph monitor crashes, and there is no other Ceph monitor running, then Ceph cannot function, and data could be lost.
- The first OSD on the head node requires its own Ceph-compatible filesystem. If that filesystem is not provided, then Ceph on the cluster will run, but in a degraded state.
- Running a monitor service on the same host as an OSD may impair performance due to fsync issues with the kernel.

Using such a system to try to get familiar with how Ceph behaves in a production environment with Bright Cluster Manager is unlikely to be worthwhile.

A more useful minimum: if there is a node to spare, then it is possible to installing Ceph over 3 nodes as follows:

- 1 node, the head node, runs one Ceph Monitor.
- 1 node, the regular node, runs the first OSD.
- 1 more node, also a regular node, runs the second OSD.

In this case the OSD pool default size should be set to 2 in the Global OSD Settings (figure 4.10).

Although useful for some testing purposes, this is again not a production system, due to the possible loss of data as well as loss of service if the single Ceph Monitor has issues. This can therefore also not be regarded as a good test cluster.

For production use: a redundant number of Ceph Monitor servers is recommended. This is because Ceph Monitors are crucial to Ceph operations. Since the number of Ceph Monitoring servers must be odd, then at least 3 Ceph Monitor servers, with each on a separate node, are recommended for production purposes. The recommended minimum of nodes for production purposes is then 5:

- 2 regular nodes running OSDs.
- 2 regular nodes running Ceph Monitors.
- 1 head node running a Ceph Monitor.

Drives usable by Ceph: Ceph OSDs can use any type of disk that presents itself as a block device in Linux. This means that a variety of drives can be used.

4.2 Ceph Installation With `cm-ceph-setup`

Ceph installation for Bright Cluster Manager can be carried out with the Ncurses-based `cm-ceph-setup` utility. It is part of the `cluster-tools` package that comes with Bright Cluster Manager. If the Ceph packages are not already installed, then the utility is able to install them for the head and regular nodes, assuming the repositories are accessible, and that the package manager priorities are at their defaults.

4.2.1 Ceph Installation: The Configuration Stage

The `cm-ceph-setup` utility can be run as root from the head node.

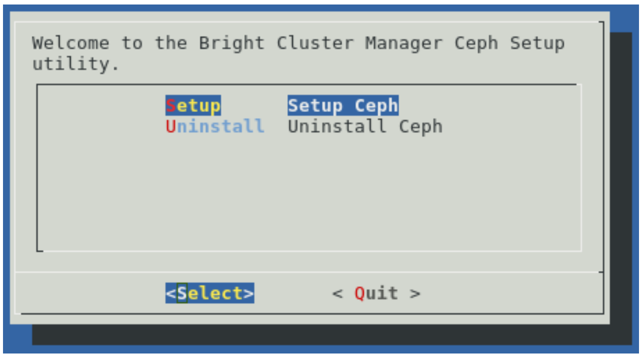

Figure 4.2: Ceph Installation Welcome

At the welcome screen (figure 4.2), the administrator may choose to

- Set up Ceph
- Remove Ceph if it is already installed.

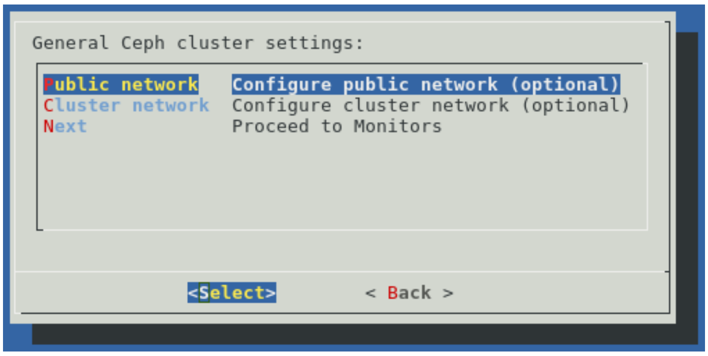

Figure 4.3: Ceph Installation General Cluster Settings

If the setup option is chosen, then a screen for the general Ceph cluster settings (figure 4.3) is displayed. The general settings can be adjusted via subscreens that open up when selected. The possible general settings are:

- `Public network`: This is the network used by Ceph Monitoring to communicate with OSDs. For a standard default Type 1 network this is `internalnet`.
- `Cluster network`: This is the network used by OSDs to communicate with each other. For a standard default Type 1 network this is `internalnet`.

Network Types are discussed in section 3.3.7 of the *Installation Manual*.

Selecting the `Next` option in figure 4.3 continues on with the next major screen of the setup procedure, and displays a screen for Ceph Monitors configuration (figure 4.4).

Ceph Monitors Configuration

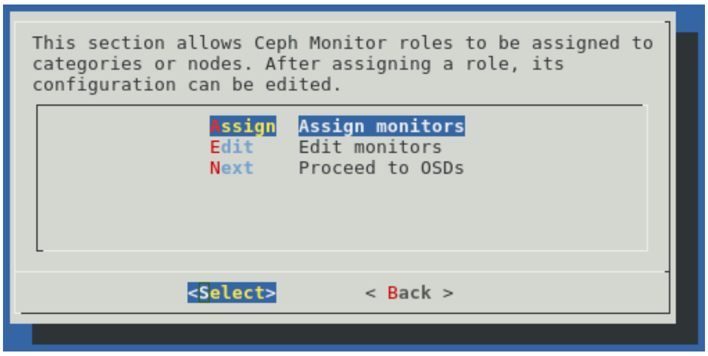

Figure 4.4: Ceph Installation Monitors Configuration

In this screen:

- Ceph Monitor roles can be assigned to, or removed from, nodes or categories. On assigning a role, the nodes running the monitors can have their properties edited with the edit menu option.
- Existing Ceph Monitors can have their properties edited (figure 4.5), after selecting the nodes or categories.
- The OSD configuration screen can be reached after making changes, if any, to the Ceph Monitor configuration.

Typically in a first run, the head node has a Ceph Monitor added to it.

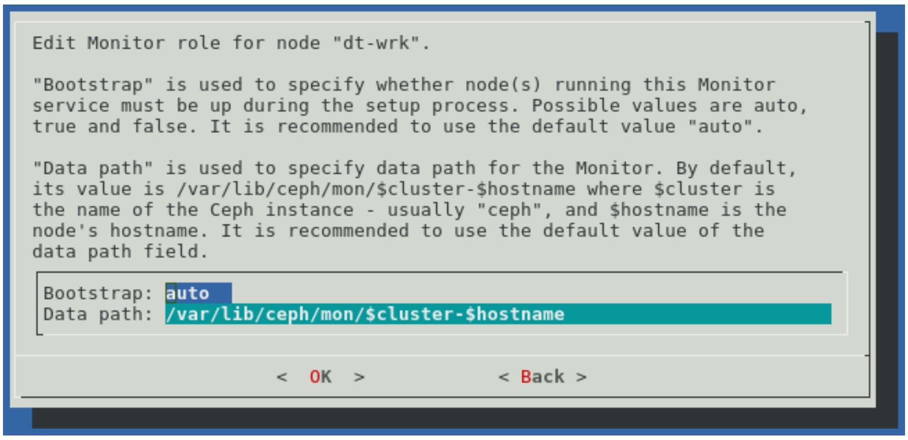

Figure 4.5: Ceph Installation Monitors Editing: Bootstrap And Data Path

Editing Ceph Monitors: The `Edit` option in figure 4.4 opens up a screen, figure 4.5, that allows the editing of existing or newly-added Ceph Monitors for a node or category:

- The `bootstrap` option can be set. The option configures initialization of the maps on the Ceph Monitors services, prior to the actual setup process. The `bootstrap` option can take the following values:
  - `auto`: This is the default and recommended option. If the majority of nodes are tagged with `auto` during the current configuration stage, and configured to run Ceph Monitors, then
    * If they are up according to Bright Cluster Manager at the time of deployment of the setup process, then the Monitor Map is initialized for those Ceph Monitors on those nodes.

  * If they are down at the time of deployment of the setup process, then the maps are not initialized.
- true: If nodes are tagged true and configured to run Ceph Monitors, then they will be initialized at the time of deployment of the setup process, even if they are detected as being down during the current configuration stage.
- false: If nodes are tagged false and configured to run Ceph Monitors, then they will not be initialized at the time of deployment of the setup process, even if they are detected as being up during the current configuration stage.

- The data path is set by default to:

  /var/lib/ceph/mon/$cluster-$hostname

  where:

  - $cluster is the name of the Ceph instance. This is ceph by default.
  - $hostname is the name of the node being mapped.

- The Back option can be used after accessing the editing screen, to return to the Ceph Monitors configuration screen (figure 4.4).

Ceph OSDs Configuration

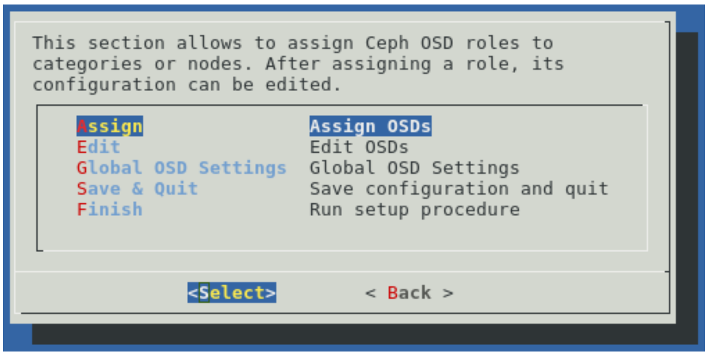

Figure 4.6: Ceph OSDs Configuration

If Proceed to OSDs is chosen from the Ceph Monitors configuration screen in figure 4.4, then a screen for Ceph OSDs configuration (figure 4.6) is displayed, where:

- OSDs roles can be assigned or removed from nodes or categories.
- Existing OSDs can be edited (figure 4.9) from nodes or categories.
- Global OSD settings can be edited (figure 4.10).
- The configuration can be saved for later with the Save & Quit option.
- To finish up on the installation, the Finish option runs the Ceph setup procedure itself.

Editing Ceph OSDs: The Edit option in figure 4.6 allows the properties of existing or newly-added Ceph OSDs for a node or category to be edited. To do this, the storage backend must first be chosen (figure 4.7):

Figure 4.7: Ceph Installation OSDs Editing: Select Storage Backend

The recommended storage backend is BlueStore (section 4.1.2). If `BlueStore` is chosen, then a screen is displayed that allows further BlueStore configuration (figure 4.8).

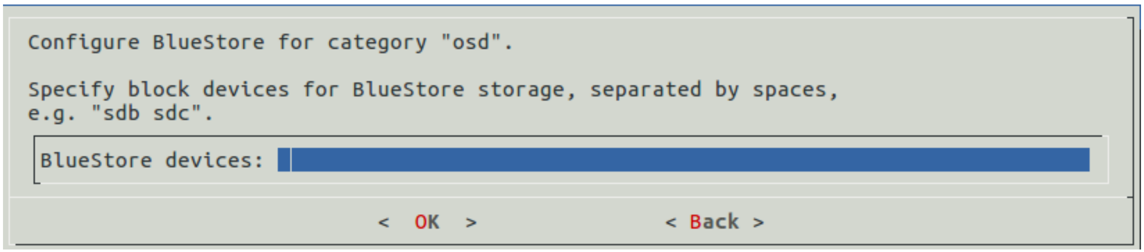

Figure 4.8: Ceph Installation OSDs Editing: BlueStore Configuration

The list of block devices to be used for Ceph storage is entered into the corresponding field. During installation these block devices are cleared and prepared for Ceph storage.

When entering the device names, the `/dev/` directory path prefix can be omitted.

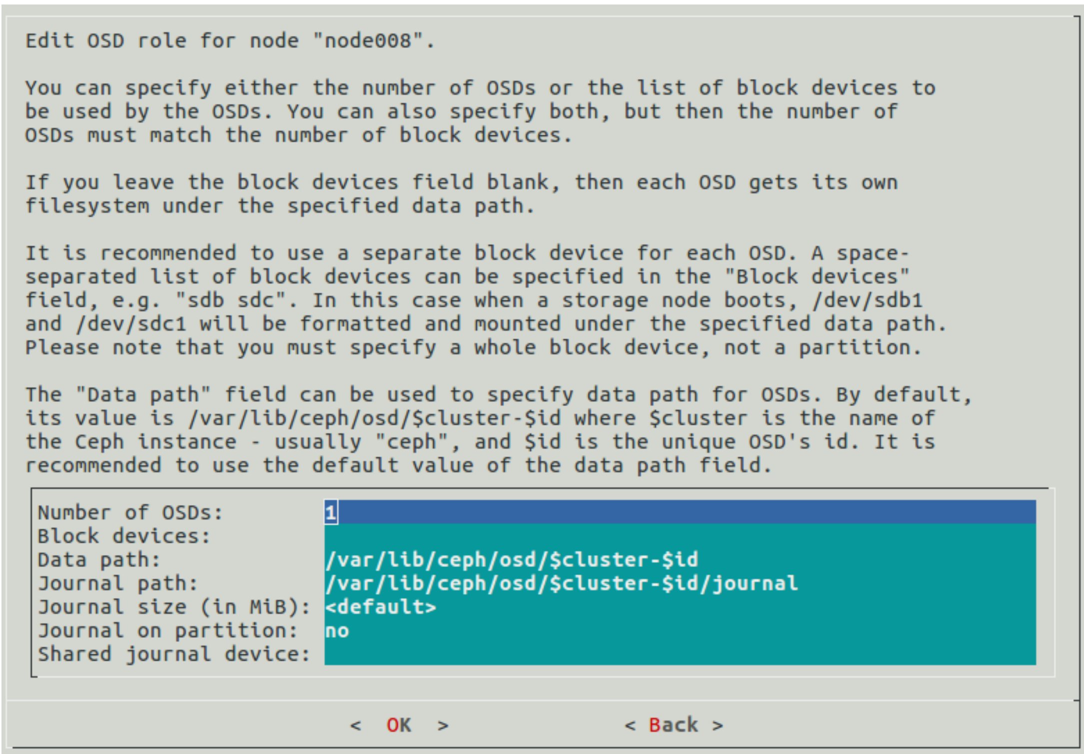

Figure 4.9: Ceph Installation OSDs Editing: Legacy/FileStore Properties For Categories Or Nodes

When `Legacy` is chosen, a screen with FileStore properties is shown (figure 4.9). For this screen:

- When considering the `Number of OSDs` and the `Block devices`, then it is best to set either

  - the `Number of OSDs`

  or

  - the `Block devices`

  Setting *both* the number of OSDs and block devices is also possible, but then the number of OSDs must match the number of block devices.

- If only a number of OSDs is set, and the block devices field is left blank, then each OSD is given its own filesystem under the data-path specified.

- `Block devices` can be set as a comma- or a space-separated list, with no difference in meaning.

  Example

  `/dev/sda,/dev/sdb,/dev/sdc`
  and
  `/dev/sda /dev/sdb /dev/sdc`
  are equivalent.

- For the OSD `Data path`, the recommended, and default value is:

  `/var/lib/ceph/osd/$cluster-$id`

  Here:

  - `$cluster` is the name of the head node of the cluster.
  - `$id` is a number starting from `0`.

- For the `Journal path`, the recommended, and default value is:

  `/var/lib/ceph/osd/$cluster-$id/journal`

- The `Journal size`, in MiB, can be set for the category or node. A value set here overrides the default global journal size setting (figure 4.3). This is just the usual convention where a node setting can override a category setting, and a node or category setting can both override a global setting.

  Also, just like in the case of the global journal size setting, a journal size for categories or nodes must always be greater than zero. Defining a value of 0 MiB means that the default that the Ceph software itself provides is set. At the time of writing (July 2016), Ceph software provides a default of 5GiB.

  The `Journal size` for a category or node is unset by default, which means that the value set for `Journal size` in this screen is determined by whatever the global journal size setting is, by default.

- Setting `Journal on partition` to `yes` means that the OSD uses a dedicated partition. In this case:

  - The disk setup used is modified so that the first partition, with a size of `Journal size` is used
  - A value of `0` for the `Journal size` is invalid, and does not cause a Ceph default size to be used.

The default value of `Journal on partition` is `no`.

- The `Shared journal device` path must be set if a shared device is used for all the OSD journals in the category or node for which this screen applies. If it is used, then the device is partitioned automatically, and the available space is divided equally among the OSD journals.

  The path is unset by default, which means the device is not used by default.

The `Back` option can be used after accessing the editing screen, to return to the Ceph OSDs configuration screen (figure 4.6).

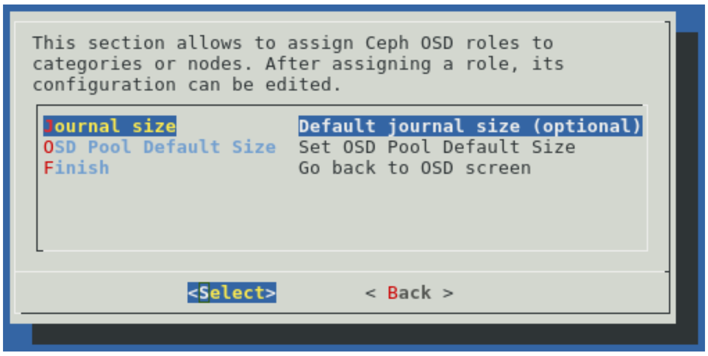

Figure 4.10: Ceph Installation OSD Global Settings Editing: Default Journal Size, Default OSD Pool Size

Editing Ceph `Global OSD Settings`: The `Global OSD Settings` option can be selected from the `Ceph OSDs` main configuration screen (figure 4.6). The screen then displayed (figure 4.10) allows the following options to be modified:

- `Journal size`: The default OSD journal size, in MiBs, used by an OSD. The actual size must be greater than zero. The value can be overridden by a category or node setting later on.

  Defining a value of 0 MiB here means that the default that the Ceph software itself provides is set. At the time of writing (July 2016), Ceph software provides a default of 5GiB.

  For BlueStore devices the OSD journal size is irrelevant.

- `OSD Pool Default Size`: The default OSD pool size. This sets the number of replicas for objects in the pool. It should be less than or equal to the number of OSD nodes. If unsure the administrator can just leave it at the default value.

The `Back` or `Finish` options can then be used to return to the Ceph OSDs configuration screen (figure 4.6).

Save Configuration And Quit

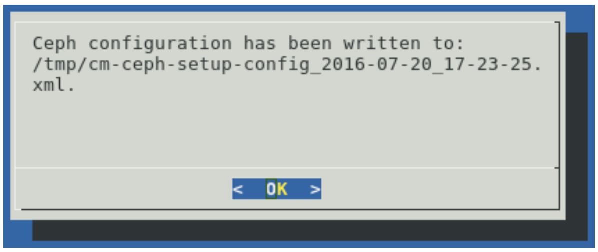

Figure 4.11: Ceph Installation Configuration Was Saved

After selecting the `Save & Quit` option of figure 4.6, the Ceph setup configuration file is saved, (figure 4.11), and the configuration part of the `cm-ceph-setup` script is completed. The deployment stage of the installation is next.

4.2.2 Ceph Installation: The Deployment Stage

After selecting the `Finish` option of figure 4.6, the Ceph setup proceeds. First, the results of sanity checks are displayed:

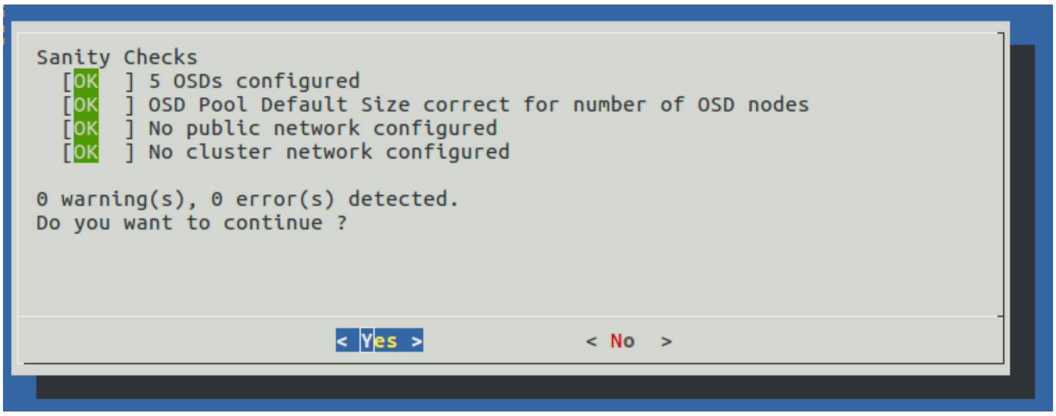

Figure 4.12: Ceph Installation Sanity Checks Results

Then installation confirmation is displayed:

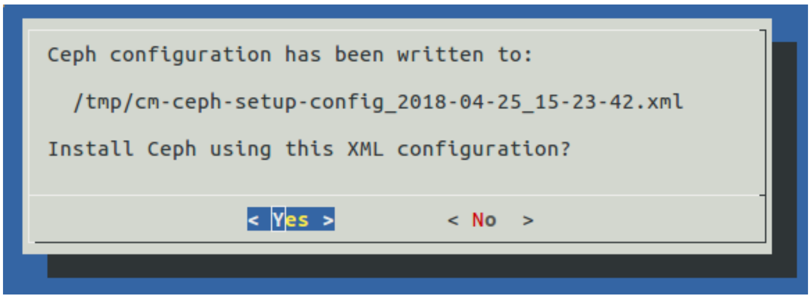

Figure 4.13: Ceph Installation Confirmation

After that, the installation process starts. It first asks to confirm that the Ceph nodes may be rebooted:

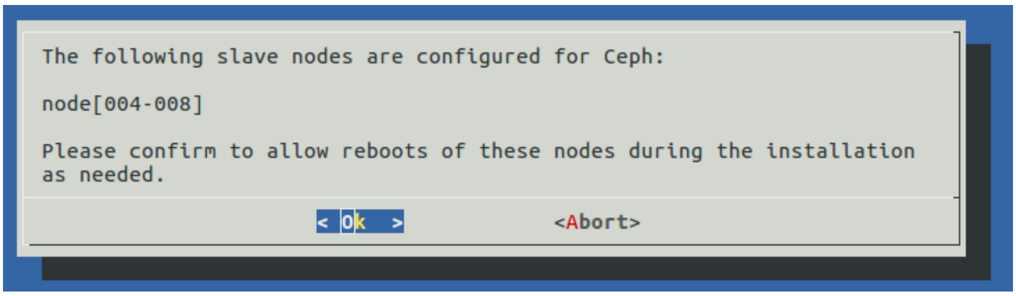

Figure 4.14: Nodes Reboot Confirmation

A successful run displays a screen as in figure 4.15:

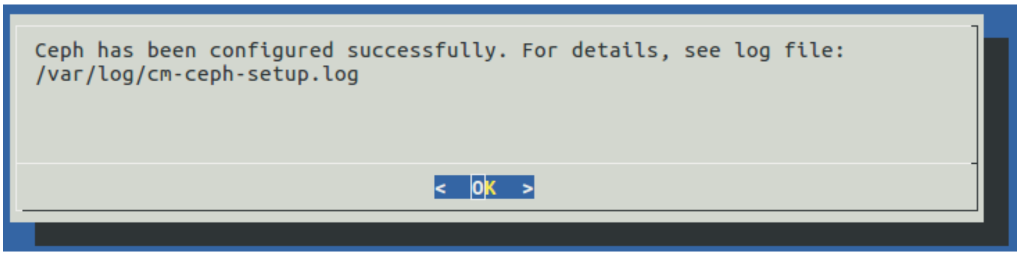

Figure 4.15: Ceph Installation Completion

4.3 Checking And Getting Familiar With Ceph Items After `cm-ceph-setup`

4.3.1 Checking On Ceph And Ceph-related Files From The Shell

After deployment, the OSD and Monitor services take some time be created and to start up. When all is up and running, the status of a healthy system, according to the output of the `ceph -s` command, should look something like the following:

Example

```
[root@bright81 ~]# ceph -s
    cluster 163589aa-c50e-46f3-8514-6f053cae5f2a
     health HEALTH_OK
     monmap e1: 1 mons at bright81=10.141.255.254:6789/0
            election epoch 3, quorum 0 bright81
     osdmap e7: 3 osds: 3 up, 3 in
            flags sortbitwise,require_jewel_osds
      pgmap v1005: 64 pgs, 1 pools, 0 bytes data, 0 objects
            25291 MB used, 5392 MB / 30684 MB avail
                  64 active+clean
```

A nearly full Ceph system, which is still functioning properly, would show something like:

Example

```
[root@bright81 ~]# ceph -s
    cluster 3404ee8a-1c3d-44b0-bbb8-3d2c5ae539f2
     health HEALTH_WARN
            3 near full osd(s)
     monmap e1: 1 mons at bright81=10.141.255.254:6789/0
            election epoch 3, quorum 0 bright81
     osdmap e9: 3 osds: 3 up, 3 in
            flags nearfull,sortbitwise,require_jewel_osds
      pgmap v38: 64 pgs, 1 pools, 0 bytes data, 0 objects
            26407 MB used, 4276 MB / 30684 MB avail
                  64 active+clean
```

The `-h` option to `ceph` lists many options. Users of Bright Cluster Manager should usually not need to use these, and should find it more convenient to use the Bright View or `cmsh` front ends instead.

