OpenStack IN ACTION

V. K. Cody Bumgardner

MANNING

OpenStack in Action

by V. K. Cody Bumgardner

Chapters 3, 9, and 11
ISBN 9781617292163

UNEDITED PROOF

To pre-order or learn more about this book go to www.manning.com/bumgardner/

brief contents

3 *Learning basic OpenStack operations*

This chapter covers

- Using the OpenStack CLI to manipulate your deployment
- Understanding the OpenStack tenant model by building a new tenant
- Understanding and using basic tenant networking as it relates to intra-tenant configuration
- Understanding and using OpenStack Networking as it relates to internal and external network configuration
- Using tenant quotas to control resource allocation

This chapter builds on the deployment from chapter XREF ch02 by demonstrating basic operations you'll encounter as an OpenStack administrator or user. Chapter XREF ch02 was more focused on end-user interaction with OpenStack, so those examples were based on the Dashboard, which is easy to use and can be used to perform many user and administrative functions. This chapter is focused on operational exercises, so examples are based on the OpenStack command-line interface (CLI).

If you have systems administration experience, you'll certainly appreciate the ability to script a repetitive function, such as creating a thousand users. The OpenStack APIs can also be used for these tasks, and they'll be briefly introduced. As you'll discover, if you can perform an operation with the CLI, you can easily perform the same operation with an API directly. For the examples in this chapter we'll stick with using the CLI, but this chapter is constructed so that you can walk through the examples using either the API or Dashboard once you understand the concepts demonstrated through the CLI.

The CLI also has the added benefit of using separate applications for each OpenStack component. While at first this might seem like a bad thing, it will help

you better understand which component is responsible for what.

The basic OpenStack operations covered in this chapter can be applied to a DevStack deployment, like the one in chapter XREF ch02, or to a very large multiserver production deployment. In chapter XREF ch02 you used the `Demo` tenant (project) and the `demo` user. This tenant, user, and other objects were created by DevStack, but tenants, users, networks, and other objects won't be created for you automatically in manual deployment. In this chapter you'll walk through the process of creating the necessary objects to take a test-drive in a tenant you create. By the end of the chapter you'll know how to separate resource assignments using the OpenStack tenant model..

The chapter starts by introducing the OpenStack CLI. Then you'll walk through the process of creating a tenant, user, and networks. Finally you'll learn about quota management from the tenant perspective. As you walk through the examples, take note of the CLI applications used in each step. As you go through this chapter, you'll not only learn basic OpenStack operations, you should get a better understanding of which OpenStack components provide what functions. In chapter XREF ch04 we'll cover OpenStack component relations in more detail.

3.1 Using the OpenStack CLI

Let's take a brief look at how you can interact with OpenStack on the command line (CLI). Before you can run CLI commands, you must first set the appropriate environment variables in your shell. Environment variables tell the CLI how and where to identify you. You can provide input for these variables directly to the CLI, but for the sake of clarity, all examples will be shown with the appropriate environment variables in place.

To set these variables, run the commands shown in listing 3.1 in your shell. Each time you log in to a session you'll have to set your environment variables.

Listing 3.1 Set environmental variables

```
source /opt/stack/python-novaclient/tools/nova.bash_completion   ❶
source openrc demo demo   ❷
```

❶ Set variables for shell completion so that pressing tab after entering "something /bo" completes "something /boot".

❷ Run this command from the ~/devstack directory. When you run OpenStack CLI commands, your identity will be <user: demo> in <tenant:demo>.

SIDEBAR **Setting environment variables manually**

If you're an experienced user or you're not using DevStack, you can manually set your environmental variables by running the following commands and substituting your values for the ones in these examples:

```
export OS_USERNAME=admin
export OS_PASSWORD=devstack
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://10.163.200.32:5000/v2.0
```

These example commands will set the current shell user as the OpenStack `admin` user of the `admin` tenant.

To make sure your variables have been properly set, you should make sure you can run an OpenStack CLI command. In your shell, run the `nova image-list` command, as shown in listing 3.2. This CLI command reads the environment variables you just set and will use them as identification. If you're properly identified and have rights to do so, the CLI will query OpenStack Compute (Nova) for your currently available `image-list`.