Generated XML Configuration File

An XML configuration file is generated after a run by the `cm-ceph-setup` utility. The configuration file is stored under the `/tmp` directory. The name of the file is composed of `cm-ceph-setup-config_` followed by a timestamp:

Example

```
/tmp/cm-ceph-setup-config_2017-09-28_13-54-37.xml
```

The name of the Ceph instance is `ceph` by default. If a new instance is to be configured with the `cm-ceph-setup` utility, then a new name must be set in the configuration file, and the new configuration file must be used.

Using An XML Configuration File

The `-c` option to `cm-ceph-setup` allows an existing XML configuration file to be used.

Example

```
[root@bright81 ~]# cm-ceph-setup -c /root/myconfig.xml
```

A Sample XML Configuration File

The following Ceph XML configuration schema is an example that has MONs and OSDs running on different hosts, and has a category `osd` set up for BlueStore. It also has a node `node001` set up for a legacy FileStore configuration, with two OSD block devices (`osd0` and `osd1`), and a shared journal device (`/dev/sdb`):

Example

```
<cephConfig>
  <journalsize>0</journalsize>
  <osdpooldefaultsize>3</osdpooldefaultsize>
  <monitors>
    <monitor>
      <hostname>bright</hostname>
      <monitordata>/var/lib/ceph/mon/$cluster-$hostname</monitordata>
    </monitor>
  </monitors>
  <osds>
    <osd>
      <category>osd</category>
      <osdconfigs>
        <osdbluestoreconfig>
          <name>osd0</name>
          <osddevice>/dev/sdb</osddevice>
        </osdbluestoreconfig>
        <osdbluestoreconfig>
          <name>osd1</name>
          <osddevice>/dev/sdc</osddevice>
        </osdbluestoreconfig>
      </osdconfigs>
    </osd>
    <osd>
      <hostname>node001</hostname>
      <sharedjournal>
        <blockdev>/dev/sdb</blockdev>
      </sharedjournal>
      <osdconfigs>
        <osdlegacyconfig>
          <name>osd0</name>
          <blockdev>/dev/sdc</blockdev>
          <osddata>/var/lib/ceph/osd/$cluster-$id</osddata>
          <journaldata>/var/lib/ceph/osd/$cluster-$id/journal</journaldata>
        </osdlegacyconfig>
```

```
        <osdlegacyconfig>
          <name>osd1</name>
          <blockdev>/dev/sdd</blockdev>
          <osddata>/var/lib/ceph/osd/$cluster-$id</osddata>
          <journaldata>/var/lib/ceph/osd/$cluster-$id/journal</journaldata>
        </osdlegacyconfig>
      </osdconfigs>
    </osd>
  </osds>
</cephConfig>
```

For legacy FileStore configurations, the partitioning of Ceph OSD storage devices is done using the disk setup functionality as described in section 3.9.3 of the *Administrator Manual*. For BlueStore, the corresponding devices are listed in the CephOSDBlueStoreConfig of the CephOSDRole only, and no entries are added to the XML disk layout. The following example corresponds to the Legacy configuration of node001 above.

Example

```
<diskSetup>
  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a2">
      <size>max</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
  <device origin="cm-ceph-setup">
    <blockdev>/dev/sdb</blockdev>
    <partitionTable>gpt</partitionTable>
    <partition id="/dev/vdc1">
      <size>1/2</size>
      <type>linux</type>
    </partition>
    <partition id="/dev/vdc2">
      <size>max</size>
      <type>linux</type>
    </partition>
  </device>
  <device origin="cm-ceph-setup;copied-from-category">
    <blockdev>/dev/sdc</blockdev>
    <partition id="osd0">
      <cephosdassociation>osd0</cephosdassociation>
      <size>max</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mkfsFlags>-i size=2048</mkfsFlags>
      <mountOptions>defaults,noatime,nodiratime,inode64</mountOptions>
    </partition>
  </device>
  <device origin="cm-ceph-setup">
    <blockdev>/dev/sdd</blockdev>
    <partition id="osd1">
```

```
      <cephosdassociation>osd1</cephosdassociation>
      <size>max</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mkfsFlags>-i size=2048</mkfsFlags>
      <mountOptions>defaults,noatime,nodiratime,inode64</mountOptions>
    </partition>
  </device>
</diskSetup>
```

Installation Logs

Installation logs to Ceph are kept at:

```
/var/log/cm-ceph-setup.log
```

4.3.2 Ceph Management With Bright View And cmsh

Only one instance of Ceph is supported at a time. Its name is `ceph`.

Ceph Overview And General Properties

From within `cmsh`, `ceph` mode can be accessed:

Example

```
[root@bright81 ~]# cmsh
[bright81]% ceph
[bright81->ceph]%
```

From within `ceph` mode, the `overview` command lists an overview of Ceph OSDs, MONs, and placement groups for the `ceph` instance. Parts of the displayed output are elided in the example that follows for viewing convenience:

Example

```
[bright81->ceph]% overview ceph
Parameter                         Value
--------------------------------- ----------------------------
Status                            HEALTH_OK
Number of OSDs                    2
Number of OSDs up                 2
Number of OSDs in                 2
Number of mons                    1
Number of placements groups       192
Placement groups data size        0B
Placement groups used size        10.07GB
Placement groups available size   9.91GB
Placement groups total size       19.98GB

Name                              Used       Objects      ...
--------------------------------- ---------- ----------   ...
bright81:.rgw                     1B         0            ...
bright81:data                     0B         0            ...
bright81:metadata                 0B         0            ...
bright81:rbd                      0B         0            ...
...
```

The Bright View equivalent of the `overview` command is the Ceph `Overview` window, accessed via the clickpath `Storage→Ceph→Ceph Settings→Overview`.

Some of the major Ceph configuration parameters can be viewed and their values managed by CMDaemon from `ceph` mode. The `show` command shows parameters and their values for the `ceph` instance:

Example

```
[bright81->ceph]% show ceph
Parameter                          Value
------------------------------     ----------------------------------------
Admin keyring path                 /etc/ceph/ceph.client.admin.keyring
Auto Adjust CRUSH Map              no
Bootstrapped                       yes
Client admin key                   AQDkUM5T4LhZFxAA/JQHvz8agavb9xH0bwvxUSQ==
Cluster networks
Config file path                   /etc/ceph/ceph.conf
Creation time                      Mon, 02 Jul 2018 16:26:02 CEST
Extra config parameters
Monitor daemon port                6789
Monitor key                        AQDkUM5TwM2lEBBg0CcdH/UFhGJ902n3y/Avng==
Monitor keyring path               /etc/ceph/ceph.mon.keyring
Public networks
Revision
auth client required cephx         yes
auth cluster required cephx        yes
auth service required cephx        yes
filestore xattr use omap           no
fsid                               abf8e6af-71c0-4d75-badc-3b81bc2b74d8
mon max osd                        10000
mon osd full ratio                 0.95
mon osd nearfull ratio             0.85
name                               ceph
osd pool default min size          0
osd pool default pg num            8
osd pool default pgp num           8
osd pool default size              2
rbd cache                          yes
rbd cache max dirty                25165824
rbd cache max dirty age            1.000000
rbd cache size                     33554432
rbd cache target dirty             16777216
rbd cache writethrough until flush yes
rbd readahead disable after bytes  52428800
rbd readahead max bytes            524288
rbd readahead trigger requests     10
version                            12.2.5
[bright81->ceph]%
```

The Bright View equivalent of these settings is in the `Settings` window, accessed via a clickpath of `Storage`→`Ceph`→`Ceph Settings`→`Overview`→`Settings`.

Ceph `extraconfigparameters` setting: The `Extra config parameters` property of a `ceph` mode object can be used to customize the Ceph configuration file. The Ceph configuration file is typically in `/etc/ceph.conf`, and using `extraconfiparameters` settings, Ceph can be configured with changes that CMDaemon would otherwise not manage. After the changes have been set, CMDaemon manages them further.

Thus, the following configuration section in the Ceph configuration file:

```
[mds.2]
host=rabbit
```

could be placed in the file via `cmsh` with:

Example

```
[root@bright81 ~]# cmsh
[bright81]% ceph
[bright81->ceph[ceph]]% append extraconfigparameters "[mds.2] host=rabbit"
[bright81->ceph*[ceph*]]% commit
```

If a section name, enclosed in square brackets, `[]`, is used, then the section is recognized at the start of an appended line by CMDaemon.

If a section that is specified in the square brackets does not already exist in `/etc/ceph.conf`, then it will be created. The `\n` is interpreted as a new line at its position. After the commit, the extra configuration parameter setting is maintained by the cluster manager.

If the section already exists in `/etc/ceph.conf`, then the associated key=value pair is appended. For example, the following appends `host2=bunny` to an existing `mds.2` section:

```
[bright81->ceph[ceph]]% append extraconfigparameters "[mds.2] host2=bunny"
[bright81->ceph*[ceph*]]% commit
```

If no section name is used, then the key=value entry is appended to the [global] section.

```
[bright81->ceph[ceph]]% append extraconfigparameters "osd journal size = 128"
[bright81->ceph*[ceph*]]% commit
```

The `/etc/ceph.conf` file has the changes written into it about a minute after the commit, and may then look like (some lines removed for clarity):

```
[global]
auth client required = cephx
osd journal size=128

[mds.2]
host=rabbit
host2=bunny
```

As usual in `cmsh` operations (section 2.5.3 of the *Administrator Manual*):

- The `set` command clears `extraconfigparameters` before setting its value
- The `removefrom` command operates as the opposite of the `append` command, by removing key=value pairs from the specified section.

There are similar `extraconfigparameters` for Ceph OSD filesystem associations (page 68) and for Ceph monitoring (page 69).

Ceph OSD Properties

From within `ceph` mode, the `osdinfo` command for the Ceph instance displays the nodes that are providing OSDs along with their OSD IDs:

Example

```
[bright81->ceph]% osdinfo ceph
OSD id       Node                    OSD name
------------ ----------------------- ------------
0            node001                 osd0
1            node002                 osd0
```

Within a device or category mode, the `roles` submode allows parameters of an assigned `cephosd` role to be configured and managed.

Example

```
[bright81->device[node001]->roles]% show cephosd
Parameter                    Value
---------------------------- ------------------------------
Add services                 yes
Name                         cephosd
OSD configurations           <1 in submode>
Provisioning associations    <0 internally used>
Revision
Type                         CephOSDRole
```

Within the `cephosd` role the templates for OSD filesystem configurations, `osdconfigurations`, can be set or modified:

Example

```
[bright81->device[node001]->roles]% use cephosd
[bright81...[node001]->roles[cephosd]]% osdconfigurations
[bright81...osd]->osdconfigurations]% show osd0
Parameter                         Value
--------------------------------- ---------------------------------------
Automatically adjust weight       off
Extra config parameters
Initial weight                    0.1
Journal data                      /var/lib/ceph/osd/$cluster-$id/journal
Journal size                      0 MiB
Name                              osd0
OSD data                          /var/lib/ceph/osd/$cluster-$id
Production weight                 1
Revision
Type                              CephOSDLegacyConfig
Weight adjust interval            5
Weight adjust rate                0.1
Weight interpretation             scale
```

The Bright View equivalent to access the preceding `cmsh` OSD configuration settings is via the role for a particular node or category. The clickpath that brings up these configuration options for node `node001` is, for example:

`Devices→Physical Nodes→node001→Edit→Settings→Roles→cephosd→Edit→osd0→Edit`

OSD filesystem association `extraconfigparameters` setting: Extra configuration parameters can be set for an OSD filesystem association such as `ods0` by setting values for its `extraconfigparameters` option. This is similar to how it can be done for Ceph general configuration (page 66):

Example

```
[bright81...osd]->osdconfigurations]% use osd0
[bright81...osdconfigurations[ods0]]% show
Parameter                         Value
--------------------------------- ---------------------------------------
```

```
...
Automatically adjust weight      off
Extra config parameters
...
[bright81...osdconfigurations[osd0]]% set extraconfigparameters "a=b"
...
```

Ceph Monitoring Properties

Similarly to Ceph OSD properties, the parameters of the `cephmonitor` role can be configured and managed from within the node or category that runs Ceph monitoring.

Example

```
[bright81]% device use bright81
[bright81->device[bright81]]% roles ; use cephmonitor
[ceph->device[bright81]->roles[cephmonitor]]% show
Parameter                        Value
-------------------------------- ------------------------------------
...
Extra config parameters
Monitor data                     /var/lib/ceph/mon/$cluster-$hostname
Name                             cephmonitor
Provisioning associations        <0 internally used>
Revision
Type                             CephMonitorRole
```

Ceph monitoring `extraconfigparameters` setting: Ceph monitoring can also have extra configurations set via the `extraconfigparameters` option, in a similar way to how it is done for Ceph general configuration (page 66).

The Bright View equivalent to access the preceding `cmsh` Monitor configuration setting is via the role for a particular node or category. The clickpath that brings up these configuration options for node `node004` is, for example:

`Devices→Physical Nodes→node004→Edit→Settings→Roles→cephmonitor→Edit`

Ceph `bootstrap`

For completeness, the `bootstrap` command within `ceph` mode can be used by the administrator to initialize Ceph Monitors on specified nodes if they are not already initialized. Administrators are however not expected to use it, because they are expected to use the `cm-ceph-setup` installer utility when installing Ceph in the first place. The installer utility carries out the bootstrap initialization as part of its tasks. The `bootstrap` command is therefore only intended for use in the unusual case where the administrator would like to set up Ceph storage without using the `cm-ceph-setup` utility.

4.4 RADOS GW Installation, Initialization, And Properties

4.4.1 RADOS GW Installation And Initialization

If Ceph has been installed using `cm-ceph-setup`, then RADOS is installed and initialized so that it provides a REST API, called the RADOS GW service.

4.4.2 Setting RADOS GW Properties

RADOS GW Properties In `cmsh`

RADOS GW properties can be managed in `cmsh` by selecting the device, then assigning the `radosgateway` role to the device. The properties of the role can then be seen and altered:

```
[bright81]% device use node004
[bright81->device[node004]]% roles
[bright81->device[node004]->roles]% assign radosgateway; commit
[bright81->device[node004]->roles[radosgateway]]% show
Parameter                        Value
-------------------------------- --------------------------------
Name                             radosgateway
Provisioning associations        <0 internally used>
Revision
Type                             RadosGatewayRole
Server Port                      7480
Enable Keystone Authentication   yes
Keystone Accepted Roles
Keystone Revocation Interval     600
Keystone Tokens Cache Size       500
NSS DB Path                      /var/lib/ceph/nss
```

The role properties can also be accessed in via category roles and configuration overlay roles.

Keystone authentication can be disabled or enabled using cmsh to set the enablekeystoneauthentication property.

For example, setting enablekeystoneauthentication to no on a RADOS GW node, and committing it makes RADOS GW services unavailable to that node.

Example

```
[bright81->device[node004]->roles[radosgateway]]% set enablekeystoneauthentication no
[bright81->device*[node004*]->roles*[radosgateway*]]% commit
```

RADOS GW Properties In Bright View

RADOS GW properties can be accessed in Bright View via:

- via the node clickpath:
  Devices→Nodes→Settings→Roles→Rados Gateway Role
- via the node category clickpath:
  Grouping→Node Categories→Settings→Roles→Add→Rados Gateway Role
- or via the configuration overlay clickpath:
  Configuration Overlays→Edit→Roles→radosgateway

4.5 Installation Of Ceph From Bright View

Ceph can be installed from Bright Cluster Manager in the following two ways:

- Using the text-based cm-ceph-setup utility (section 4.2). The utility is a part of the standard cluster-tools package.
- Using the Bright View Ceph Wizard (this section). This is the recommended installation method.

4.5.1 Bright View Ceph Install: Main Details Screen

The clickpath Storage→Ceph→Ceph Wizard brings the browser to the Ceph main details screen, (figure 4.16), if Ceph has not yet been installed by Bright Cluster Manager. This screen is beginning of the Ceph installation process, and the page displayed asks for details of the main Ceph configuration settings:

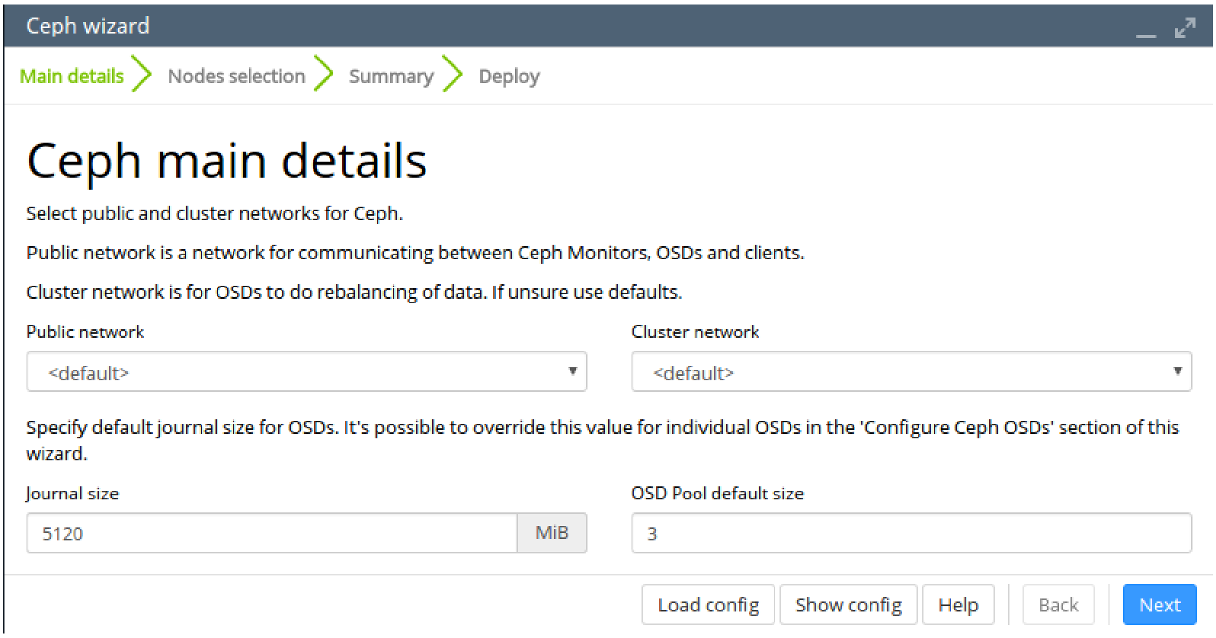

Figure 4.16: Ceph Wizard Installation: General Cluster Settings

The GUI screen of figure 4.16 is a combination of the Ncurses Ceph Installation General Cluster Settings screen figure 4.3 (page 55), together with the Ncurses OSD journal settings of figure 4.10, (page 60). The settings of the Bright View screen are explained in the texts in the section for figures 4.3 and 4.10.

4.5.2 Bright View Ceph Install: Nodes Selection Screen

The next screen is the Ceph Nodes selection screen (figure 4.17). This allows items to be selected for use as Ceph Monitors and OSDs. The items to be selected can be categories or nodes:

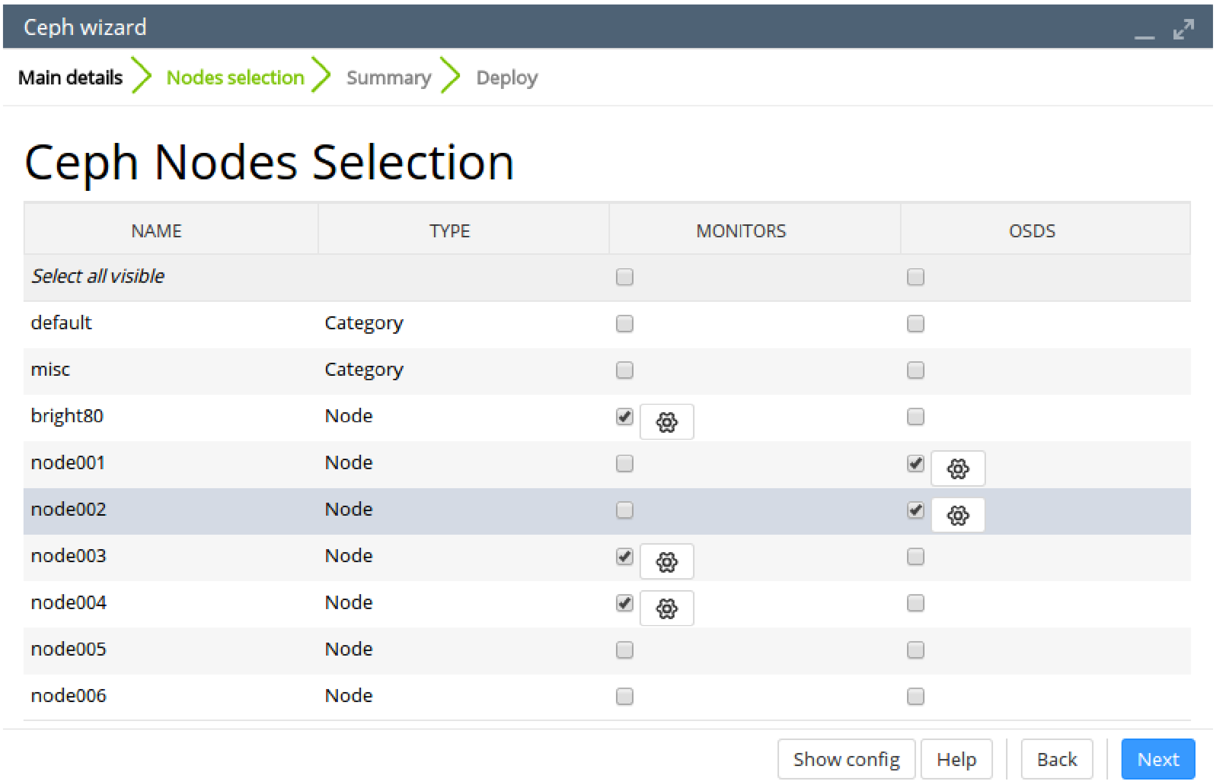

Figure 4.17: Ceph Wizard Installation: Ceph Nodes Selection Screen

4.5.3 Bright View Ceph Install: Summary Screen

The next screen is the Summary screen (figure 4.18). This summarizes the choices that have been made. The Show config button displays the underlying raw XML configuration in a popup window (figure 4.19).

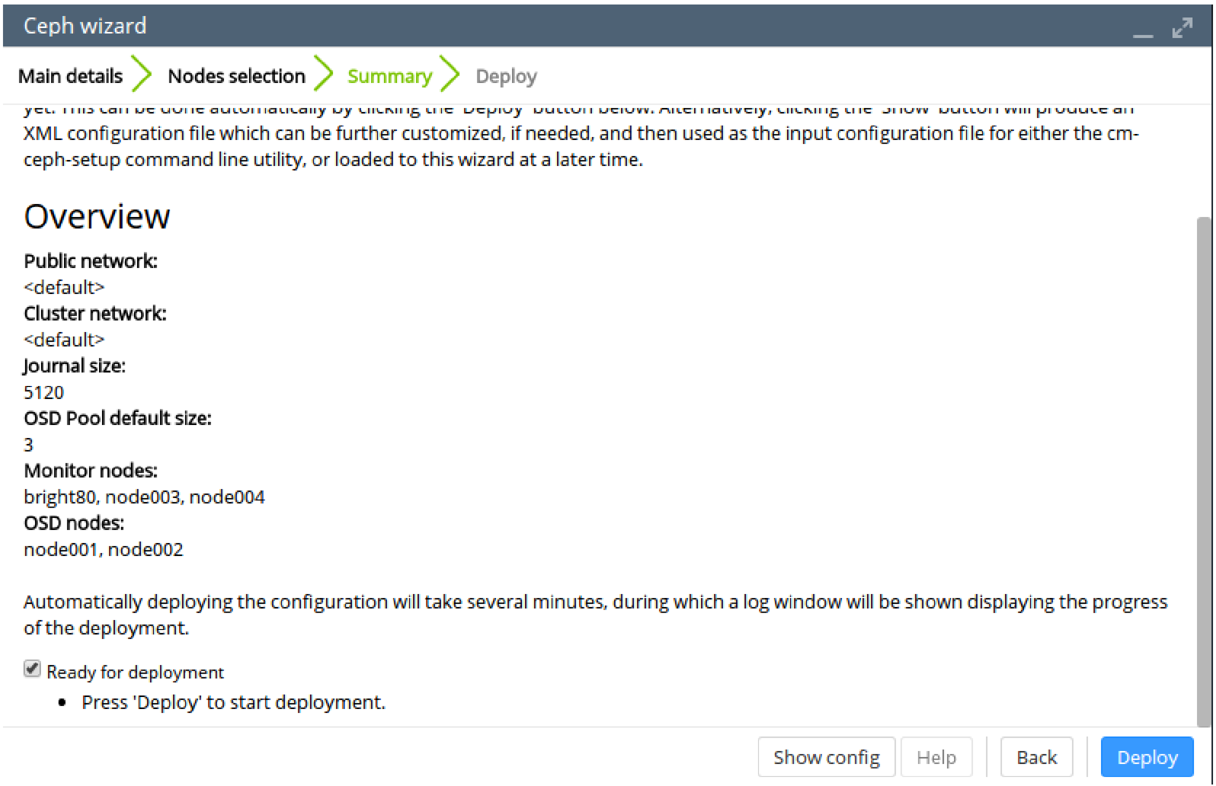

Figure 4.18: Ceph Wizard Installation: Configuration Summary

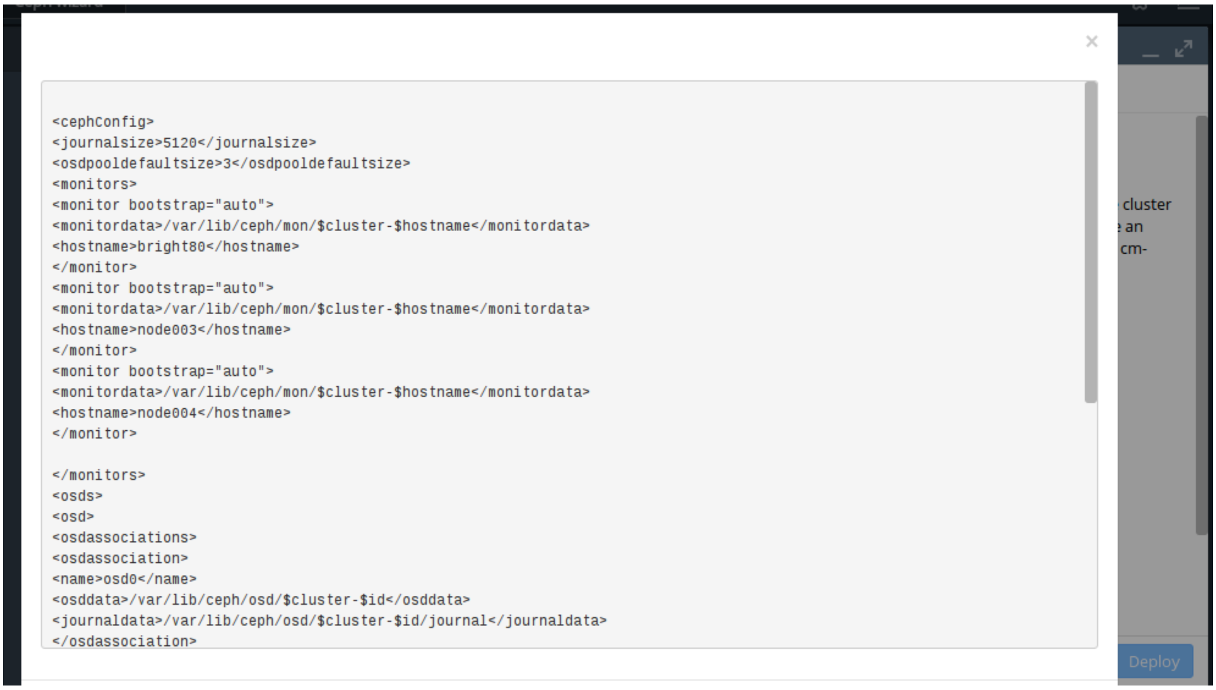

Figure 4.19: Ceph Wizard Installation: Show XML Configuration

In figure 4.18 after the `Ready for deployment` checkbox is checked, the `Deploy` button proceeds with deploying Ceph according to the configuration specified in the wizard.

4.5.4 Bright View Ceph Install: Deployment Screen

During the deployment process, the progress is displayed (figure 4.20).

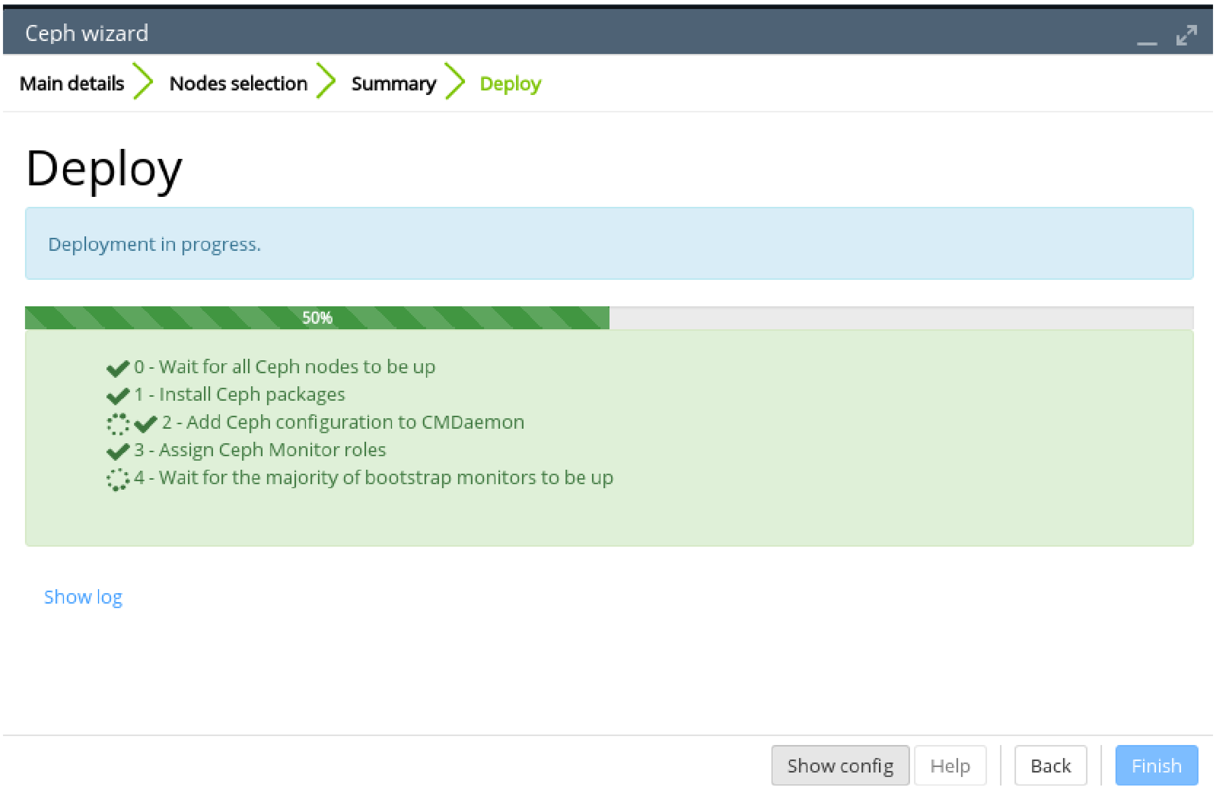

Figure 4.20: Ceph Wizard Installation: Deployment Progress

The event viewer in Bright View also shows the changes taking place. When deployment is complete, the `Finish` button ends the wizard.

The state of the deployed system can be checked as shown in section 4.3.1.

5

User Management And Getting OpenStack Instances Up

In this chapter:

- Section 5.1 describes Bright Cluster Manager's user management integration with OpenStack.
- Section 5.2 describes how a user instance can be run with OpenStack under Bright Cluster Manager. A user instance is an instance that is not a tightly-integrated Bright-managed instance. A Bright-managed instance is a special case of an user instance. Bright-managed nodes are treated by Bright Cluster Manager very much like a regular nodes.
- Section 5.3 describes how a Bright-managed instance is managed in Bright Cluster Manager

5.1 Bright Cluster Manager Integration Of User Management In OpenStack

User management in Bright Cluster Manager without OpenStack is covered in Chapter 6 of the *Administrator Manual*. Users managed in this way are called *Bright users*.

OpenStack allows a separate set of users to be created within its projects. By default, these *OpenStack users* are set up to be independent of the Bright users.

OpenStack user accounts are of two kinds:

- regular users: these are end users who get to use an OpenStack user instance or a Bright-managed OpenStack instance. These can be managed by Bright Cluster Manager's LDAP, or can also simply be managed within OpenStack, depending on the Keystone backend driver used.
- service users: these user accounts are used to run the OpenStack service components. They are associated with the `service` project and `admin` role. Thus, the Nova service has a `nova` user, the Cinder service has a `cinder` user, and so on, and these are all assigned an `admin` role. The list of service user names can be listed in the default installation as follows:

```
[bright81->openstack[default]->roleassignments]% list -f name:25 | grep service
admin:service:admin
cinder:service:admin
cmdaemon:service:admin
glance:service:admin
heat:service:admin
keystone:service:admin
neutron:service:admin
nova:service:admin
radosgw:service:admin
```

The OpenStack service users are stored in the Keystone database, managed by the OpenStack MariaDB running on the controller nodes.

In cmsh the role assignment name field is in the form of:

<OpenStack user name>:*<project>*:*<role>*

The background note on page 81 has some further details on role assignment in Bright Cluster Manager OpenStack edition.

The service user radosgw is created only if the RADOS GW is installed (section 4.4).

Regular OpenStack users can be created in several ways, including:

- using cmsh, from within openstack mode
- using Bright View, via the clickpath OpenStack→Identity→Users
- using the OpenStack Horizon dashboard, where clicking on the Identity sidebar resource leads to the Users management window
- using the openstack command line utility that comes with OpenStack.
- using the Keystone Python API, which is an option that is more likely to be of interest to developers rather than system administrators

The details of how this is carried out depends on user database backend configuration. OpenStack users and Bright users can be given the same name and password in several ways, depending on the database driver used by Keystone (section 3.1.4), and how the administrator configures the users using the initialization and migration scripts (section 5.1.2).

Having the OpenStack service users not be in the Bright Cluster Manager LDAP and thus not be the same as Bright users has some advantages.

Having OpenStack regular users be the same as Bright users is also something that administrators may want.

Background Note: The User Database Drivers, User Migration And Initialization

This section on database drivers is offered as background material to provide a deeper understanding of user management in Bright Cluster Manager with OpenStack. It can be skipped by administrators who have no need to understand how the configuration can be customized, or who have been provided with a customized configuration already.

It should be understood that Bright users are not OpenStack users by default when OpenStack setup is carried out. To make a Bright user able to use OpenStack under the same user name, some configuration must be carried out. The exact configuration depends upon the use case. The main configuration involves the type of backend user database driver used, and can additionally include the option of initialization and migration scripts.

Initialization and migration scripts are scripts that can be used to initialize and migrate Bright users to become OpenStack users, after OpenStack setup has been carried out.

In this background note, two kinds of Bright users are defined:

1. **legacy users**: These are Bright users created from before the OpenStack initialization and migration scripts are working.
2. **fresh users**: These are Bright users created after the OpenStack initialization and migration scripts are working.

The following table displays the driver configuration options that allow the Bright Cluster Manager user to use OpenStack.

Keystone Driver	Legacy BrightUser And Fresh Bright User Access To OpenStack?	Use To Create New OpenStack Users In OpenStack?
MySQL	No, but migrating Bright users to become regular OpenStack users often makes sense, and can be done for fresh users automatically by configuring initialization and migration scripts.	Yes, OpenStack does it in a self-contained manner within the `members` project without having to configure OpenStack user initialization or user migration scripts. When the initialization and migration scripts are configured, then creating a fresh Bright user can still create a new OpenStack user with the same name.
MySQL + PAM/NSS (Hybrid)	Yes, via `pam` domain, and using role and project assignments.	Not recommended. Typically set up PAM users instead
Bright LDAP	Yes, via `ldap` domain, and some OpenStack project and OpenStack role assignment	No.

The first Keystone driver, MySQL: has Keystone use only Galera's MySQL database for OpenStack users, that is for both the service OpenStack and the regular OpenStack users. It means that Bright Cluster Manager's regular LDAP user database remains in use as another, independent database for Bright users, and these users cannot be used for OpenStack functionality unless the users are duplicated across from Bright Cluster Manager's regular LDAP into the OpenStack domain. Thus, without that duplication, the regular OpenStack users are created by OpenStack actions and are stored in the Galera MySQL database, in the `default` domain associated with a default OpenStack installation.

Not having unified user databases—having the OpenStack MySQL user database distinct from Bright Cluster Manager's regular LDAP user database—means that using the Keystone MySQL driver is typically used for proof-of-concept deployments, or small deployments, rather than larger scale deployments.

User duplication from the Bright Cluster Manager user names to the OpenStack users can be useful for this driver: If a migration script and an initialization script are configured to run on the Bright Cluster Manager user name in CMDaemon (section 5.1.2), then fresh Bright users, when created, have their names duplicated as OpenStack user names, and these names are stored in Galera as well as in the regular LDAP user database. Legacy Bright users are not migrated or initialized by this configuration. The databases remain independent, which means that passwords for a duplicated user name are not matched. The passwords can of course be matched manually by the end user.

The second driver, MySQL + PAM/NSS (Hybrid): has Keystone using Galera's MySQL and also Bright Cluster Manager's PAM/NSS, and is called a hybrid driver. The driver handles the `admin`, `cmdaemon`, and OpenStack service users via Galera's MySQL in the OpenStack domain called `default`. On the other hand, all other users—Bright PAM/NSS authenticated users, and any other PAM/NSS authenticated users—are authenticated via PAM/NSS through this driver, and access OpenStack via the special OpenStack domain `pam`. The Bright Cluster Manager administrator is therefore normally only concerned with the PAM/NSS part of the driver when it concerns managing users.

A convenience with this driver is that there is only one password per user, so that this driver is typically used for larger deployments. It is also a cleaner deployment, having normal users placed in the pam domain and handling them there. Also, if using Bright Cluster Manager for user management, then the administrator can manage passwords and other properties in the standard Bright Cluster Manager way from the top-level cmsh user mode.

With this driver, Bright users that are authenticated with LDAP, can be authenticated by Keystone via PAM/NSS. The driver assigns the user the OpenStack pam domain. Within the OpenStack pam domain, an assignment must be carried out by the administrator for the OpenStack role and for the OpenStack project. Without these role and project assignments within the pam domain, the users are merely authenticated, but disallowed the use of OpenStack services. Typically, therefore, to manage the PAM users in the pam domain of OpenStack, an administrative user, for example, pamadmin, can be created within the pam domain, and given the OpenStack admin role. Such a pamadmin administrator is a separate user from the admin created by default in the default domain. This pamadmin can then assign an appropriate OpenStack role and OpenStack project to the user in the pam domain.

User duplication from the Bright user names to the OpenStack users, using a migration script and an initialization script, is typically not useful for this driver, since it works against the clean placement described earlier. If a migration script and an initialization script are configured to run on the Bright user name in CMDaemon (section 5.1.2), then fresh Bright users, when created, have their names duplicated as OpenStack user names, and these names are stored in Galera together with the service OpenStack users, as well as in the regular LDAP user database. Legacy Bright users are not migrated or initialized by this configuration. The databases remain independent, which means that passwords for a duplicated user name are not matched. The passwords can of course be matched manually by the end user.

The third driver, Bright LDAP: has Keystone using Bright Cluster Manager's own LDAP database, and does not use the OpenStack user database for regular users. That is, Keystone, when using this driver, handles Bright LDAP users only, ignores any NSS/PAM users, and ignores any regular OpenStack users in Galera. The admin, cmdaemon, and service OpenStack users, on the other hand, are still used by Keystone from Galera in OpenStack.

Creation of a fresh user via OpenStack actions will fail, because the LDAPS access from OpenStack is read-only. There is no account ldapadmin that can be created analogous to pamadmin that has the same abilities that pamadmin had with the second driver. That is, there is no account ldapadmin to assign projects and roles to LDAP users. Current LDAP users can be created via a CMDaemon front-end, such as the top-level user mode of cmsh in Bright, and automatically go to the domain associated with OpenStack called ldap. OpenStack projects and OpenStack roles can be assigned to a user from the OpenStack command line. The convenience of a single password for users, the simple architecture, and having everything is contained within Bright Cluster Manager, means that this driver is typically useful for small or medium organizations that are using Bright Cluster Manager as is, without authenticating it to an external database via PAM/NSS.

An aside on duplication when using this driver: Duplication is mentioned here for completeness. It is available, but typically pointless for this driver. If a migration script and an initialization script are configured to run on the Bright user name in CMDaemon (section 5.1.2), then a fresh LDAP user name is duplicated during creation, as an OpenStack user name, and also stored in Galera, but not used from Galera. The databases remain independent, which means that passwords for a duplicated user name are not matched. The passwords can of course be matched manually by the end user. Legacy users are not migrated or initialized by this configuration.

Normally one of the three driver types is chosen in the user management screen during the wizard installation (section 3.1.4) or Ncurses installation (section 3.2.4).

However, the driver type can be added or removed after OpenStack installation, within cmsh by using the authbackends submode. For example, adding a name to the chosen driver type adds the driver while assigning it a name in CMDaemon:

Example

```
[bright81->openstack[default]->settings->authentication->authbackends]% add<TAB><TAB>
hybrid  ldap    sql
[bright81->...authbackends]]% add sql sql |#choosing sql as name for type sql backend
```

Further configuration to suit needs can be quite involved. It is therefore recommended to select the appropriate driver during a wizard or Ncurses installation to begin with.

5.1.1 Managing OpenStack Users As Bright Cluster Manager Users

Most administrators should find that the most convenient way to set up Bright Cluster Manager and OpenStack users is using `cmsh`. Bright Cluster Manager users can be set up from the main `user` mode, while OpenStack users can be set up from within the `users` submodes, under OpenStack mode, in the `cmsh` hierarchy.

Background Note: Avoiding Confusion About User(s) And (Sub)Modes

The administrator should understand that there is a difference between:

- `OpenStack->users` submode: OpenStack users are managed from this submode
- `OpenStack->settings->users` submode: the settings for OpenStack users are managed from this submode
- Bright Cluster Manager `user` mode: Bright Cluster Manager users are managed from this mode

The following treeview illustrates these user(s) (sub)modes in the `cmsh` hierarchy:

```
[cmsh]
  |-- ...
  |-- openstack
  |    |-- ...
  |    |-- settings
  |    |    |--...
  |    |    `-- users
  |    |-- ...
  |    `-- users
  |-- ...
  `-- user
```

5.1.2 Synchronizing Users With The OpenStack Initialization And Migration Scripts

Setting the initialization and migration scripts: Bright Cluster Manager provides initialization and migration scripts that can be called after creating a Bright user. When applied to a Bright Cluster Manager user, the OpenStack user of the same name is created as follows:

- The migration script, `/cm/local/apps/cluster-tools/bin/cm-user-migration`, copies a Bright Cluster Manager user name from the LDAP records over to the OpenStack Keystone records, and by default sets a random password for the OpenStack user.
- The initialization script, `/cm/local/apps/cluster-tools/bin/cm-user-init`, creates an OpenStack project for the OpenStack user with the same name, if it does not already exist. The user is also assigned the `member` role. Role assignment here means that the OpenStack user is associated with a project and assigned a role for the purposes of the OpenStack utility (page 81, Background Note: Automated Role Assignment In OpenStack).

The `cmsh` parameters `userinitscript` and `migrationscript` can be set to these initialization and migration script paths. The parameters are initially blank by default. They can be set from within the OpenStack settings submode of `cmsh` for users as follows:

Example

```
[root@bright81 ~]# cmsh
[bright81]% openstack
[bright81->openstack[default]]% settings ; users
[...settings->users]% set userinitscript /cm/local/apps/cluster-tools/bin/cm-user-init
[...settings*->users*]% set migrationscript /cm/local/apps/cluster-tools/bin/cm-user-migration
[...settings*->users*]% commit
```

If Bright View is used, then the path parameters can be accessed via the clickpath:

`OpenStack→Settings→Users`

If the default scripts are set as in the preceding example, then they are automatically executed for the user when creating a regular Bright Cluster Manager user.

The administrator can customize the scripts, according to need, for example by copying them, then modifying the copies and assigning the modified copies to the `userinitscript` and `migrationscript` parameters.

Automated OpenStack user creation: With the initialization and migration scripts set, OpenStack user creation now automatically takes place during regular user creation:

Example

```
[...settings->users]% user
[bright81->user]% add fred
[bright81->user*[fred*]]% set password secret123; commit
```

If Keystone uses the MySQL driver, then the password of the Bright Cluster Manager user and the password for the OpenStack user of the same name are independent. By default, the OpenStack user has a password that is random, and which the migration script places in `~/.openstackrc_password`.

To check that user `fred` can login as an OpenStack user, a login can be attempted via `http://`*<load balancer IP address>*`:10080` using the password defined in his `.openstackrc_password` file (figure 5.1):

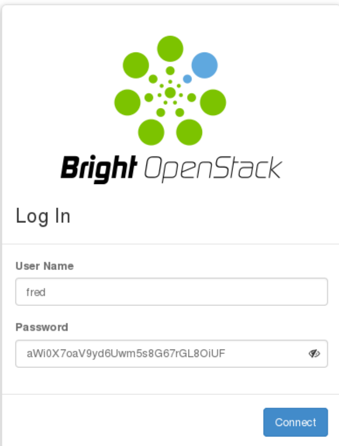

Figure 5.1: Login With Horizon At `http://`*<load balancer IP address>*`:10080`

If all is well, then the login for the end user succeeds and leads to an overview screen for the user (figure 5.2):

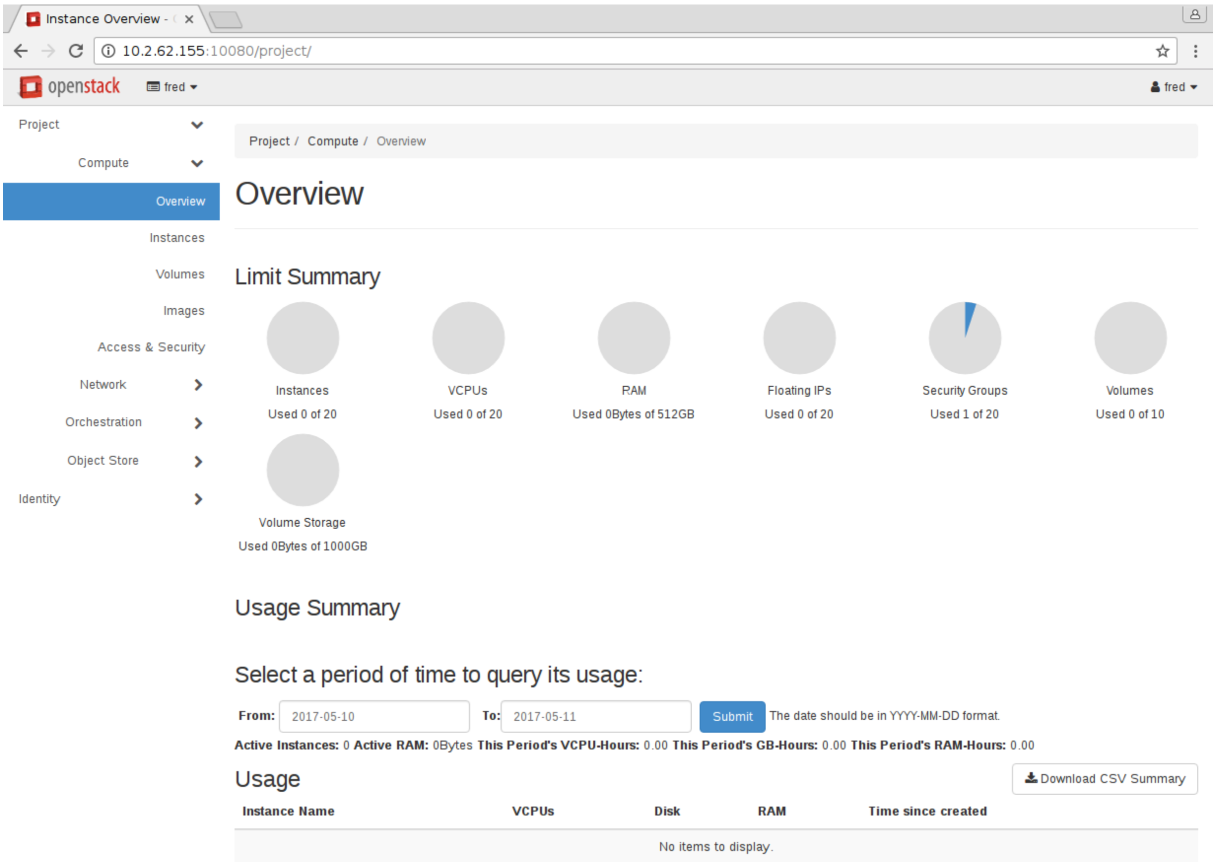

Figure 5.2: Successful Login With Horizon At `http://`*<load balancer IP address>*`:10080`

In an unmodified cluster there should be no instances running yet.

At this point, some background notes to help understand what is going on can be read by simply continuing with reading this chapter sequentially. Alternatively, if an administrator has a sufficiently deep understanding of and familiarity with Bright Cluster Manager and OpenStack, then it is possible to skip ahead to section 5.2, where getting an OpenStack instance up and running is described.

Background Note: Automated Role Assignment In OpenStack

If the default scripts for migration and initialization are in place, then the creation of a Bright user automatically creates an OpenStack user, with a default role assignment in the form of:

<OpenStack user name>:*<project>*:*<role>*

For example, creating the LDAP user `fred` in Bright Cluster Manager, automatically:

- creates an OpenStack user `fred`
- assigns the OpenStack user `fred` the default project `fred`, creating the project if needed
- assigns the OpenStack user `fred` the default role `member`
- assigns the OpenStack user `fred` a key `fred:fred:member` that can be used by the OpenStack utility

Example

```
[bright81->user[fred]]% openstack users
```

```
[bright81->openstack[default]->users]% list -f name
name (key)
--------------------
admin
cinder
cmdaemon
fred
glance
heat
keystone
neutron
nova
[bright81->openstack[default]->users]% projects
[bright81->openstack[default]->projects]% list
Name (key)   UUID (key)                        Domain             Enabled MOTD
------------ -------------------------------- ------------------ ------- ------
bright       83b48ea2016c4658b3b1e01a910011d9  Default (default)  yes
fred         b48cd2f6da4645a8886b494ad5f459c6  Default (default)  yes
service      aa239b1f054a470cbe40f74984a9331d  Default (default)  yes
[bright81->openstack[default]->projects]% roleassignments; list -f name,user,project,role
name (key)           user                 project              role
-------------------- -------------------- -------------------- --------------------
admin:bright:admin   admin (bfd7fd66b1ab+ bright (83b48ea2016+ admin (c7b7e8f8c885+
admin:service:admin  admin (bfd7fd66b1ab+ service (aa239b1f05+ admin (c7b7e8f8c885+
cinder:service:admin cinder (e173c5545c8+ service (aa239b1f05+ admin (c7b7e8f8c885+
cmdaemon:bright:adm+ cmdaemon (fae4250c3+ bright (83b48ea2016+ admin (c7b7e8f8c885+
cmdaemon:service:ad+ cmdaemon (fae4250c3+ service (aa239b1f05+ admin (c7b7e8f8c885+
fred:fred:member     fred (80e16841e3df2+ fred (b48cd2f6da464+ member (6cb5e5359b6+
glance:service:admin glance (2a0d739783d+ service (aa239b1f05+ admin (c7b7e8f8c885+
heat:service:admin   heat (7acdc31888534+ service (aa239b1f05+ admin (c7b7e8f8c885+
keystone:service:ad+ keystone (1048db4a5+ service (aa239b1f05+ admin (c7b7e8f8c885+
neutron:service:adm+ neutron (e1b01d92e9+ service (aa239b1f05+ admin (c7b7e8f8c885+
nova:service:admin   nova (634f35b3ee0e4+ service (aa239b1f05+ admin (c7b7e8f8c885+
[bright81->openstack[default]->roleassignments]%
```

Background Note: Automated Writing Out Of The `.openstackrc*` Files
Bright OpenStack users have a `.openstackrc` file and a `.openstackrc_password` file associated with them. The `.openstackrc` file provides the OpenStack environment, while the `.openstackrc_password` file provides the OpenStack password. This environment can be used by `openstack`, the OpenStack utility that an OpenStack user can run to manage instances.

The `.openstackrc*` files are generated only when adding an OpenStack user by using the `cmsh` or Bright View front ends to CMDaemon. Using the OpenStack client (`/usr/bin/openstack`) directly to add a user does not create the `.openstackrc*` files.

Within the `settings` submode of OpenStack there is a `users` submode. Within that `users` submode the administrator can set the following parameters to configure the `.openstackrc*` files:

- `Write out OpenStack RC for users`: This parameter configures how the `.openstackrc` file is written for an OpenStack user:
  - `matchinghomedirectories`: writes the file only to home directories that match OpenStack user names
  - `allhomedirectories`: writes the file to all home directories. That is, even if no OpenStack user matches that name
  - `off`: does not write out a file

- `Write out .openstackrc_password`: This parameter can take `yes` or `no` as its value. The value decides if the `.openstackrc_password` file is written for an OpenStack user. This feature only operates when the user is created. So if this option is made active after user creation, then no password file is written out.

Example

```
[root@bright81 ~]# cmsh
[bright81]% openstack
[bright81->openstack[default]]% settings; users
[...settings->users]% set writeoutopenstackrcforusers matchinghomedirectories
[...settings->users*]% set writeout.openstackrc_password yes
[...settings->users*]% commit
```

With the preceding configuration for the `.openstackrc*` files, if an OpenStack user `fred` is created as in the example on page 80, then the home directory for `fred` would look something like:

Example

```
[root@bright81 ~]# ls -a /home/fred/
.  ..  .bash_logout  .bash_profile  .bashrc  .mozilla  .openstackrc  .openstackrc_password
```

The `.openstackrc*` file contents are similar to the following output (some output elided):

Example

```
[root@bright81 ~]# cat /home/fred/.openstackrc_password
export OS_PASSWORD="LMlr6oRENZoIp0iqaI4304JGNn632P"

[root@bright81 ~]# cat /home/fred/.openstackrc
# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
# This file has been generated by the CMDaemon and is meant
# to be sourced from within the ~/.bashrc
unset OS_AUTH_TYPE
unset OS_AUTH_URL
...
unset OS_USER_ID
unset OS_VOLUME_API_VERSION
# could not find default tenant name for this user
# export OS_TENANT_NAME=""
export OS_USERNAME="fred"
export OS_PROJECT_DOMAIN_ID="84d9325c8ff341838cb02a78b76df8ce"
export OS_USER_DOMAIN_ID="84d9325c8ff341838cb02a78b76df8ce"
# Public Auth URL (used by users)
export OS_AUTH_URL="http://<load balancer IP address>:5000/v3"

# For keystone v3
export OS_IDENTITY_API_VERSION=3 # for the 'openstack' utility to work
export OS_CACERT="/etc/keystone/ssl/certs/ca.pem"
# END AUTOGENERATED SECTION   -- DO NOT REMOVE
```

The value of *<load balancer IP address>* in the `.openstackrc` output is a dotted quad value or a resolvable host name, and is the address or name of the HAProxy load balancer that Bright Cluster Manager uses for its OpenStack deployment. The load balancer address is normally the IP address of the head node on the external network on a smaller cluster.

Background Note: Changing The End User OpenStack Password

The end user is given a password for OpenStack user access by the initialization script. This password, stored in ~/.openstackrc_password, is long, and somewhat random. Most users would therefore like to change it to something that is easier for them to remember. This can be done in the dashboard by, for example, user fred, by clicking on the name fred in the top right hand corner, then selecting the Settings option, and then selecting the Change Password option.

The OpenStack APL CLI client openstack can be set to use the .openstackrc and .openstackrc_password files, which were initialized by the cm-user-init and cm-user-migration scripts earlier on (page 79). The end user can, if required, update the ~/.openstackrc_password file by hand after a password change is made by the dashboard.

5.2 Getting A User Instance Up

By default, after creating a user as in the example where user fred is created (page 80) the user can log in as an OpenStack user. However, unless something extra has been prepared, a user that logs in at this point has no instances up yet. End users typically want an OpenStack system with running instances.

In this section, getting an instance up and running is used to illustrate the management of OpenStack in Bright Cluster Manager.

5.2.1 Making An Image Available In OpenStack

A handy source of available images is at http://docs.openstack.org/image-guide/obtain-images.html. There is also a guide for creating images manually, from an ISO, at https://docs.openstack.org/image-guide/create-images-manually.html. Both of the URIs are for major, and some minor distributions, and they also include guidance for versions of Microsoft Windows.

Cirros is one of the distributions listed there. It is a distribution that aims at providing a small, but reasonably functional cloud instance. The Cirros image listed there can therefore be used for setting up a small standalone instance, suitable for an m1.xtiny flavor, which is useful for basic testing purposes.

Installing The Image Using The openstack Utility

If the qcow2 image file cirros-0.4.0-x86_64-disk.img, 12MB in size, is picked up from the site and placed in the local directory, then an image cirros040 can be set up and made publicly available by the administrator or user by using the openstack image create command as follows:

Example

```
[fred@bright81 ~]$ wget http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.img
```

To run an openstack command, as is done shortly, the OpenStack environment should be in place. If the .openstackrc file has been generated, then it can be sourced to provide the environment. The .openstackrc file is generated by setting the Write out OpenStack RC for users option (page 82), and it can be sourced with:

Example

```
[fred@bright81 ~]$ . .openstackrc
```

Sourcing in this case means that running the file sets the environment variables in the file, so that after returning to the shell the shell now has these environment variables.

The openstack command to create the Cirros image can now be run:

Example

```
[fred@bright81 ~]$ openstack image create --disk-format qcow2 --file cirros-0.4.0\
-x86_64-disk.img cirros040
```

If all goes well, then the image is installed and can be seen by the user or administrator, via OpenStack Horizon, by navigation to the `Images` pane, or using the URI `http://`*<load balancer hostname, or IP address>*`:10080/project/images/` directly (figure 5.3).

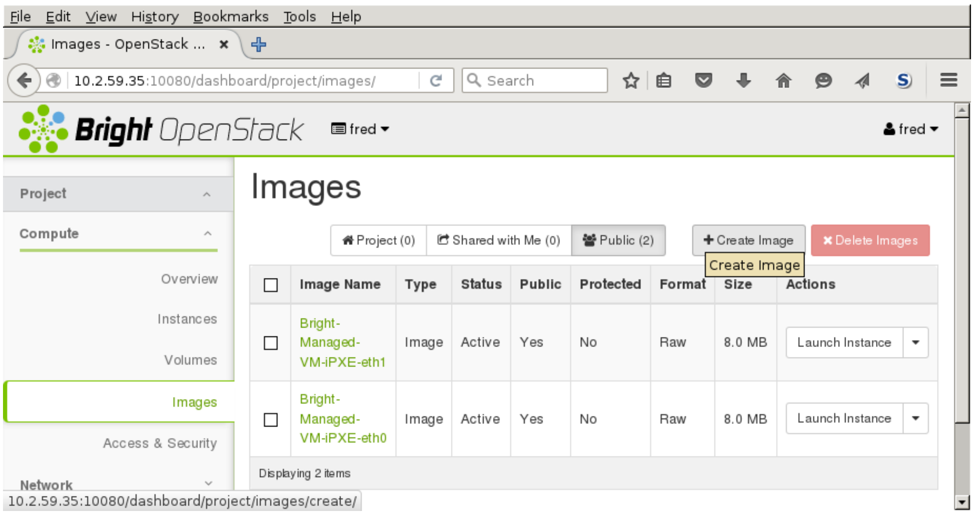

Figure 5.3: Images Pane In Horizon

Installing The Image Using Horizon

Alternatively, instead of using the `openstack` utility, the image can also be installed by the user or administrator using OpenStack Horizon directly. The Horizon procedure to do this is described next:

Clicking on the `Create Image` button of the `Images` pane launches a pop-up dialog. Within the dialog, a name for the image for OpenStack users can be set, the disk format of the image can be selected, the HTTP URL from where the image can be picked up can be specified, and the image can be made public (figure 5.4).

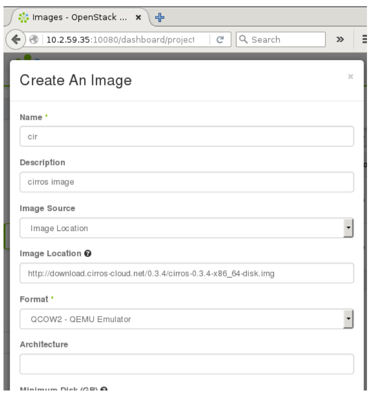

Figure 5.4: Images Pane—Create Image Dialog

The State Of The Installed Image

After the image has been installed by user `fred`, then it is available for launching instances by `fred`. If the checkbox for `Public` was ticked in the previous dialog, then other OpenStack users can also use it to launch their instances.

It should however be pointed out that although the image is available, it is not yet ready for launch. The reasons for this are explained shortly in section 5.2.2.

The image properties can be viewed as follows:

- by the authorized OpenStack users with OpenStack Horizon, by clicking through for `Image`

```
Details
```

- by `cmsh`, from within the `images` submode of `openstack` mode.
- using Bright View, via the clickpath `OpenStack→Compute→Images`

5.2.2 Creating The Networking Components For The OpenStack Image To Be Launched

Launching an image that is installed as in section 5.2.1 needs networking components to be configured with it, so that it can work within OpenStack, and so that it can be managed by OpenStack. An instance that is up, but has no networking set up for it, cannot launch an image to get a virtual machine up and running.

Why Use A New Network For An End User?

If it is the OpenStack administrator, `admin` that is preparing to launch the instance, as a `bright` project, then the OpenStack launch dialog by default allows the instance to use the default flat internal network of the cluster, `bright-internal-flat-internalnet`. As instances are run with root privileges, this means that all the internal network traffic can be read by whoever is running the instance. This is a security risk and would be a bad practice.

By default, therefore, the non-`admin` end user cannot launch the instance using the flat internal network of the cluster. The end user therefore typically has to create a new network, one that is isolated from the internal network of the cluster, in order to launch an instance.

This is thus the case for the end user `fred`, who earlier on had logged into the OpenStack dashboard and created an image by the end of section 5.2.1. User `fred` cannot run the image in the instance until a network exists for the future virtual machine.

Creating The Network With Horizon

For the sake of this example and clarity, a network can be created in OpenStack Horizon, using the `Network` part of the navigation menu, then selecting `Networks`. Clicking on the `Create Network` button on the right hand side opens up the `Create Network` dialog box.

In the first screen of the dialog, the network for `fred` can be given the unimaginative name of `frednet` (figure 5.5):

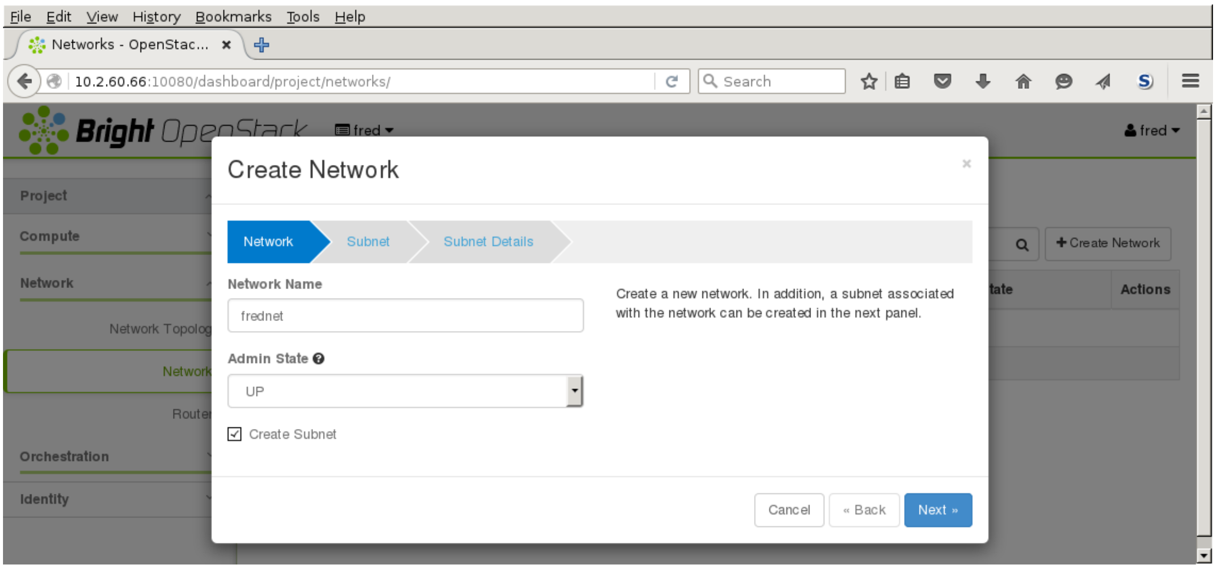

Figure 5.5: End User Network Creation

Similarly, in the next screen a subnet called `fredsubnet` can be configured, along with a gateway address for the subnet (figure 5.6):

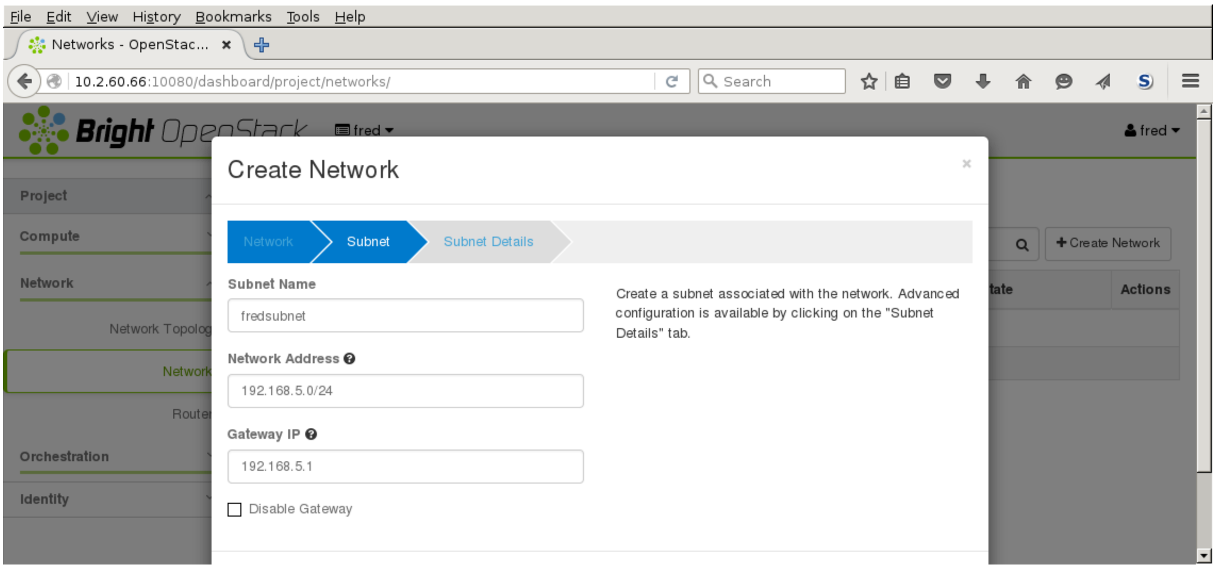

Figure 5.6: End User Subnet Creation

In the next screen (figure 5.7):

- a range of addresses on the subnet is earmarked for DHCP assignment to devices on the subnet
- a DNS address is set
- special routes for hosts can be set

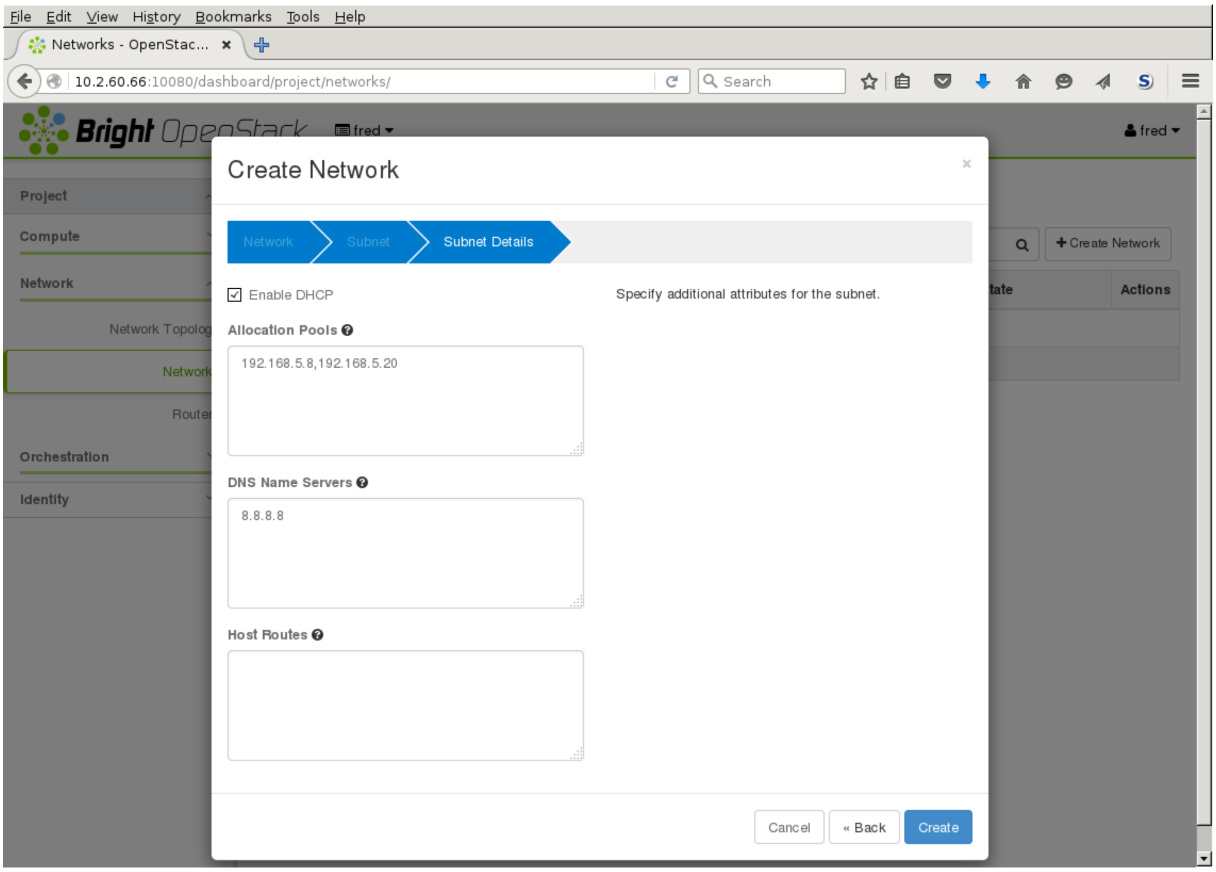

Figure 5.7: End User DHCP, DNS, And Routes

At the end of a successful network creation, when the dialog box has closed, the screen should look similar to figure 5.8:

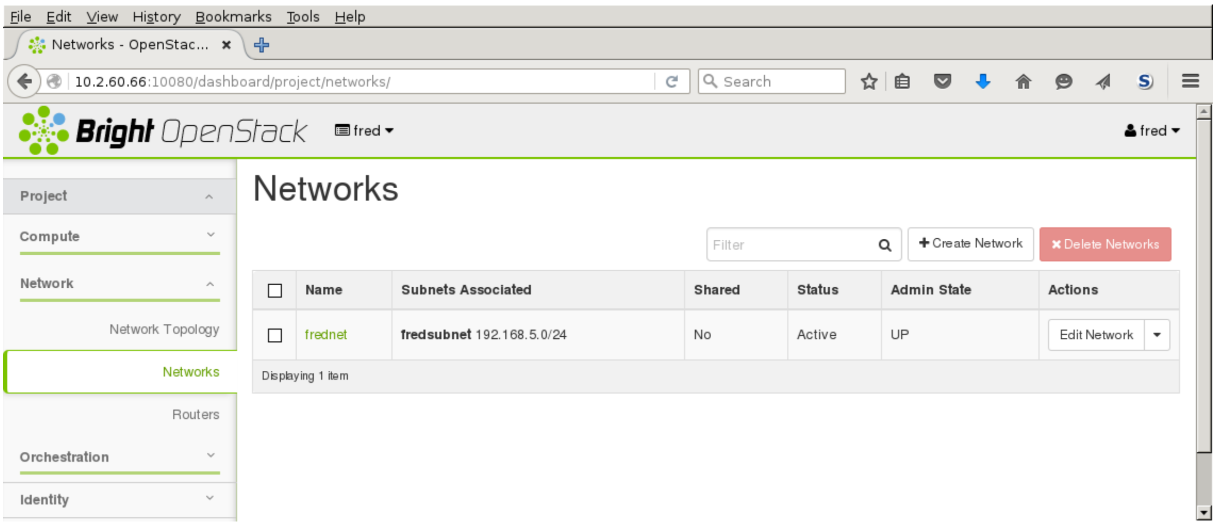

Figure 5.8: End User Node Network Configuration Result

The State Of The Image With Its Network Configured

At this point, the image can be launched, for example using Horizon's `Compute` resource in the navigation panel, then choosing the `Instances` pane, and then clicking on the `Launch Instance` button.

On launching, the image will run. However, it will only be accessible via the OpenStack console, which has some quirks, such as only working well in fullscreen mode in some browsers.

It is more pleasant and practical to login via a terminal client such as `ssh`. How to configure this is described next.

5.2.3 Accessing The Instance Remotely With A Floating IP Address

For remote access from outside the cluster, this is possible if a floating IP address, that is from the external network, has been configured for instances on the OpenStack network. The floating IP address is taken from the pool of addresses specified earlier during OpenStack installation (section 3.1.12). The subnet for these addresses needs to be accessible via a router. The configuration of such a router is described in the next subsection.

For remote access from within the cluster, an alternative method to creating a floating IP address, is for the administrator to configure the Bright Cluster Manager internal network to be a shared external network from the point of view of the instance. Sharing the internal network in this way is a security risk due to the reasons given earlier on on page 86. However, it may be appropriate in an isolated cluster with no external network, and with trusted users, in which case the administrator can mark the Bright Cluster Manager internal network from OpenStack Horizon as `shared`.

Remote access from outside the cluster with a floating IP address can be configured as follows:

Router Configuration For A Floating IP Address

Router configuration for a floating IP address with Horizon: A router can be configured from the `Network` part of the navigation menu, then selecting `Routers`. Clicking on the `Create Router` button on the right hand side opens up the `Create Router` dialog box (figure 5.9):

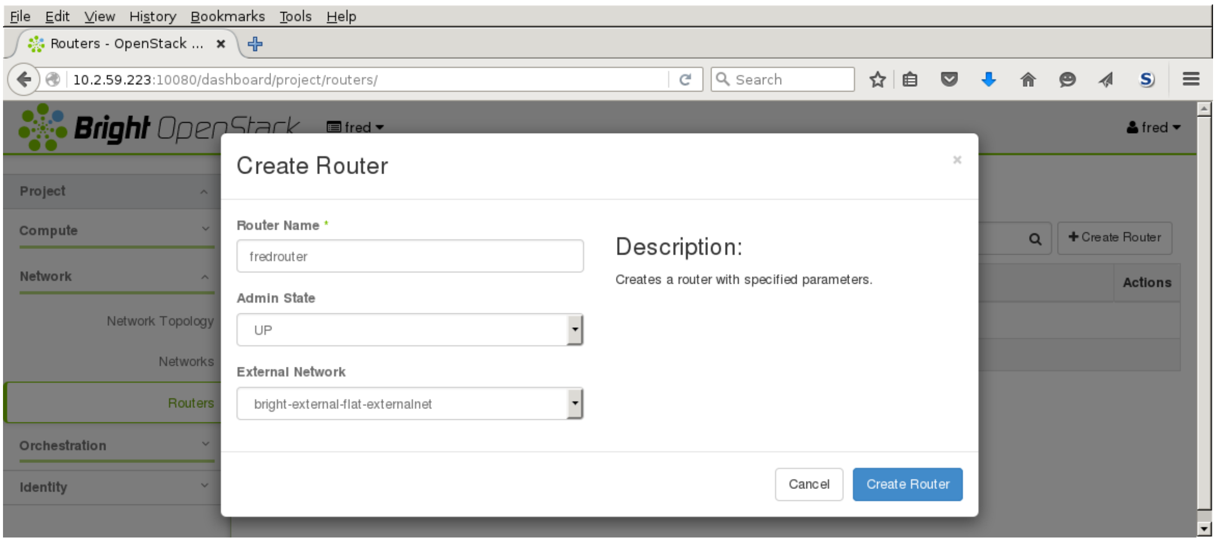

Figure 5.9: End User Router Creation

The router can be given a name, and connected to the external network that provides the floating IP addresses of the cluster.

Next, an extra interface for connecting to the network of the instance can be added by clicking on the router name, which brings up the `Router Details` page. Within the `Interfaces` subtab, the `Add Interface` button on the right hand side opens up the `Add Interface` dialog box (figure 5.10):

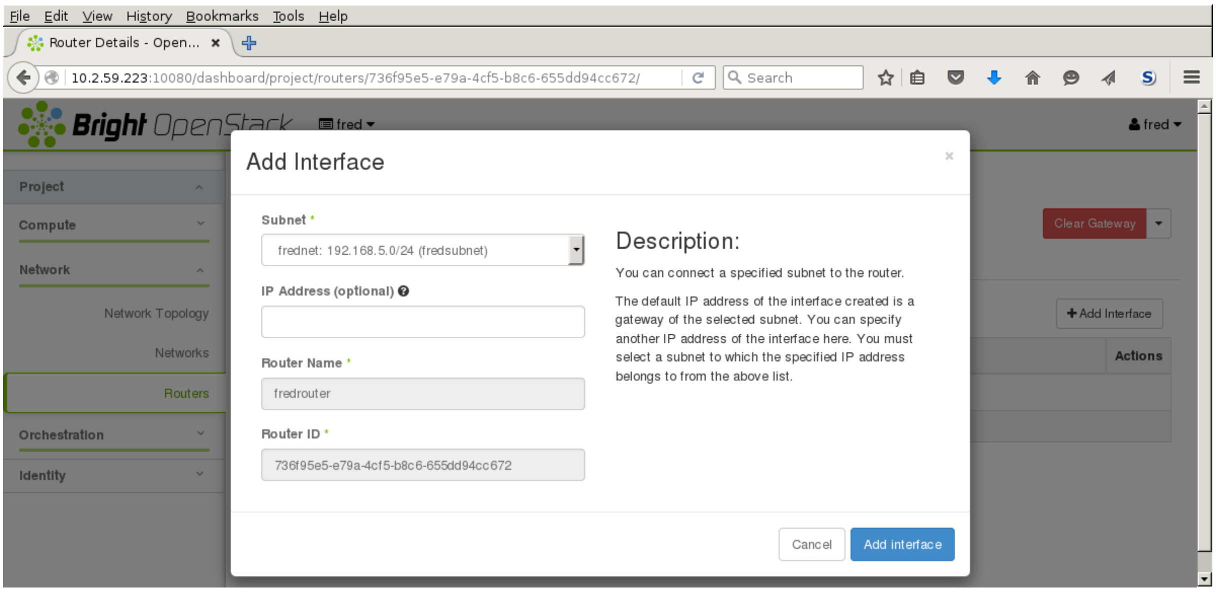

Figure 5.10: End User Router Interfaces Creation

After connecting the network of the instance, the router interface IP address should be the gateway of the network that the instance is running on (figure 5.11):

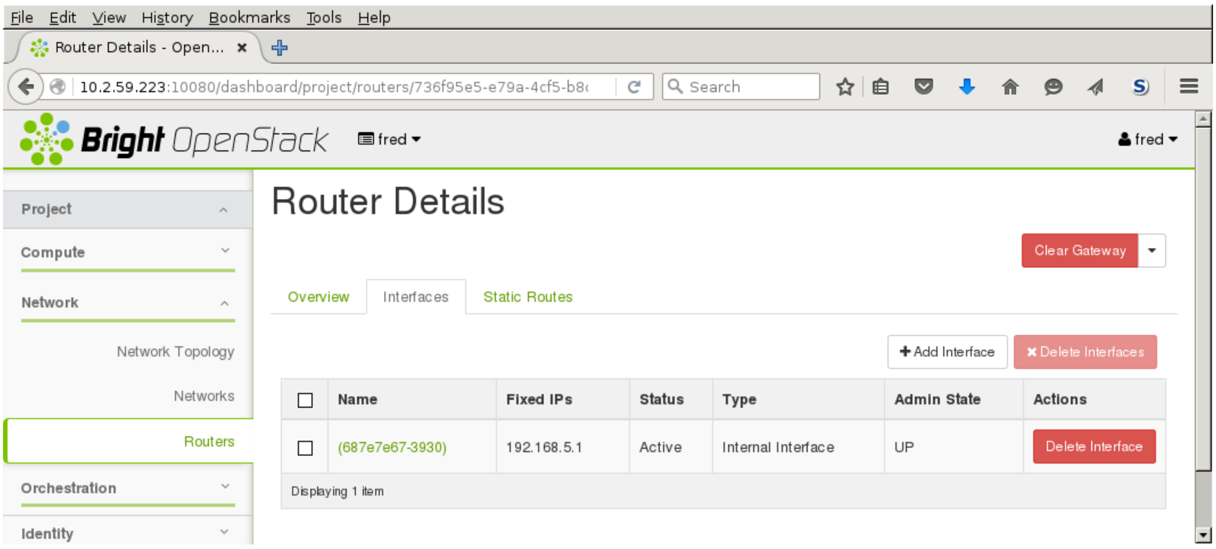

Figure 5.11: End User Router Interface Screen After Router Configuration

The state of the router after floating IP address configuration: To check the router is reachable from the head node, the IP address of the router interface connected to the cluster external network should show a ping response.

The IP address can be seen in the `Overview` subtab of the router (figure 5.12):

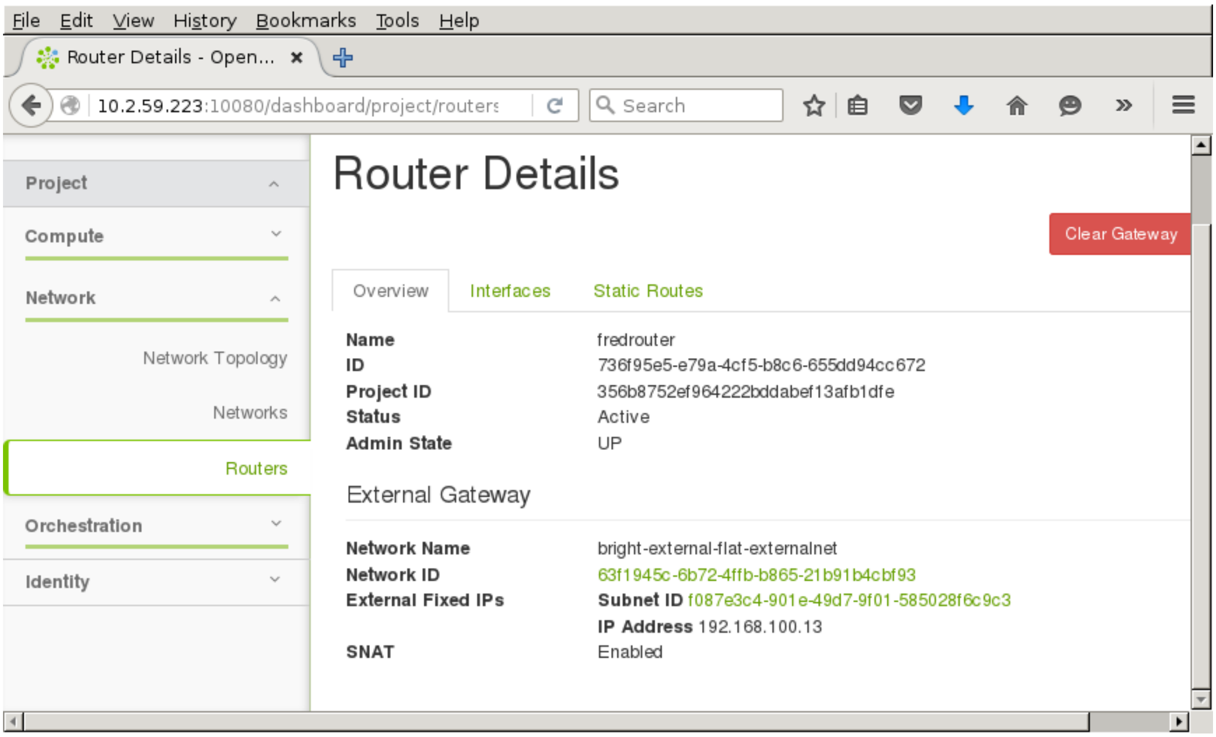

Figure 5.12: End User Router Details After Router Configuration

A ping behaves as normal for the interface on the external network:

Example

```
[fred@bright81 ~]$ ping -c1 192.168.100.13
PING 192.168.100.13 (192.168.100.13) 56(84) bytes of data.
64 bytes from 192.168.100.13: icmp_seq=1 ttl=64 time=0.383 ms

--- 192.168.100.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.383/0.383/0.383/0.000 ms
```

Security group rules to allow a floating IP address to access the instance: The internal interface to the instance is still not reachable via the floating IP address. That is because by default there are security group rules that set up iptables to restrict ingress of packets across the hypervisor.

The rules can be managed by accessing the `Compute` resource, then selecting the `Access & Security` page. Within the `Security Groups` subtab there is a `Manage Rules` button. Clicking the button brings up the `Manage Security Group Rules` table (figure 5.13):

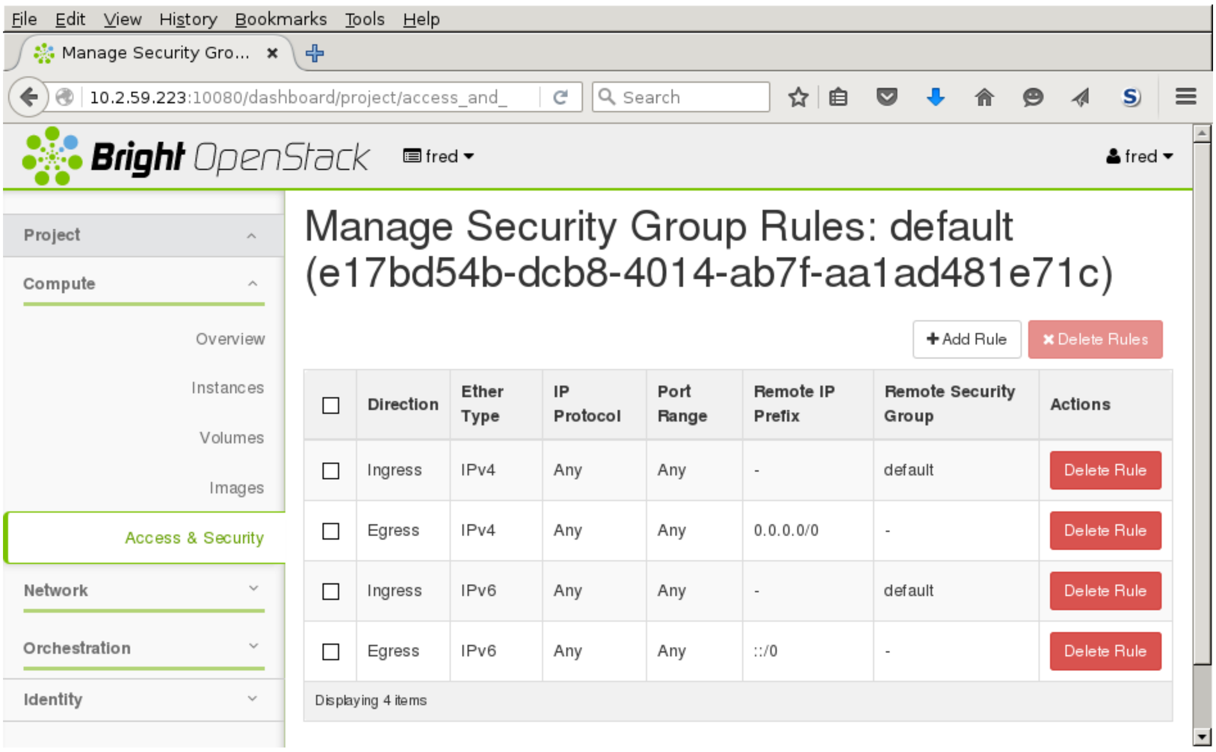

Figure 5.13: Security Group Rules Management

Clicking on the `Add Rule` button brings up a dialog. To let incoming pings work, the rule `All ICMP` can be added. Further restrictions for the rule can be set in the other fields of the dialog for the rule (figure 5.14).

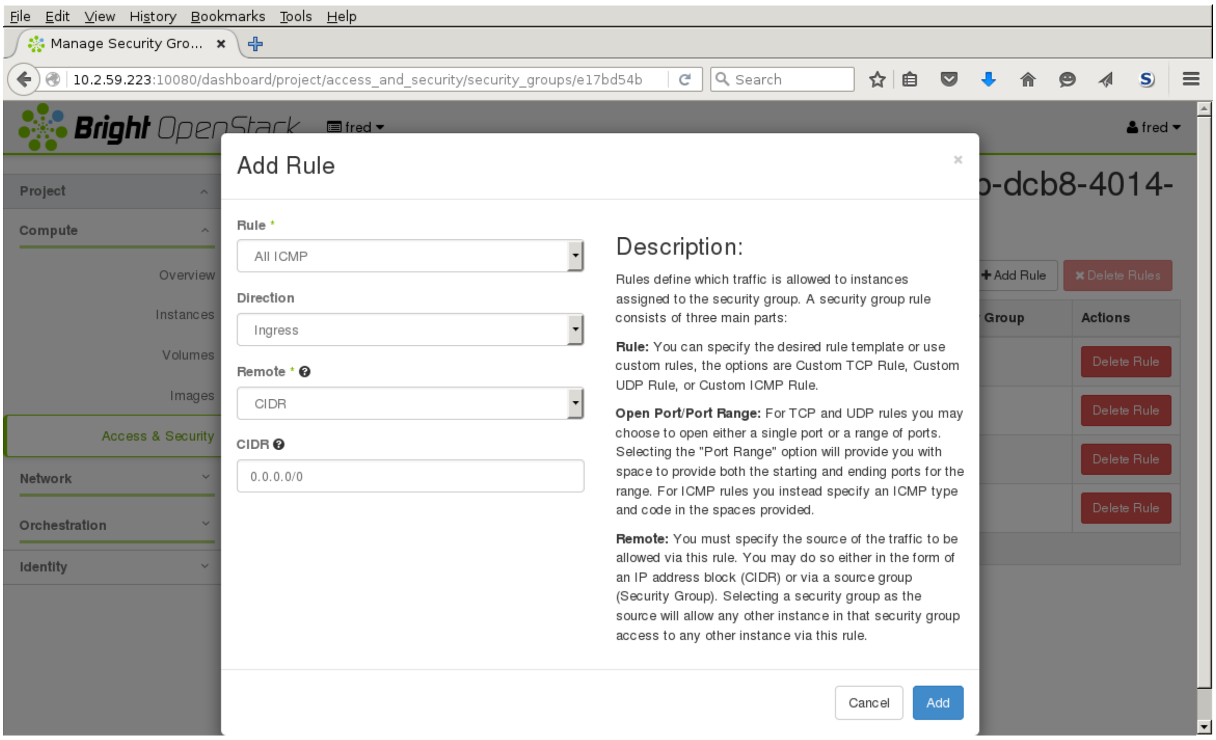

Figure 5.14: Security Group Rules Management—Adding A Rule

Floating IP address association with the instance: The floating IP address can now be associated with the instance. One way to do this is to select the `Compute` resource in the navigation window, and select `Instances`. In the `Instances` window, the button for the instance in the `Actions` column allows an IP address from the floating IP address pool to be associated with the IP address of the instance (figure 5.15).

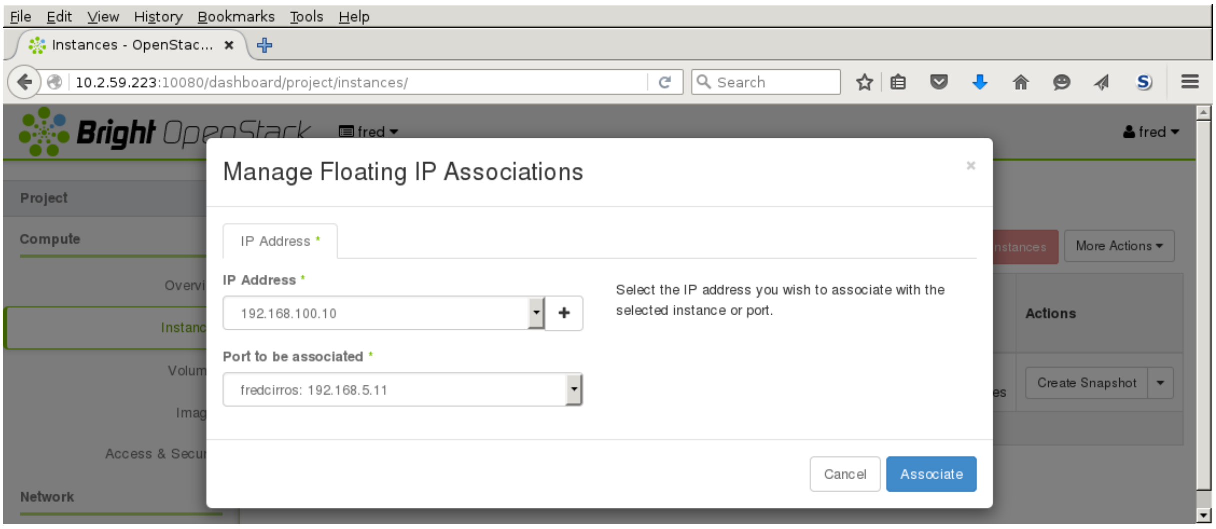

Figure 5.15: Associating A Floating IP Address To An Instance

After association, the instance is pingable from the external network of the head node.

Example

```
[fred@bright81 ]$ ping -c1 192.168.100.10
PING 192.168.100.10 (192.168.100.10) 56(84) bytes of data.
64 bytes from 192.168.100.10: icmp_seq=1 ttl=63 time=1.54 ms

--- 192.168.100.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.544/1.544/1.544/0.000 ms
```

If SSH is allowed in the security group rules instead of ICMP, then `fred` can run ssh and log into the instance, using the default username/password `cirros/cubswin:)`

Example

```
[fred@bright81 ~]$ ssh cirros@192.168.100.10
cirros@192.168.100.10's password:
$
```

Setting up SSH keys: Setting up SSH key pairs for a user `fred` allows a login to be done using key authentication instead of passwords. The standard OpenStack way of setting up key pairs is to either import an existing public key, or to generate a new public and private key. This can be carried out from the `Compute` resource in the navigation window, then selecting the `Access & Security` page. Within the `Key Pairs` subtab there are the `Import Key Pair` button and the `Create Key Pair` button.

- **importing a key option:** For example, user `fred` created in Bright Cluster Manager as in this chapter has his public key in `/home/fred/.ssh/id_dsa.pub` on the head node. Pasting the text of the key into the import dialog, and then saving it, means that the user `fred` can now login as the user `cirros` without being prompted for a password from the head node. This is true for images that are cloud instances, of which the `cirros` instance is an example.

- **creating a key pair option:** Here a pair of keys is generated for a user. A PEM container file with just the private key <*PEM file*>, is made available for download to the user, and should be placed in a directory accessible to the user, on any host machine that is to be used to access the instance. The corresponding public key is stored in Keystone, and the private key discarded by the generating machine. The downloaded private key should be stored where it can be accessed by `ssh`, and should be kept read and write only. If its permissions have changed, then running `chmod 600` <*PEM file*> on it will make it compliant. The user can then login to the instance using, for example, `ssh -i` <*PEM file*> `cirros@192.168.100.10`, without being prompted for a password.

The `openstack keypair` options are the CLI API equivalent for the preceding Horizon operations.

Setting up SSH key pairs in this way relies on a properly functioning `cloud-init`. `cloud-init` is a set of initialization utilities that is part of the image available for the VMs that run under OpenStack (section 5.2.1). It is `cloud-init` that gets the VMs contact the OpenStack metadata server to pick up the public key and place it in the proper location on the VMs.

5.3 Running A Bright-managed Instance

A Bright-managed instance is a special case of the user instance in section 5.2. A Bright-managed instance is a virtual machine that is treated very similarly to a regular node by Bright Cluster Manager, and runs by default as a *vnode*. For example, it runs with the default names of `vnode001`, `vnode002...` rather than a `node001`, `node002` and so on. The default number of `vnodes` that is set, if Bright-managed instances are enabled, is 5, altough this number can be altered during OpenStack installation. The number of vnodes can be modified after installation in several ways, including:

- by adding a vnode as a node of type `virtualnode`, in the `device` mode of `cmsh`, or via Bright View's clickpath of `OpenStack`→`Virtual Nodes`
- by cloning an existing vnode and modifying it if needed

Since Bright Cluster Manager is integrated tightly with vnodes, getting a Bright-managed instance running is much easier than the procedure for user instances described earlier in sections 5.1 and 5.2. It is also a cluster administrator that typically creates Bright-managed instances, which run under the `bright` project, whereas it is end users that typically create regular VM instances, which typically run under a non-`bright` project name.

To get a default vnode up, it can be powered up from `cmsh`:

Example

```
[root@bright81 ~]# cmsh -c "device power on vnode001"
```

or it can be powered up from Bright View via the clickpath `Openstack`→`Virtual Nodes`→`Power`

Most settings for vnodes are like those for regular nodes.

The main exceptions are accessible via the vnode `Virtual node settings` option. This option is available via the clickpath `Openstack`→`Virtual Nodes`→`Edit`→`Virtual node settings`. It brings up a window that allows, among others, a `Flavor` to be set.

The end user typically notices very little difference between `vnodes` and regular nodes.

6

Cluster-On-Demand For OpenStack

6.1 Introduction

If Bright OpenStack is running on a cluster, then ordinary users of the cluster can run a Cluster On Demand (COD) within it. A COD is a complete cluster that is managed by Bright Cluster Manager, which means that a regular user can become a cluster administrator of a virtual cluster running under the hosting cluster. This kind of setup is a case of running a cluster within a cloud service, with the cloud in this case being the Bright OpenStack. Analogous setups where the cloud service is provided by AWS or Azure can also be managed with Bright Cluster Manager (Chapter 2 of the *Cloudbursting Manual*). In Bright jargon these setups are conveniently called COD-OS, COD-AWS, and COD-Azure.

Bright provides a client, `cm-cod-os`, to launch a COD within a Bright OpenStack cluster. The client can run on the head node of the cluster itself, or from a remote location.

Some possible uses for a Bright OpenStack COD are:

- a staging environment to test the production configuration of a software running on a Bright cluster
- a way to do batch job processing using whatever workload manager the user would like, instead of being limited to one due to the possiblity of the workload managers interfering with each other.
- a way to run virtual Bright clusters to try things out on, so that a user can become familiar with administrating Bright cluster, without breaking a production system.

The `cm-cod-os` client is provided by the `cm-cluster-on-demand-openstack` package:

Example

```
[root@bright81 ~]# yum install cm-cluster-on-demand-openstack
```

The `cm-cod-os` client is a Python script that can be run with configuration files (section 6.3) and expects some environment variable settings. If arguments are used by the script, then the arguments override the corresponding configuration file values.

The `cm-cod-os` client is usually run by an ordinary user, and not the administrator, of the host cluster. The ordinary user then typically becomes the owner and administrator of the COD. The COD is thus a virtual cluster that is hosted by the host cluster. This is called a nested cluster.

6.2 The `cm-cod-os` Arguments

6.2.1 The `cm-cod-os` Top Level Arguments

Options to `cm-cod-os` can be viewed with the help option, `-h|--help`:

Example

```
[fred@bright81 ~]$
usage: cm-cod-os [-h] [--config CONFIG] [--no-system-config] [-v]
                 [--show-configuration]
                 cluster,c,cluster create,cc,cluster list,cl,cluster
                 delete,cd,cremove,node,n,image,i,image list,il,image
                 delete,id,vnc,v,vnc list,vl,flavor,config ...

Cluster-on-demand by Bright Computing

positional arguments:
  cluster,c,cluster create,cc,cluster list,cl,cluster delete,cd,cremove,node,n,image,i,\
image list,il,image delete,id,vnc,v,vnc list,vl,flavor,config
    cluster (c)         Cluster operations
    cluster create (cc)
                        Create cluster
    cluster list (cl)   List clusters
    cluster delete (cd,cremove)
                        Delete all resources in a cluster
    node (n)            Node operations
    image (i)           Image subcommands
    image list (il)     Lists cluster images
    image delete (id)   Deletes image from glance
    vnc (v)             VNC subcommands
    vnc list (vl)       List VNC ports
    flavor              Flavor subcommands
    config              Configuration operations

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG, -c CONFIG
  --no-system-config
  -v, -vv, -vvv
  --show-configuration
```

6.2.2 The `cm-cod-os` Context Tree

`cm-cod-os` has a hierarchy of options. Some special positional subcommand options allow particular hierarchies to be accessed. The context of such a special positional subcommand option decides the hierarchy available.

In Bright terminology, such a "special positional subcommand option" is therefore more conveniently called a "context". If the context is one level deeper in the hierarchy, then it is called a "subcontext" for precision. However, in more loose usage, the word "context" is generally just assumed to include the idea of "subcontext".

The syntax is indicated by:

`cm-cod-os [`*context*` [`*subcontext*`]] `*options*

The `cm-cod-os` tree has its context and subcontext branches organized as follows:

```
cod
   + cluster
            +create
            +list
            +delete
            +description
```

```
          +shelve
          +show
          +start
          +stop
          +tag

 + config
          +dump
 + node
          +create
 + image
          +list
          +stats
          +repo-list
          +install
          +download
          +repo-download
          +delete
          +show
          +update
          +usage
 + vnc
          +list
 + flavor
          +list
```

6.2.3 The cm-cod-os Contexts And Optional Arguments Help Text

Most contexts and subcontexts, like at the top level of cm-cod-os, have around 10 contexts and argument options or less. For example, the cm-cod-os→cluster help text has the following contexts and options (some output ellipsized):

Example

```
[fred@bright81 ~]$ cm-cod-os cluster -h
usage: cm-cod-os cluster [-h]

                         {create,c,list,l,delete,d,remove,description,shelve,show,start,stop,tag}
                         ...

positional arguments:
  {create,c,list,l,delete,d,remove,description,shelve,show,start,stop,tag}
    create (c)          Create cluster
    list (l)            List clusters
    delete (d,remove)   Delete all resources in a cluster
    description         View/change cluster description
    shelve              This command shelves all instances in a cluster. It
                        performs a clean shutdown first.
    show                This command shows details for a (list of) cluster(s)
                        and all related objects.
    start               Start/unshelve cluster(s)
    stop                Stop cluster(s)
    tag                 Set tags on a heat stack to turn it into a cluster

optional arguments:
  -h, --help            show this help message and exit
```

The `cm-cod-os cluster create` Options Help Text

Probably the most used path in the `cm-cod-os` command hierarchy for a user is `cm-cod-os→cluster→create`. This can be used to create a cluster in many different ways, so naturally it has the most options.

Example

```
[fred@bright81 ~]$ cm-cod-os cluster create -h
usage: cm-cod-os cluster create [options] -- label

positional parameters:
  label                 A custom label for this cluster, appended to the
                        autogenerated name. Use --name to force the entire
                        name.

required parameters missing a value:
  --license-product-key LICENSE_PRODUCT_KEY
                        Bright Cluster Manager Product Key.

cluster create parameters:
  -n NODES, --nodes NODES
                        Number of compute nodes to be created according to one
                        of the following formats (several space separated
                        groups can be specified): node_number
                        node_number:node_flavor node_number:node_template_path
                        node_number:node_template_path:node_flavor You find an
                        example of what node templates look like in:
                        /usr/lib/python2.7/site-
                        packages/clusterondemandopenstack/static/cod-os-node-
                        template.yml.
  --node-disk-setup-path PATH_TO_XML
                        Path to the XML file with the disk setup which is to be
                        used for the nodes.
  --node-disk-setup NODE_DISK_SETUP

  --wlm {lsf,torquemoab,pbspro,dont-configure,slurm,pbspro-ce,torque,uge,torquemaui,sge}
                        Workload Manager of choice. dont-configure - do not
                        configure any (can be configured later).
  --copy-file COPY_FILE
                        Colon separated source and destination path, or just
                        the source path which will then also be used as the
                        destination path for the file that is copied to the
                        headnode. Note: the keyword {COD_BRIGHT_VERSION} will
                        be replaced by the cod bright version.
  --copy-file-with-env COPY_FILE_WITH_ENV
                        Same as --copy-file but replaces instances of
                        ${ENV_VAR} inside of the file being copied with the
                        content of the environment variable 'ENV_VAR' as well
                        as {COD_BRIGHT_VERSION} by the bright version in the
                        file path.
  --name NAME           (default: auto) Name of the cluster to createBy default
                        the name is generated from version, distro, label.
  --timezone TIMEZONE   Timezone of the cluster.
  --store-head-node-ip PATH_TO_FILE
```

```
                        Once the cluster has been created, store the IP of the
                        head node in a file. Useful for automation.
  --ask-to-confirm-cluster-creation {TRUE,FALSE}
                        Ask for confirmation when creating a new cluster. Use
                        -y to skip this question.
  --run-cm-bright-setup {TRUE,FALSE}
                        Whether or not to initialize the cluster by running cm-
                        bright-setup (activate license, etc).
  -m HEAD_NODE_TYPE, --head-node-type HEAD_NODE_TYPE
                        Flavor for the head node. Use flavor name from 'nova
                        flavor-list'.
  --head-node-availability-zone HEAD_NODE_AVAILABILITY_ZONE
                        Name of the availability zone to create the head node
                        on. If not specified, OpenStack's scheduler will
                        decide. This argument can also be used to force the
                        head node onto a specific hypervisor. To do so, specify
                        "<availability_zone>:<hypervisor>", e.g.
                        "default:hyper01"..
  --ssh-key-pair SSH_KEY_PAIR
                        Name of the key pair used to access the head node.
  --internal-cidr INTERNAL_CIDR
                        CIDR of the cluster's internal network..
  --internal-mtu NUMBER
                        MTU of the cluster's internal network..
  --ingress-ports INGRESS_PORTS
                        Ports to open on the head node. 8081 is CMDaemon
                        (CMGUI).
  --ingress-cidr INGRESS_CIDR
                        CIDR (IP range) which will be allowed to attempt access
                        the head node from the Internet. 0.0.0.0/0 means it
                        will be possible to connect from anywhere on the
                        Internet.
  --ingress-icmp {TRUE,FALSE}
                        Whether to enable incoming ICMP.
  --wait-ssh SECONDS    Wait up to that many seconds for SSH to come up.
  --wait-cmdaemon SECONDS
                        Wait up to that many seconds for CMDaemon to come up.
  --prebs COMMAND       Command(s) executed by cloud-init before cm-bright-
                        setup (before CMDaemon starts). Useful for package
                        update. Multiple arguments are allowed.
  --postbs COMMAND      Command(s) executed by cloud-init post cm-bright-setup
                        (Once CMDaemon starts).
  --append-to-root-bashrc ENV=VAR
                        Lines to append to the /root/.bashrc file on the head
                        node.
  -d, --dry-run         Dry run - do not actually create the cluster. Useful
                        with --template.
  --description DESCRIPTION
                        Cluster description.
  -t OUTPUT_FILE, --template OUTPUT_FILE
                        Generate resulting heat template to the file. Use '-'
                        as the file name to output the template to stdout.
                        Useful with --dry-run.
  -y, --yes             Do not ask for confirmation when creating a new
                        cluster.
```

```
  --wait-for-nodes SECONDS
                        Wait for up to that many seconds for the compute nodes
                        to come up.
             (35 additional parameters can be displayed with the --advanced-help argument)


image selection parameters:
  --version VERSION     Bright Cluster Manager version.
  --distro DISTRO       Linux distribution name.
  --package-groups PACKAGE_GROUPS
                        Package group.
  --image IMAGE_SPEC    Single image selector statement. See cm-cod-os image
                        list --help. Overrides filter arguments such as
                        --version, --distro, etc.
  --head-node-image HEAD_NODE_IMAGE
                        Single image selector statement for head node image (as
                        in advanced mode). Overrides the head node image
                        selected by --image and all other image filter
                        arguments.
  --node-image NODE_IMAGE
                        Single image selector statement for node image (as in
                        advanced mode) '--node-image none' will force the
                        cluster to not use a node image at all. Overrides the
                        head node image selected by --image and all other image
                        filter arguments.
  --tags TAGS           Single image selector statement. See cm-cod-os image
                        list --help.
  --status STATUS       Glance status of the image.
  --cmd-revision-min NUMBER
                        Minimum CMDaemon revision required.
  --cmd-revision-max NUMBER
                        Maximum CMDaemon revision.
  --revision NUMBER     Select clusters with specified revision.
             (4 additional parameters can be displayed with the --advanced-help argument)


root login method to the head node:
  --log-cluster-password
                        Log cluster password to the screen and log files. This
                        option is mandatory if no custom password, nor SSH
                        keypairs, were specified.
  --cluster-password CLUSTER_PASSWORD
                        The root user password to the cluster. If not
                        specified, a random one will be generated (use --log-
                        cluster-password to see it). This is also the root user
                        SQL password on the head node. Upon cluster creation
                        the password is stored in the
                        /cm/local/apps/cmd/etc/cmd.conf on the head node.
  --ssh-password-authentication {TRUE,FALSE}
                        If set to true, it will be possible to SSH to the head
                        node using a password. This option should NOT be used
                        in untrusted environments as it exposes the head node
                        to brute force login attacks.
  --no-access-validation
                        Force the cluster creation process even when specified
```

```
                         argument combinations does not guarantee that the
                         access the cluster will be possible (e.g. no SSH key
                         and no random password logging was set). Useful if e.g.
                         the public SSH key is being delivered to the image in
                         some other way, than the usual command line argument.
                         Note, that access validation does not attempt to
                         actually connect to the cluster. Instead, it merely
                         tries to predict whether the cluster will be accessible
                         to the user, given the specified argument combination.
  --ssh-pub-key-path PATH_TO_FILE
                         Path to the public key.
               (2 additional parameters can be displayed with the --advanced-help argument)


node volume parameters:
  --node-root-volume-size SIZE_IN_GB
                         Root volume size in GB.
  --node-root-volume-type NODE_ROOT_VOLUME_TYPE
                         Compute node root disk volume type. Allows for
                         specifying a special volume type with different Quality
                         of Service policy (more IOPS etc).
  --head-node-root-volume-size SIZE_IN_GB
                         Head node root disk size in GB. Should be bigger than
                         the Image size.
  --head-node-root-volume-type HEAD_NODE_ROOT_VOLUME_TYPE
                         Head node root disk volume type. Allows for specifying
                         a special volume type with different Quality of Service
                         policy (more IOPS etc).
  --head-node-extra-volume-type HEAD_NODE_EXTRA_VOLUME_TYPE
                         Head node extra disk volume type. Allows for specifying
                         a special volume type with different Quality of Service
                         policy (more IOPS etc).
  --head-node-extra-volume-size SIZE_IN_GB
                         Second volume for the extra.
               (1 additional parameters can be displayed with the --advanced-help argument)


Bright Cluster Manager licensing information:
  --license-unit LICENSE_UNIT
                         License unit.
  --license-locality LICENSE_LOCALITY
                         License locality.
  --license-country LICENSE_COUNTRY
                         Two characters.
  --license-product-key LICENSE_PRODUCT_KEY
                         Bright Cluster Manager Product Key.
  --license-organization LICENSE_ORGANIZATION
                         Name of your organization.
  --license-state LICENSE_STATE
                         Name of your state or province.
               (1 additional parameters can be displayed with the --advanced-help argument)


common parameters:
  -h, --help             Show this message and exit.
```

```
  --advanced-help        Don't omit advanced configuration parameters from the
                         help output. Implies --help.
  -v, -vv, -vvv, --verbose
                         Verbosity level.
               (5 additional parameters can be displayed with the --advanced-help argument)


configuration parameters:
  -c CONFIG, --config CONFIG
                         Extra config files.
               (8 additional parameters can be displayed with the --advanced-help argument)
```

6.3 The cm-cod-os Configuration Files

6.3.1 The cm-cod-os Configuration Files Locations

By default, configuration files for setting up the COD are searched for in the following locations, and in the following sequence. Settings found earlier in the sequence are overwritten by settings later on in the sequence.

- /etc/cm-cluster-on-demand.ini
- /etc/cm-cluster-on-demand.conf
- /etc/cm-cluster-on-demand.d/*
- ~/cm-cluster-on-demand.ini
- ~/cm-cluster-on-demand.conf
- ~/cm-cluster-on-demand.d/*
- a file location specified on the command line. For example, a file mycodsettings can be accessed using the --config option of cm-cod-os:

  Example

  ```
  [fred@bright81 ~]$ cm-cod-os --config mycodsettings
  ```

Typically, the administrator sets up configuration options in one of the first 3 locations, and the regular user modifies the options or adds other options in one of the last 4 locations.

6.3.2 Viewing The cm-cod-os Configuration File Options

A dump of the existing configuration can be viewed using cm-cod-os config dump

To check what options have been applied, and their sequence, the log to STDOUT can be viewed if the -v|--verbose option has been applied.

A list of configuration options for the cm-cod-os cluster create command can be seen with the --show-configuration option (output truncated):

Example

```
[fred@bright81 ~]$ cm-cod-os cluster create --show-configuration
+-------------------------------------------+------------------------------------------+
| option                                    | value                                    |
+-------------------------------------------+------------------------------------------+
| advanced_help                             | False (default)                          |
| append_to_root_bashrc                     | [] (default)                             |
```

```
| ask_to_confirm_cluster_creation          | True (default)                                |
| cluster_password                         | None (default)                                |
| cmd_revision_max                         | None (default)                                |
...
```

Arguments to cm-cod-os override the equivalent configuration file settings. This means that the configuration file settings of a working configuration can be used as a default template, and modifications to the template can conveniently be carried out via command line.

6.3.3 Setting The cm-cod-os Configuration File Options And Corresponding Arguments

The configuration options can be placed under sections that are associated with the corresponding cm-cod-os contexts and subcontexts of the tree in section 6.2.2.

For example, the path to the subcontext cm-cod-os cluster create has a large number of possible options (the options listed starting on page 98). The configuration file for the options can then have a section that begins with:

```
[openstack.cluster.create]
```

The section and options that can be placed in a configuration file can be worked out from the help text output of the cm-cod-os for the associated context or subcontext.

For example, the help text for cm-cod-os cluster create -h has the excerpt:

```
  --ask-to-confirm-cluster-creation {TRUE,FALSE}
                        Ask for confirmation when creating a new cluster. Use
                        -y to skip this question.
```

The configuration option for this option then takes the form:

```
openstack.cluster.create.ask_to_confirm_cluster_creation
```

The options are placed in a key=value format under the associated section.

Thus, the option that can be set to ask for confirmation before creating a new cluster is then ask_to_confirm_cluster_creation. It can be placed under the section [openstack.cluster.create] in key=value format as follows:

```
[openstack.cluster.create]
ask_to_confirm_cluster_creation=yes
```

6.4 The cm-cod-os Environment Variables

The environment in which cm-cod-os runs also provides the script with information via OpenStack environment variables.

These OpenStack variables are typically exported in the .bashrc or .openstackrc file (page 82) for the COD owner using the Bright OpenStack cluster. The environment variables, which are prefixed with OS_, then typically exist in the environment of the OpenStack COD owner.

Example

```
[fred@bright81 ~]$ grep OS_ .bashrc
export OS_AUTH_URL="http://master:5000/v3"
export OS_PROJECT_NAME="${USER}-project"
export OS_USERNAME="${USER}"
export OS_TENANT_NAME="${USER}-project"
export OS_PROJECT_DOMAIN_ID="9b9d86bb35934072b7c2a5c73ce75d43"
export OS_USER_DOMAIN_ID="9b9d86bb35934072b7c2a5c73ce75d43"
```

```
export OS_IDENTITY_API_VERSION=3
export OS_CACERT="/etc/keystone/ssl/certs/ca.pem"
export OS_INITIALS=$COD_PREFIX
[fred@bright81 ~]$
```

So, when the cm-cod-os script runs, it works for the COD owner with the local Bright OpenStack cluster by default.

To have cm-cod-os work with other clusters requires appropriate changes in these environment variables, as well as in the cm-cod-os configuration options.

6.5 Launching A COD

This section consists of example sessions, to show how the material in the preceding sections of this Chapter can be used to launch a nested COD. Some administrative preparation is first carried out on the host cluster (subsection 6.5.1). Once the host cluster is ready, the nested cluster can be launched and configured (subsection 6.5.2).

6.5.1 Administrative Preparation Of The Host Cluster

It is assumed that the host cluster is configured with Bright OpenStack already. For example, with Bright View OpenStack wizard (section 3.1), or with cm-openstack-setup (section 3.2).

Installing cm-cod-os

On the head node, the cm-cod-os utility should be present. If it is missing, then it should be installed with:

Example

```
[root@bright81 ~]# yum install cm-cluster-on-demand-openstack
```

Setting The OpenStack Port Security Extension Driver

The OpenStack port security plugin (https://wiki.openstack.org/wiki/Neutron/ML2PortSecurityExtensionDriver) should be enabled to allow the toggling of packet filtering for the hosted devices, so that the hosted nodes can be served DHCP leases.

```
[root@bright81 ~]# cmsh
[bright81]% openstack
[bright81->openstack[default]]% settings
[bright81->openstack[default]->settings]% networking
[...settings->networking]% set enableml2portsecurityplugin yes
[...settings*->networking*]% commit
```

Viewing The Existing Flavors

If the OpenStack credentials and environment are in place, then the standard flavor list provided by Bright OpenStack can be seen by using the OpenStack CLI client. The root user can see these with:

Example

```
[root@bright81 ~]# source .openstackrc; source .openstackrc_password
[root@bright81 ~]# openstack flavor list
+------------------------------+-----------+-------+------+-----------+-------+-----------+
| ID                           | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+------------------------------+-----------+-------+------+-----------+-------+-----------+
| 0f57527b-4fb2-452d-9b52-13c2 | m1.large  |  8192 |   20 |        80 |     4 | True      |
| 5309e46d-5c2d-47a0-9ad8-dae9 | m1.xtiny  |   256 |    1 |         0 |     1 | True      |
| 9b711a53-282c-4901-8bef-20ac | m1.xlarge | 16384 |   40 |       160 |     8 | True      |
| ccbad1b4-bdb2-40fe-b948-1819 | m1.tiny   |   512 |    5 |         5 |     1 | True      |
```

```
| e6408d86-a9ca-4694-9066-f8a8 | m1.medium |  4096 |   20 |         40 |      2 | True       |
| ebf16a3f-c3ee-41e7-952d-2643 | m1.xsmall |  1024 |   10 |         10 |      1 | True       |
| f66090cd-b020-49a2-808c-393b | m1.small  |  2048 |   10 |         20 |      2 | True       |
+------------------------------+-----------+-------+------+-----------+-------+-----------+
```

Alternatively, without having to explicitly source the credentials and the environment, the Bright Cluster Manager `cmsh` equivalent can be run:

Example

```
cmsh -c "openstack; flavors; list"
```

Adding COD Flavors

Some arbitrary COD flavors can be defined by the hosting administrator according to the possible requirements. A convenient set could be:

Example

```
[root@bright81 ~]# openstack flavor create --ram 1024 --vcpus 1 cod.xsmall
[root@bright81 ~]# openstack flavor create --ram 2048 --vcpus 2 cod.small
[root@bright81 ~]# openstack flavor create --ram 4096 --vcpus 2 cod.medium
[root@bright81 ~]# openstack flavor create --ram 8192 --vcpus 4 cod.large
[root@bright81 ~]# openstack flavor create --ram 16384 --vcpus 8 cod.xlarge
```

Alternatively, if using `cmsh`, then the flavors can be set within the `flavors` submode of `openstack` mode. For example, the preceding `cod.xsmall` flavor can be set with:

Example

```
cmsh -c "openstack flavors; add cod.xsmall; set ram 1GiB; set vcpus 1; commit"
```

Creating A Volume Type

A volume is a block storage that can be used for persistent storage and attached to instances. A simple volume type with default properties can be created by the root user, and given the unimaginative name of `default`, with:

Example

```
[root@bright81 ~]# openstack volume type create default
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description | None                                 |
| id          | 6486cff8-c2bf-4a3d-a32e-3a67b94ca564 |
| is_public   | True                                 |
| name        | default                              |
+-------------+--------------------------------------+
```

If the user has access to `cmsh`, then another way to set this would be: `cmsh -c "openstack; volumetypes; add default; commit"`

At this stage in the session, the following points may help orient the reader:

OpenStack client commands typically have `cmsh` or Bright View equivalents: As the preceding examples illustrate, the more useful of the OpenStack client commands can be carried out with a `cmsh` or Bright View equivalent. Whatever is used is a matter of preference and convenience.

Non-root methods to carry out the OpenStack client commands: So far in this section (6.5.1) the preparation to launch a COD has been carried out as a root user. However a non-root user can run these tasks too, with the right credentials.

If using the OpenStack client, then the appropriate credentials for a user are created when an OpenStack user with the same name as the Bright user is created (section 5.1.2). Thus, for example, if a non-root user who is the Bright user `fred` is created as an OpenStack user too, then `fred` becomes able to view the flavor list too, as follows:

Example

```
[fred@bright81 ~]$ source .openstackrc; source .openstackrc_password
[fred@bright81 ~]$ openstack flavor list
```

OpenStack flavors are a superset of COD flavors. To view only COD flavors—flavors that are used by COD instances only—the following command can be run, as root or as the regular user:

Example

```
cod flavor list
```

It should be noted that the `cmsh` or Bright View equivalents for commands in this section (6.5.1) can always be run by a non-root user. However, to be able to run them, the non-root user must have sufficient privileges. Such privileges can be set by the host cluster administrator modifying the `profile` settings (section 6.4 of the *Administrator Manual*) for the non-root user.

Generally, the preparation in this section (6.5.1) is done as a root user. In the following section (6.5.2), the non-root end user gets to launch and configure the cluster.

6.5.2 Launching And Configuring The Nested Cluster As A User

Typically, the administrator is expected to have configured a global default configuration file already (section 6.3).

If the user would like to generate and modify one for themselves, then the steps in this section can be followed. The steps here are also useful for an administrator who is setting up and trying out a global default configuration file for users.

Generating And Modifying A Configuration File For `cm-cod-os`

A .ini configuration file can be generated for the user with

```
[fred@bright81 ~]$ cm-cod-os config dump --output-type ini > ~/cm-cod-os.ini
```

To launch the nested cluster, the settings that must be modified within this file are the following:

- `license_product_key=`<*the product key*>
- `cluster_password=`<*a password*>
- `floating_ip_network_uuid=`<*network UID*>

  If the Bright flat external net is used for floating IP addresses, then the network UID can be obtained from the OpenStack client:

  Example

  ```
  [fred@bright81 ~]$ openstack network show bright-external-flat-externalnet -f value -c id
  9a41d19c-7d04-441e-85e8-ee4f15c3cb7f
  ```

The following settings in the .ini file are assumed to have been defined, but can be modified:

- `node_boot_image=`<*image name*>

  By default, the boot image name is `iPXE-plain-eth0`.

- `head_node_type=`<*image name*>

  By default, the head node image name is `cod.medium`.

- `default_node_type=`<*image name*>

  By default, the regular node image name is `cod.xsmall`.

- `internal_mtu=`<*MTU size of the hosting internal network*>

  By default, the hosted network MTU size is set to a standard size of 1500. This allows hosted nodes to PXE boot because PXE booting does not accept MTU options in the DHCP server. For VXLAN-based network isolation this means that the hosting internal network must have a larger MTU value than 1500 to accomodate the hosted network MTU size.

  If the MTU value of the hosting internal network must remain at a value of 1500, then VLAN-based network isolation can be used instead.

The arguments to the preceding options take the form `--license-product-key`, `--cluster-password`, `--floating-ip-network-uuid`, and so on, as listed within the section on the `cm-cod-os cluster create` options help text (page 98).

Viewing And Picking Up An Image To Be Used For `cm-cod-os`

The image IDs that are available from the Bright Computing repositories, and their properties can be listed:

Example

```
[fred@bright81 ~]$ cm-cod-os image repo-list
+---------------------+----------+----------+-----------+----------+-------------+
| ImageID:Revision    | Head(GB) | Node(GB) | Distro    | CMD Rev. | BCM Version |
+---------------------+----------+----------+-----------+----------+-------------+
| centos7u5-8.1:9     | 3.72     | 1.4      | centos7u5 | 131439   | 8.1         |
| centos7u2-8.0:7     | 3.65     | 1.25     | centos7u2 | 127928   | 8.0         |
| centos7u2-7.3:14    | 3.38     | 1.04     | centos7u2 | 35931    | 7.3         |
+---------------------+----------+----------+-----------+----------+-------------+
```

A suitable image ID value can then be chosen. Each image ID has one head image and one regular node image associated with it. These images are then both installed:

Example

```
[fred@bright81 ~]$ cm-cod-os image install --is-public yes <ImageID:Revision>
```

The installation can take some time (minutes between each stage). A session run displays output similar to the following:

Example

```
[fred@bright81 ~]$ cm-cod-os image install centos7u5-8.1:9
+------------------+----------+----------+-----------+----------+-------------+
| ImageID:Revision | Head(GB) | Node(GB) | Distro    | CMD Rev. | BCM Version |
+------------------+----------+----------+-----------+----------+-------------+
| centos7u5-8.1:9  | 3.72     | 1.4      | centos7u5 | 131439   | 8.1         |
+------------------+----------+----------+-----------+----------+-------------+
About to install these images
```

```
Proceed? [yes/no] yes
INFO: Downloading bcmn-centos7u5-8.1-9 to /home/fred/bcmn-centos7u5-8.1-9.img.gz...
INFO: Creating manifest file /home/fred/bcmn-centos7u5-8.1-9.img.gz.manifest
INFO: Downloading https://s3-eu-west-1.amazonaws.com/cod-os-images.support.brightcomputing.com/
      bcmn-centos7u5-8.1-9.img.gz
INFO: Checking MD5 sum of /home/fred/bcmn-centos7u5-8.1-9.img.gz
INFO: Uploading /home/fred/bcmn-centos7u5-8.1-9.img.gz to glance...
INFO: Upload operation finished successfully.
INFO: Downloading bcmh-centos7u5-8.1-9 to /home/fred/bcmh-centos7u5-8.1-9.img.gz...
INFO: Creating manifest file /home/fred/bcmh-centos7u5-8.1-9.img.gz.manifest
INFO: Downloading https://s3-eu-west-1.amazonaws.com/cod-os-images.support.brightcomputing.com/
      bcmh-centos7u5-8.1-9.img.gz
INFO: Checking MD5 sum of /home/fred/bcmh-centos7u5-8.1-9.img.gz
INFO: Uploading /home/fred/bcmh-centos7u5-8.1-9.img.gz to glance...
INFO: Upload operation finished successfully.
[fred@bright81 ~]$
```

The image ID for the installed images installed by `cm-cod-os` locally can then be seen with:

Example

```
[fred@bright81 ~]$ cm-cod-os image list
+------------------+----------+----------+-----------+----------+-------------+
| ImageID:Revision | Head(GB) | Node(GB) | Distro    | CMD Rev. | BCM Version |
+------------------+----------+----------+-----------+----------+-------------+
| centos7u5-8.1:9  | 12.46    | 3.93     | centos7u5 | 131439   | 8.1         |
+------------------+----------+----------+-----------+----------+-------------+
```

A more extensive images list can be seen with:

Example

```
[fred@bright81 ~]$ openstack image list
+--------------------------------------+--------------------------------+--------+
| ID                                   | Name                           | Status |
+--------------------------------------+--------------------------------+--------+
| 43797ed5-0cdf-40f6-bbcd-1bdb11bfbc74 | Bright-COD-headnode-bootloader | active |
| 71568b33-1ef7-4099-aa10-f9690e52aaf1 | Bright-Managed-VM-iPXE-eth0    | active |
| 61856733-8f1f-4df7-a68b-d5ca27873f85 | Bright-Managed-VM-iPXE-eth1    | active |
| c34cbb10-3bfc-4d5f-b320-d446f7ded267 | bcmh-centos7u5-8.1-9           | active |
| ca519e84-5fac-4c03-8050-299400e822d2 | bcmn-centos7u5-8.1-9           | active |
| 953f9337-d5a1-4170-9edd-21d9149bbe4a | iPXE-plain-eth0                | active |
| 0d94b9a9-3d06-4a6a-b446-bc3cea7dfdd9 | iPXE-plain-eth1                | active |
+--------------------------------------+--------------------------------+--------+
[fred@bright81 ~]$
```

Using `cm-cod-os`

The user can now start creating clusters. A basic command using the .ini configuration file would be:

```
[fred@bright81 ~]$ cm-cod-os -c cm-cod-os.ini cluster create
11:15:09:     INFO: Please wait...
11:15:10:     INFO: --------------------------------------------------------
11:15:10:     INFO:         Cluster:  c-09-05-8.1-c7u5
11:15:10:     INFO: --------------------------------------------------------
11:15:10:     INFO:      Image name:  bcmh-centos7u5-8.1-9(centos7u5-8.1:9)
11:15:10:     INFO:      Node image:  bcmn-centos7u5-8.1-9(centos7u5-8.1:9)
11:15:10:     INFO:      Image date:  2018-08-17 10:48 (19d 0h ago)
```

```
11:15:10:     INFO:  Package groups:  none
11:15:10:     INFO:        CMD rev.:  131439
11:15:10:     INFO:         Version:  8.1
11:15:10:     INFO:          Distro:  centos7u5
11:15:10:     INFO:       Head node:  1 cod.medium: 4096 RAM, 2 vCPUs
11:15:10:     INFO:           Nodes:  5 cod.xsmall: 1024 RAM, 1 vCPUs
11:15:10:     INFO:                   Disk: 10 GiB (type: default)
11:15:10:     INFO: --------------------------------------------------------
Press ENTER to continue and create the cluster.
Press ctrl+c (or type 'a') to abort. Type 'i' for more info.
```

Ideally, if the administrator of the host cluster has set things up conveniently for the end user, the command needs no configuration file, and is run simply as `cm-cod-os cluster create`.

After a cluster has been created, it can be listed with:

```
[fred@bright81 ~]$ cm-cod-os cluster list
+----------------------+---------------+-------------+----------------------+----------+
| Cluster (stack) name | IP            | Head Status | Head Image           | CMD rev. |
+----------------------+---------------+-------------+----------------------+----------+
| c-09-05-b8.1-c7u5    | 192.168.200.8 | ACTIVE      | bcmh-centos7u5-8.1-9 | 131439   |
+----------------------+---------------+-------------+----------------------+----------+
```

Inactive clusters can additionally be listed by appending the `-a|--all` option to the list subcontext, as: `cm-cod-os cluster list -a`

Cluster removal is possible by specifying the cluster name to be deleted with the `delete|d|remove` option:

Example

```
[fred@bright81 ~]$ cm-cod-os cluster delete c-09-05-b8.1-c7u5
 a list of resources that are to be removed is shown
You are about to delete all of the above resources.
Proceed? [yes/no] yes
...
14:18:25:     INFO: Done. Stack 'c-09-05-b81-c7u5' deleted.
```

A less drastic way to conserve resources than the `delete` option, can be to use the `shelve` option. Shelving a cluster means that it is made inactive and stored. It can then be resumed with the `start` option.

The help options for the COD contexts can be seen by appending `--help` to them

Example

```
[fred@bright81 ~]$ cm-cod-os cluster --help
```

Example

```
cm-cod-os cluster create --help
```

7

Cluster Extension For OpenStack

7.1 Introduction

7.1.1 Cluster Extension In General

If Bright OpenStack is running on a cluster that a user can access and run OpenStack instances on, then the user can administrator of another Bright cluster (the client cluster) can run a cluster extension on the Bright OpenStack cluster (host cluster). That is, the client cluster can extend into the Bright OpenStack cluster and start up nodes that run within OpenStack there.

Users that run their own cluster may sometimes wish to extend their cluster into a second, Bright OpenStack, cluster. This second cluster can then be regarded as an OpenStack host, which hosts are service for the client cluster.

This kind of setup is a case of running a cluster within a cloud service.

Thus, for this case, the cloud service is provided by Bright OpenStack. Analogous setups where the cloud service is provided by AWS or Azure can also be managed with Bright Cluster Manager (Chapter 3 of the *Cloudbursting Manual*). In Bright jargon these cluster extension setups are conveniently abbreviated as:

- CX-OS, for cluster extension into OpenStack
- CX-AWS, for cluster extension into Amazon Web Services
- CX-Azure, for cluster extension into Microsoft Azure

This chapter explains cluster extension into OpenStack.

7.1.2 Overview Of Carrying Out Cluster Extension Into OpenStack

Bright provides a client, `cm-cluster-extension`, to set up and launch a CX-OS. The client can run on the head node of the Bright OpenStack cluster itself, or from a remote location. Bursting in Bright Cluster Manager version 8.1 is possible into Bright Cluster Manager versions 7.3, 8.0, and 8.1 running Bright OpenStack.

The `cm-cluster-extension` client is provided by default with Bright Cluster Manager as part of the `cm-setup` package. Its CLI options are covered in Chapter 4.1 of the *Cloudbursting Manual*.

Running it without CLI options brings up an Ncurses dialog, which provides an easier option to deploy a CX-OS (section 7.2).

Another option to deploy CX-OS is via a Bright View wizard (section 7.3).

7.2 Deploying A Cluster Extension Into OpenStack Via An Ncurses Dialog

The sequence of screens that follows shows the Ncurses-based procedure for CX-OS deployment. It has some parallels with the Ncurses-based procedure for CX-AWS deployment (page 45 of the *Cloudbursting Manual*.

Figure 7.1 shows the introductory window after running `cm-cluster-extension` as root:

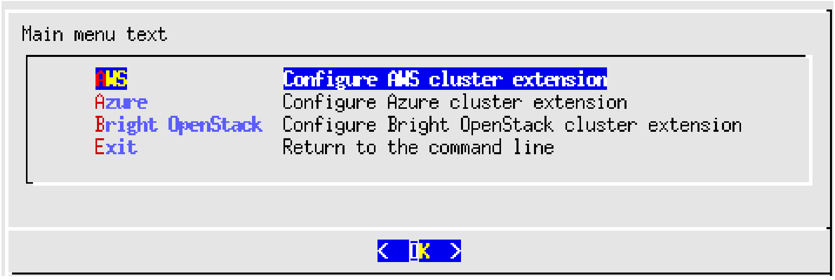

Figure 7.1: Ncurses Dialog

The type of cloud provider can be chosen. For this chapter, it means selecting Bright OpenStack as the cloud provider.

The provider instance can be added in figure 7.2. Previously added providers can also be removed in this screen (figure 7.2):

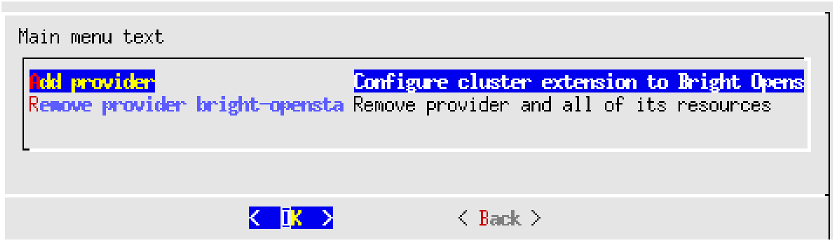

Figure 7.2: Ncurses Dialog: Add/Remove Provider

Credentials to the OpenStack instance can be configured in the screen in figure 7.3:

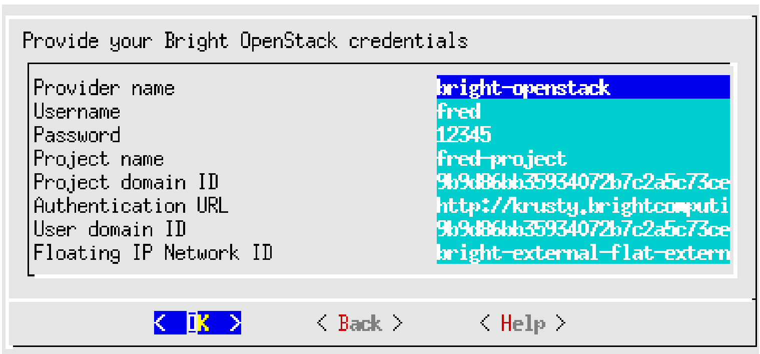

Figure 7.3: Ncurses Dialog: Add Bright OpenStack Credentials

A node-installer image source location should be set. This can be a published S3 repository. Another possibility is on the OpenStack host if images are already set up there by the tenant (figure 7.4), or if set up there by another tenant and made public:

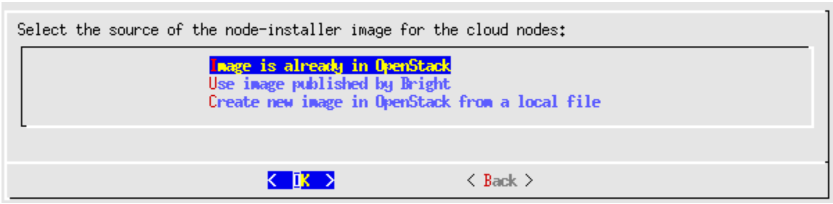

Figure 7.4: Ncurses Dialog: Select Node-installer Image Source

The node-installer image can be set (figure 7.5):

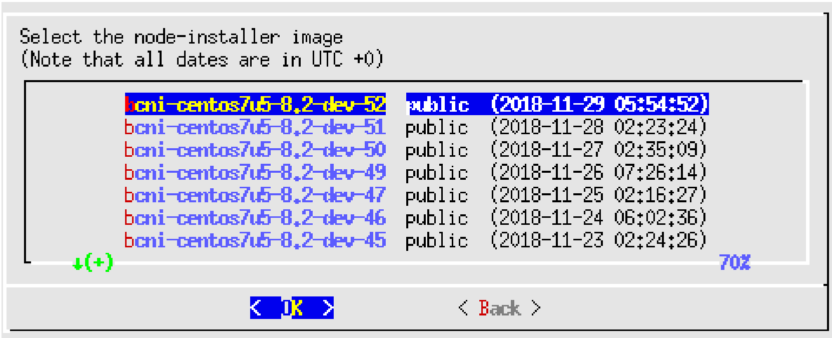

Figure 7.5: Ncurses Dialog: Select Node-installer Image

The number of cloud compute nodes available from the OpenStack host can be set (figure 7.6):

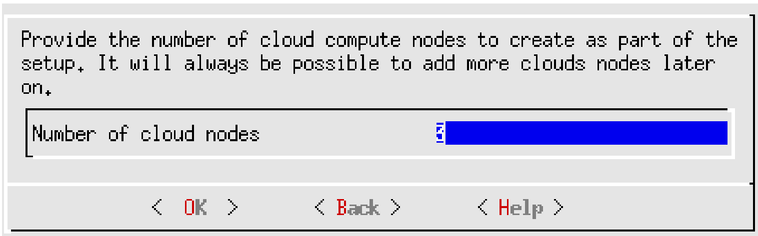

Figure 7.6: Ncurses Dialog: Setting The Number Of Cloud Compute Nodes

The cloud region (the host or hosts into which the extension takes place) can be set (figure 7.7):

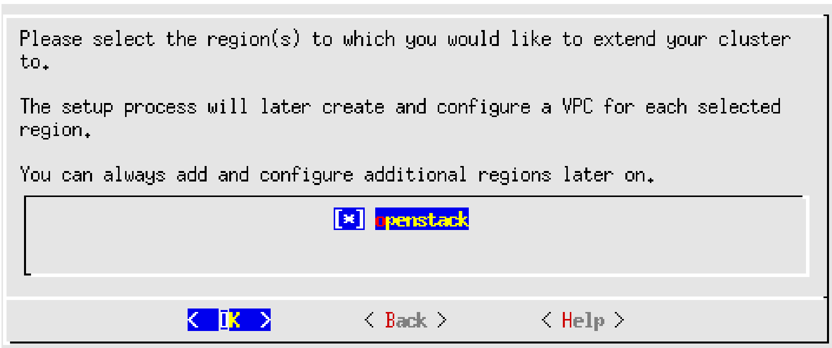

Figure 7.7: Ncurses Dialog: Cloud Region Selection

The default cloud director type can be set (figure 7.8):

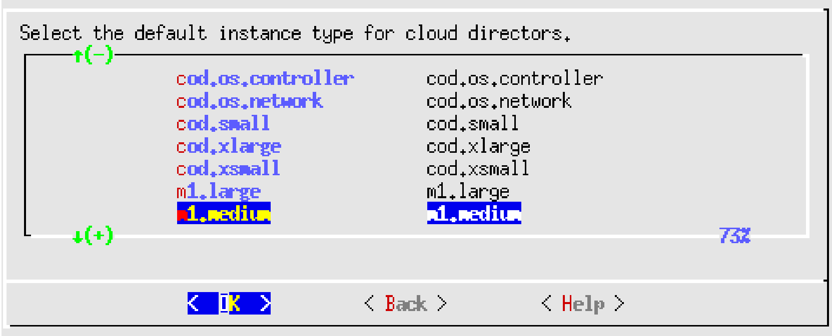

Figure 7.8: Ncurses Dialog: Default Cloud Director Instance Type

The default cloud node instance type can be set (figure 7.9):

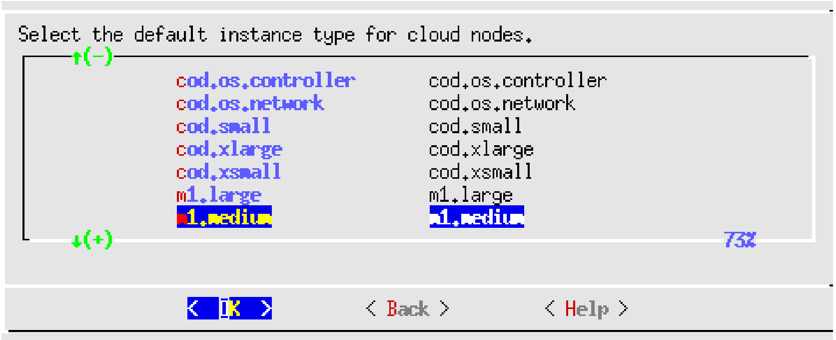

Figure 7.9: Ncurses Dialog: Default Cloud Node Instance Type

A summary of the configuration that has been done is displayed before committing (figure 7.10):

Figure 7.10: Ncurses Dialog: Summary

The configuration can be stored into a file (figure 7.11):

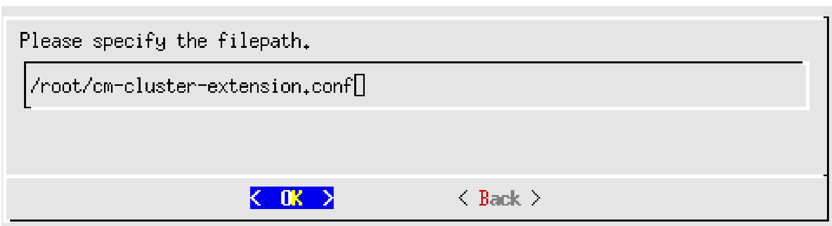

Figure 7.11: Ncurses Dialog: Setting Configuration File Location

The session then finishes with output showing the progress as the configuration is carried out.

Example

```
Executing 18 stages
################## Starting execution for 'Cluster Extension'
  - cloudstorage
  - clusterextension
## Progress: 0
#### stage: clusterextension: Fill Out Aliases In Config
## Progress: 5
#### stage: clusterextension: Check Subnet Configuration
Connecting to CMDaemon
## Progress: 11
#### stage: clusterextension: Create Provider
## Progress: 16
#### stage: clusterextension: Wait For Provider Data
## Progress: 33
#### stage: clusterextension: Create Netmap Network
NETMAP network 'netmap' already exists. Skipping
## Progress: 38
#### stage: clusterextension: Create Tunnel Networks
## Progress: 44
#### stage: clusterextension: Create Cloud Network
## Progress: 50
#### stage: clusterextension: Configure Headnodes
Configuring tunnel interfaces for headnode
Configuring roles for headnode
Adding CloudGatewayRole to bright81
## Progress: 55
#### stage: clusterextension: Configure Regular Nodes For VPC
## Progress: 61
#### stage: clusterextension: Enable Management On Tunnel Interface
## Progress: 66
#### stage: clusterextension: bright-openstack-cloud-director
## Progress: 72
#### stage: clusterextension: Create Extension
Heat Stack 4b5d2e21-8d65-412a-8a60-6eb457772d62 is being created...
## Progress: 77
#### stage: clusterextension: Create Security Groups
Creating security group 'bright81-bright-openstack-openstack-director-sg'
Creating security group 'bright81-bright-openstack-openstack-cnode-sg'
## Progress: 83
#### stage: clusterextension: Create Director
Adding FSExport for '/cm/shared' to hosts '10.42.0.0/16' to cloud director openstack-director
Adding FSExport for '/home' to hosts '10.42.0.0/16' to cloud director openstack-director
## Progress: 88
#### stage: clusterextension: Configure Node Categories
## Progress: 94
#### stage: clusterextension: Create Cloud Node
Processing node 'openstack-cnode001'
Processing node 'openstack-cnode002'
Processing node 'openstack-cnode003'
## Progress: 100

Took:     00:05 min.
```

```
Progress: 100/100
################### Finished execution for 'Cluster Extension', status: completed

Cluster Extension finished!
```

After the cloud director and associated cloud compute nodes are configured, they can be powered up (some output elided):

Example

```
[root@bright81 ~]# cmsh
[bright81]% device
[bright81->device]% list
Type         Hostname (key)      Category                         Ip              Status
------------ ------------------- -------------------------------- --------------- ----------
CloudNode    openstack-cnode001  openstack-cloud-node             172.16.0.1      [  DOWN  ]
CloudNode    openstack-cnode002  openstack-cloud-node             172.16.0.2      [  DOWN  ]
CloudNode    openstack-cnode003  openstack-cloud-node             172.16.0.3      [  DOWN  ]
CloudNode    openstack-director  bright-openstack-cloud-director  172.16.255.251  [  DOWN  ]
...
[bright81->device]% power on openstack-director
cloud .................... [ PENDING ] openstack-director (Powering on)
Thu Nov 29 18:04:55 2018 [notice] bright81: openstack-director [ PENDING ] (Instance has started)
...
Thu Nov 29 18:10:33 2018 [notice] bright81: openstack-director [   UP    ]
[bright81->device]% power on -n openstack-cnode00[1-3]
cloud .................... [ PENDING ] openstack-cnode001 (Powering on)
cloud .................... [ PENDING ] openstack-cnode002 (Powering on)
cloud .................... [ PENDING ] openstack-cnode003 (Powering on)
...
Fri Nov 30 12:55:37 2018 [notice] bright81: openstack-cnode002 [ PENDING ] (Instance has started)
Fri Nov 30 12:55:37 2018 [notice] bright81: openstack-cnode001 [ PENDING ] (Instance has started)
Fri Nov 30 12:55:37 2018 [notice] bright81: openstack-cnode003 [ PENDING ] (Instance has started)
...
Fri Nov 30 13:00:33 2018 [notice] bright81: openstack-cnode003 [   UP    ]
Fri Nov 30 13:00:34 2018 [notice] bright81: openstack-cnode002 [   UP    ]
Fri Nov 30 13:01:04 2018 [notice] bright81: openstack-cnode001 [   UP    ]
```

7.3 Deploying A Cluster Extension Into An OpenStack Instance Via A Bright View Wizard

Bright View can be used to deploy a cluster extension into OpenStack via a wizard that can be run from the clickpath:

`Cloud→OpenStack→OpenStack Wizard`

This then brings up the following display:

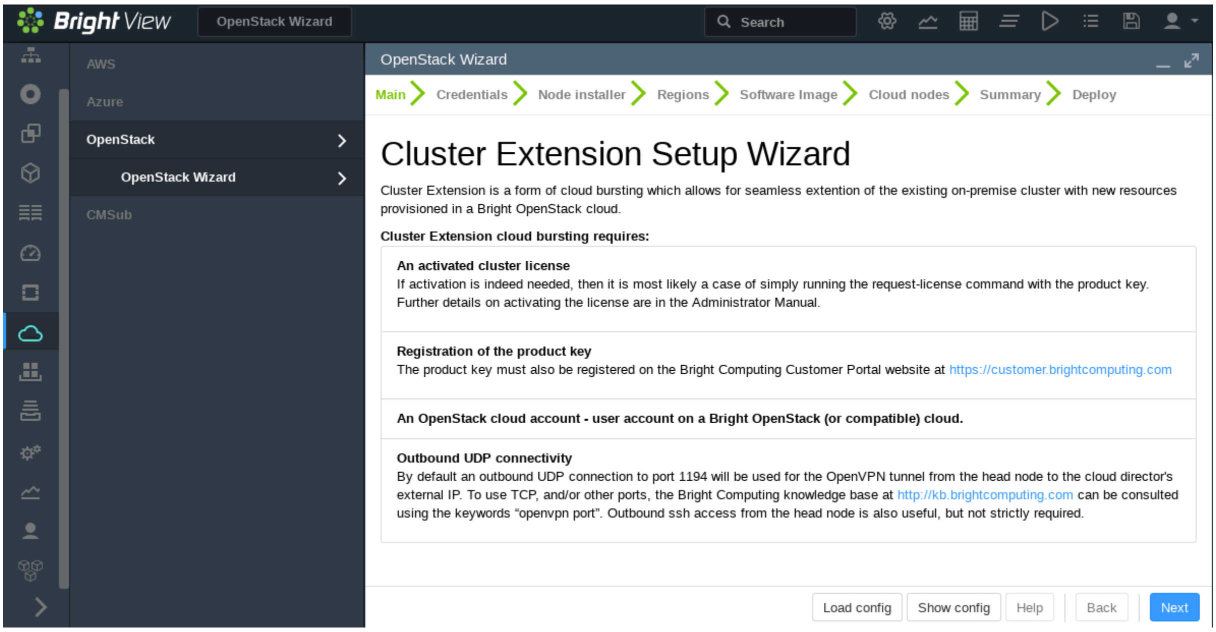

Figure 7.12: Bright View Wizard: Cluster Extension Into OpenStack Wizard Main Page

The procedure from here on is sufficiently similar to the Ncurses installation in section 7.2 that further elaboration here is not needed.

A

Storage Considerations For OpenStack

This chapter describes issues that should be considered when selecting a storage subsystem for OpenStack.

A.1 Introduction

OpenStack, to function properly, requires two types of storage:

- Some form of storage on the controller nodes (controller services), typically a Directly Attached Storage (DAS), for example an HDD or SSD.
  - Here, the Bright head node cannot be an OpenStack controller node
- Some form of Network Attached Storage (NAS) for Glance, Nova, and Cinder (for example, Ceph, NFS, GPFS, NetApp, and so on)
  - In a very special case, this storage for Glance and Nova can actually be directly attached, not on the Network.

NAS solutions for Glance, Nova, and Cinder are the focus of this chapter. DAS, and storage for other controller services, are covered here only very briefly.

Configuring QoS for OpenStack-managed block devices (for Nova and Cinder) is also discussed in this chapter.

A.2 DAS On The OpenStack Controller Nodes

- DAS is typically used for storage of RabbitMQ queue data and Galera MariaDB databases.
- It is convenient for storing the OpenStack configuration files for controller services, such as configuration files for Nova (`nova-api`, `nova-scheduler`, `nova-conductor`) (`/etc/nova/nova.conf`) and other openstack controller services or processes (`glance-api`, `glance-registry`, `cinder-api`, `cinder-volume`, and so on).
- This storage is present typically a locally attached HDD or SSD.
- As a special case, this storage need not be local, but can be a network mount to a persistent NAS. This adds latency for accessing, for example, the databases, but on the other hand it allows controller nodes to be run in a diskless manner if needed.
- Using SSDs is generally recommended for the controllers. For the databases it lowers the latency and boosts the performance of the cloud, compared with using traditional spinning disks.

A.3 NAS Storage For Glance/Nova/Cinder

OpenStack Glance, Nova, and Cinder services all require a persistent storage system. In most cases this should be a networked or clustered NAS system, such as Ceph, GPFS, or an NFS appliance.

A.3.0 Overview: Native OpenStack Access Versus Non-native OpenStack Access

There are two ways of accessing external NAS storage in OpenStack:

1. Relying on a built-in driver for a given OpenStack component. For example, an "NFS driver for Cinder" or a "Ceph driver for Glance".
    - Supported types of shared storage: Bright OpenStack supports most of types of Storage systems supported by upstream OpenStack. Many can be configured during OpenStack deployment. Those that cannot, can be configured manually after deploying Bright OpenStack, for example: by using Configuration Overlays, which make it easy to apply configuration changes to OpenStack services.

      For such post-installation manual storage configuration cases, the option to configure it later should be selected during the deployment of OpenStack, at the point when the deployment asks for a storage system for Glance (figures 3.23 and 3.6), Cinder (figures 3.24 and 3.7), or Nova (figures 3.28 and 3.8).
    - Cinder has the widest variety of native drivers (70+) as shown in the support matrix at `https://wiki.openstack.org/wiki/CinderSupportMatrix`. Compared with Cinder, the Nova and Glance services offer relatively fewer drivers as of OpenStack Newton.
        - Glance, however, can be made to use Cinder as the back end.
        - Nova's ephemeral disks can either not be used at all with a given OpenStack deployment, or can use a local DAS for storing data. In the case of using a local DAS for storage, a dedicated driver is not needed.
2. By mounting the NAS locally on the nodes which require access to it, and configuring a given OpenStack component to use that "local" mount for storing data. This is not possible with all NASes. For example, Nova's "localdisk" driver requires file-locking functionality on the filesystem used for storing ephemeral disks.

Some clusters use a combination of the two access types at the same time within the same cloud. For example, Cinder can access a NAS via a dedicated driver, whereas Nova and Glance can be configured to access (the same, or different) NAS via local file system mounts.

Network Considerations

Network equipment and bandwidth should be considered when using a Network/Clustered storage solution for any of the following components.

Using a high latency or low bandwidth network slows down the cloud, and could lead to a disturbance in function.

A.3.1 Glance

Glance uses shared storage to store images of the VMs. By storing images on a shared, network-attached, storage, accessible by all OpenStack Glance services, Glance can run in active/active HA mode. In the simplest case this can be a local NFS mount. In a more advanced case Glance can store images directly in Ceph.

Even though in theory Glance can make use of a locally attached storage (a DAS), it is recommended to use some sort of a NAS such as Ceph instead for it, especially when OpenStack is deployed with multiple controller nodes.

Glance can also be configured with an additional auxiliary HTTP store. That is, the images can be added to Glance as a reference to by an already-existing HTTP server somewhere on the network. The images are then downloaded from the web server via HTTP whenever needed.

Glance can also be configured to store images in Cinder, as volumes. In this case it will rely on whatever storage is configured for Cinder.

A.3.2 Nova

Nova requires persistent storage for storing disks of VMs, that is "ephemeral" block devices. Those block devices are bound to a VM, and their size is determined by the flavor of the VM. They are always removed when the VM is removed, which is why they are ephemeral.

There are 3 types of ephemeral disks defined (or not defined) by a Nova VM Flavor. The storage can be composed of any combination of these:

- Root disk (also called the root ephemeral disk)
  - Provisioned with the VM image (unless a Cinder volume is used for that instead)
- Ephemeral disk (used for scratch space)
- Swap disk (Linux swap partition)
- As a special case, a flavor can define all three of those to be 0 GB, in which case a Cinder Volume can be used as the root volume hosting the VM image.

By using a shared storage for Nova, VM migration is carried out more easily. This is because the ephemeral block devices associated with the VM do not have to be copied from one hypervisor to another during the migration process. It also allows the hypervisor to be run in a purely diskless configuration.

Using a locally-attached DAS storage for Nova is also possible, and in some cases desirable. By having Nova use local DAS for storing the ephemeral block devices, the access latency and throughput to this device can in some cases be much better than a comparable NAS, simply because no network access is involved.

Whether DAS or a NAS should be used for Nova is entirely up to the architect of the OpenStack cloud. Both approaches have different advantages and disadvantages.

Nova with DAS

Nova with DAS typically has:

- Lower latency
- Low resilience, unless RAID is used. But even with RAID, if a hypervisor goes down then data in ephemeral storage is lost.
- A different I/O profile compared with a Cinder volume

Nova with NAS

Nova with NAS typically has:

- Typically higher latency than local storage
- High resilience of data in the case of hypervisor failure
- Easier VM migration

One of the possible architectural designs is to use Cinder Volumes—which are always accessed via the network—for the Root filesystem of the VM, and then use the "ephemeral" (scratch) disk in Nova. This then provides a low-latency (attached-to-hypervisor) scratch space.

Another architecture could be to not use Nova's ephemeral storage at all, and instead rely entirely on Cinder volumes for root filesystem disks, as well as for any auxiliary storage.

Yet another architecture would be to not use Cinder, and rely entirely on Nova disks for all the block storage, via DAS or NAS.

Using a mixture of the above is also possible. Thus, some VMs could only use Nova, some only Cinder, some a combination of the two.

A.3.3 Cinder

Cinder requires persistent storage for storage volumes, that is: auxiliary block devices. Cinder volumes function like the “disks” managed by Nova (as in, they are block devices), but unlike them, their life cycle does not have to be bound to a specific VM. That is, a Cinder volume can be created, attached to any VM, used, detached, and then attached to another VM. Cinder volumes can be made to be automatically removed by OpenStack whenever the VM that they are attached to is removed.

Cinder is deployed by default by Bright OpenStack. However, in theory it does not need to be used or configured as part of Bright OpenStack. It is entirely possible to run a fully functional OpenStack cloud without an operational Cinder. Given that both Nova and Glance can operate without a shared storage, i.e. using only locally attached storage, then in theory it is possible to have a fully functional OpenStack cloud without any sort of persistent NAS. However it is typically not recommended to do so.

Cinder can expose different types of volumes to the end users. Typically, if more than one type of volume is used, then all of those types are backed by the same NAS system, but with different QoS characteristics. It is however possible to configure Cinder in such a way that each volume type would be mapped to a completely different NAS system. For example, there could be a “Fast NFS volume”, a “Slow NFS Volume”, and several “Ceph Volumes”, (these could be mapped to different Ceph storage pools, with different replication/performance characteristics), and several types of say, “NetApp volumes”. All of these volumes could be in a single OpenStack deployment, and visible to all, or only selected projects.

A.3.4 Considerations For NAS For OpenStack

Shared storage that performs well is a critical part of an OpenStack cloud that performs well. No matter which shared storage system is used for OpenStack, it makes sense to spend some time making sure that it suits the OpenStack cloud well.

To give hardware suggestions or software configuration for shared storage is not easy. These suggestions always depend on the types of NAS used, as well as on the expected performance characteristics of the OpenStack cloud. For example, how many VMs will use the cloud and how often, and how much and how fast the storage should be.

Several generic rules-of-thumb are outlined next. These can be applied to most types of shared storage for OpenStack. There are also several other guidelines which are specific to Ceph. Even if Ceph is not going to be run, it is wise to read through the Ceph guidelines too, as some of the suggestions are appropriate for some other types of NAS systems.

General Rules-of-thumb

- Using the same NAS for Glance, Cinder, and Nova makes things easier from an operational perspective.
  - Unless configured otherwise, these 3 services can be configured to dynamically consume the amount of storage required by them at any given time from within the same pool.
  - In some cases this allows copy-on-write (COW[1]) semantic filesystems to be used to thinly and quickly provision new block devices from an image. So, when creating a new VM from

[1] A COW design for a filesystem follows the principle that, when blocks of old data are to be modified, then the new data blocks are written in a new location (the COW action), leaving the old, now superseded, copy of the data blocks still in place. Metadata is written to keep track of the event so that, for example, the new data blocks can be used seamlessly with the contiguous old data blocks that have not been superseded. This is in contrast to the simple overwriting of old data that a non-COW filesystem such as Ext3fs carries out.

a Glance image, the image does not have to be copied to the block storage system, but instead a copy-on-write copy can be made instantaneously. Some of the NASes which support this include Ceph and GPFS. NFS does not support copy-on-write, and can therefore be a suboptimal choice for many OpenStack use cases.

- Unless storing a lot of VM images in Glance is planned, Glance typically consumes the smallest portion of raw storage, when compared to Nova ephemeral disks, or Cinder Volumes
- By default all Bright clusters come with an NFS export in the form of /cm/shared. When deploying Bright OpenStack, `/cm/shared` can be selected to be used with Glance, or Cinder, or Nova. Selecting any of these means that NFS is to be used as the NAS system for OpenStack.

  If the solution is to be used only for a small OpenStack cluster, and typically just for a proof-of-concept system, then hosting the NFS export on the head node, as is the default, is acceptable. For production use, the NFS export should be backed by a dedicated high-performance NAS appliance.
- If using NFS for Cinder, it should be borne in mind that Volume Snapshots are not available. This is because the default Cinder NFS driver does not support them.

Ceph Rules-of-thumb

It should be emphasized that the text in this section is merely a rough guide. Designing a high-performance Ceph cluster is far from trivial, and following these guidelines cannot give the best performance for every single use case.

These guidelines are, however, a good place to start when starting from scratch, and should typically result in a much better performance than just running Ceph on a random collection of storage nodes.

- With Ceph, if Glance, Nova, or Cinder share the same Ceph cluster—even if they are configured in different storage pools—then copy-on-write works just fine between Glance and Nova, and between Glance and Cinder. Cinder can also do copy-on-write between volumes and snapshots.
- When Ceph is used, then Bright OpenStack by default configures Glance, or Nova, or Cinder, with each service having its own storage pool
  - By default, Bright configures the pools with a replication factor of 3. That is, each object is stored on 3 separate Ceph OSD nodes.
  - It is possible to reduce that to 2 in order to save raw space. This also increases the overall performance of Ceph because before a write operation to Ceph returns back to the client, the object first needs to be stored in all the replicas.
- Ceph OSDs, which are the nodes that store the raw data, can be deployed as standalone nodes. Alternatively, they can be converged with the hypervisor nodes, in which case the hypervisor node will be running the 2 VMs, as well as being responsible for managing Ceph storage
  - This does not mean that VMs will be accessing the Ceph storage locally. As a very simple example case of 10 identical converged hypervisor/OSD nodes, there is only a 1 in 10 chance that the block of data accessed by a VM will happen to be hosted by the local OSD. The remaining 9 will be accessed via the network.
  - Combining Ceph OSDs with hypervisors means that VM processes may affect the performance of Ceph OSD processes, unless both are configured to be pinned only to specific sets of CPU cores.
  - For the best possible performance, OSDs should not be converged on Hypervisors. For the best possible value, convergence should be considered.

- For the best possible performance Ceph data should be stored directly on the SSDs. For the best possible performance:value ratio at the time of writing (July 2017), the data should be stored on HDDs (ideally using a high-spinning SAS), with an SSD journal in front.
- The sustained write speed of the SSD should be roughly equivalent to the sum of the sustained write speeds of the HDDs which are configured behind that SSD. For example: 1 SSD with 400 MB/s writes configured for 4 HDDs with 100 MB/s each.
- Using multiple SSDs for journaling in one Ceph OSD is also possible.
- Ceph OSD can run multiple Ceph OSD service (one service per backing disk)
  - 0.5-1 CPU cores should be assumed to be required to be available and dedicated for each OSD service
  - 500-1000 MB of RAM should be assumed to be required for each OSD service for each 1 TB of data managed by that service.
- There should be at least 3 Ceph monitor nodes. In most cases that will be enough. In OpenStack environments, Ceph monitor nodes are typically converged with OpenStack controller nodes, of which also typically 3 are used.
- The total sum of sustained write speeds for Ceph journaling SSDs should not exceed the network bandwidth available for delivering the data to the SSD. For example: 1 single SSD with 400 MB/s (or 3200 Mbps), already exceeds the typical 1 Gbps link. This means that even an average SSD will not be fully utilized in a 1 Gbps networking environment. Using 10 Gbps links for Ceph is therefore recommended.
- Networking
  - A dedicated, 10 Gbps, network the "Ceph cluster network", is strongly advised. This is the network that Ceph uses to replicate data between the OSDs.
  - In most cases, the Ceph public network—the network used by Ceph clients to access Ceph—should also be at least 10 Gbps. 1 Gbps should only be used if each Ceph OSD does not have more than 2-3 HDDs for storing raw data, and also has no SSD journals. With more disks than that, the disk throughput will be underutilized when writing to Ceph over a 1 Gbps public network.
  - To get the best performance, 3 network fabrics should be considered:
    * On hypervisor nodes:
      · 10+ Gbps for inter-VM communication (VLAN/VXLAN)
      · 10+ Gbps for accessing Ceph (Ceph public network)
    * On Ceph OSD nodes:
      · 10+ Gbps for Ceph public network
      · 10+ Gbps for Ceph private network (replication)
    * It is, however, possible to use only a single, or only two, physical network fabrics for all three of those logical networks. Performance, however, might suffer.
      · E.g. 1 fabric for VLANS/VXLANs (1Gbps or 10 Gbps)
      · 1 fabric for both the Ceph public network, and the Ceph private network (ideally 10Gbps or more).
- When selecting Ceph OSDs it is possible to use either smaller, "thin nodes", or bigger, "thick nodes". An example of a thin node is 1 socket machine, with 7-12 disks. An example of a thick node is a multi-socket machine with more than 12 disks.

  - Thick nodes are often more compelling in terms of costs and rack space
  - But thick nodes are harder to configure properly. For example, the limited throughput of the inter-socket QPI bus has to be kept in mind. There are also many other places where bottlenecks can occur in such a dense environment, and these can be hard to troubleshoot.
  - Thin nodes are typically easier to configure and set up for maximum performance.

- All other things, such as total raw storage, being equal, a typical Ceph cluster made up of thin nodes will end up having more nodes than a Ceph cluster of the same capacity made up of thick nodes.
  - More nodes in a Ceph cluster mean less impact on Ceph's overall performance if one of the nodes fails. That is, a smaller percentage of the entire cluster goes down. This is because if a thick node fails, then all the copies of data stored on the thick node need to be replicated from the remaining Ceph nodes across the remainder of the Ceph cluster.
  - Thin Ceph OSDs nodes therefore typically trade a higher upfront cost for a lower long-term maintenance effort and a lower complexity.

Questions To Answer When Selecting Hardware For Ceph

- Probably the most important: how fast is the network? 1 Gbps, 10 Gbps?
  - How many VMs will be running? Under peak VM disk I/O loads how much bandwidth needs to be available for each VM?
- Will Ceph run on SSD only or HDD only? Or on HDD with a journaling SSD? The number of either SSDs, or HDDs, or Journaling SSDs, should be selected such that their cumulative sustained write speed is about the same as the inbound network bandwidth available for delivering data to Ceph. This depends on the network speed, and on the disk capacity in the hardware enclosures used for Ceph OSDs.
- An example to illustrate the balancing out of data and network flow: one possible scenario in a 10 Gbps network environment is to have 3 x 400 MB/s in journaling SSDs (9.6 Gbps sustained write), with 4 x 100 MB/s HDDs behind each one of those SSDs.
- Balancing out the flow helps ensure that both the network, as well as the disks, are fully utilized. In a hyperconverged architecture (combining OSDs with VMs), which also uses the same networking fabric for the Ceph Public network, as well as for the VLAN/VXLAN network for inter-VM communication), oversaturating the network with Ceph traffic is best avoided, so that some network throughput for inter-VM communication is possible.

Using GPFS for OpenStack

- **Glance** does not have a native GPFS driver. However, GPFS for Glance can be used by mounting it locally on a controller node, and using a localdisk driver for Glance
- **Cinder** supports GPFS via a native driver. It can be made aware that Glance is also using GPFS, and can be made to use copy-on-write, from Glance to its volumes
- **Nova** does not support GPFS natively for ephemeral disks, but it can still be used with a localdisk driver, pointed to a local GPFS mount, similarly to Glance. Copy-on-write from Glance is not possible. But since Nova uses Cinder, Nova can issue copy-on-write requests from Glance, for creating Cinder volumes.

Using Lustre FS with OpenStack

- **Nova**: A Lustre filesystem can be used as a back end for ephemeral disks only when it is mounted to allow exclusive file locking with the flock option. Another requirement is to disable Lustre striping on the LFS folder within which the ephemeral disks will be mounted. This is to avoid Lustre being slow when transferring small files to/from the VM.

```
# cmsh
% configurationoverlay
% openstackhypervisors
% roles
% use openstack::compute
% fsmounts
% show /var/lib/nova/instances
Parameter                          Value
----------------------------------------------------------------------------------------------
Device                             10.149.0.254@o2ib0:/lustre/cm/shared/apps/openstack/nova
Dump                               no
Filesystem                         lustre
Filesystem Check                   NONE
Mount options                      defaults,_netdev,flock
Mountpoint                         /var/lib/nova/instances
RDMA                               no
```

- **Glance**: It is possible to configure Lustre for Glance in a similar way to that Nova. The only difference being that the "flock" option is not needed.

- **Cinder**: Current status for OpenStack Newton needs to be investigated.

A.3.5 Throttling IOPS

In order to prevent out-of-control VMs from consuming too large a portion of the overall bandwidth to the storage system (be it DAS or NAS), and thus starving out the other VMs, it is recommended to configure a QoS for disk I/O.

This can be done on the OpenStack layer via Nova (for the ephemeral disks), or via Cinder (for volumes). But, depending on the NAS used, it can also be done in the NAS system itself, in which case doing so is specific to the given NAS system.

When configuring QoS via OpenStack the read/write bandwidth, as well as read/write IOPS, can be regulated.

Configuring QoS For Nova

This is based on the description at `https://wiki.openstack.org/wiki/InstanceResourceQuota` at the time of writing (July 2017):

To configure QoS for Nova, it can be configured on a per-flavor basis. The QoS will be applied to ephemeral disks of the VMs. Nova allows certain VM types only to be exposed to specific Projects (tenants/users). This allows higher-performance flavors to be accessed only by specified OpenStack users.

Configuring QoS For Cinder

QoS for Cinder can be configured with Cinder's "QoS" functionality. In Cinder, the QoS values are bound to a volume types. It is therefore possible to create different Volumes Types with different QoS performance characteristics. And, for example, to have multiple volumes, each with different QoS values, attached to a single VM.

Cinder Volume Types can be made hidden (not-public), and then shared with specific projects within OpenStack (i.e. with users which need to have access to to them). This can done using the `cinder` CLI.

Not all storage drivers for Nova/Cinder support QoS. The driver documentation for the driver being considered for use should be consulted for details. Ceph (RBD) drivers for Nova and Cinder, for example, have a very good QoS support.