Listing 3.2 Setting variables and executing a first CLI command

```
devstack@devstack:~/devstack$ source \
  /opt/stack/python-novaclient/tools/nova.bash_completion
devstack@devstack:~/devstack$ source openrc demo demo
devstack@devstack:~/devstack$ nova image-list                    (1)
+---+----------------------------------+--------+--------+
| ID| Name                             | Status | Server |
+---+----------------------------------+--------+--------+
| 4.| Ubuntu 12.04                     | ACTIVE |        |
| f.| cirros-0.3.1-x86_64-uec          | ACTIVE |        |
| a.| cirros-0.3.1-x86_64-uec-kernel   | ACTIVE |        |
| a.| cirros-0.3.1-x86_64-uec-ramdisk  | ACTIVE |        |
+---+----------------------------------+--------+--------+
```

(1) **This simple command will list your nova images**

You should now be able to run OpenStack CLI commands as the `demo` user in the `demo` tenant. This is the same user you used in chapter XREF ch02, so any changes you make using the CLI will be reflected in the Dashboard.

Using the command in listing 3.3, you can create a new OpenStack instance, just like you did in the Dashboard. As mentioned in chapter XREF ch02, an OpenStack instance is a VM for the purposes of this book.

Listing 3.3 Launching an instance from the CLI

```
nova boot \   ❶
 --flavor <flavor_id> \   ❷
 --image <image_id> \   ❸
 <instance name>   ❹
```

❶ Tell Nova that you want to boot/create an instance
❷ Specify the <flavor_id> (size) as shown by the command nova flavor-list
❸ Specify the <image_id> as shown by the command nova image-list
❹ Specify the name of your instance

When you run this command, you'll get results something like the following.

```
nova boot \
--flavor 3 \
--image 48ab76e9-c3f2-4963-8e9b-6b22a0e9c0cf \
Test_Instance_3
+---+----------------------------------+--------+--------+

+--------------------------------------+----------------+
| Property                             | Value          |
+--------------------------------------+----------------+
| OS-DCF:diskConfig                    | MANUAL         |
| OS-EXT-AZ:availability_zone          | nova           |
| OS-EXT-STS:power_state               | 0              |
| OS-EXT-STS:task_state                | scheduling     |
| OS-EXT-STS:vm_state                  | building       |
| OS-SRV-USG:launched_at               | -              |
| OS-SRV-USG:terminated_at             | -              |
| accessIPv4                           |                |
| accessIPv6                           |                |
...
...
+--------------------------------------+----------------+
```

You can do everything with the OpenStack CLI that you can using the Dashboard, and more. In the preceding example you performed the Nova boot command, which provisioned a new VM. To get help with more advanced Nova commands, use the following command: `nova help COMMAND` (replacing `COMMAND` with the command you're interested in). There are similar command-line utilities for Keystone, Glance, Neutron, and so on.

You now have a basic idea of how the OpenStack CLI works. In later chapters you'll mostly be working with the CLI, so learning how things should work in DevStack should will be helpful if things don't work as expected in later chapters.

Before we move on to the tenant examples, let's take a look the mechanics of the OpenStack APIs.

3.2 Using the OpenStack APIs

At this point you might be wondering, "how does the OpenStack CLI work?" The answer to this is that the CLI applications call APIs specific to OpenStack components. The component-specific APIs interface with a number of sources, including other APIs and relational databases. This also holds true for the Dashboard, which you used in chapter XREF ch02. All OpenStack interactions eventually lead back to the OpenStack API layer.

It could certainly be argued that the inherent vendor neutrality provided by the OpenStack API is OpenStack's greatest benefit. An entire book could be devoted to working with the OpenStack APIs. People who integrate external systems or debug OpenStack code will find themselves looking at the API layer. The thing to keep in mind is that all roads lead to the OpenStack APIs. If you have further interest in the OpenStack APIs see the "Debug CLI /Expose API" sidebar.

To get started using the OpenStack API directly, you can follow the example in listing 3.4. This command will query the OpenStack API for information, which will be returned in JavaScript Object Notation (JSON) format. Python will be used to parse the JSON so it can be read on your screen.

Listing 3.4 Executing a first API command

```
curl -s -X POST http://10.163.200.32:5000/v2.0/tokens \
 -d '{"auth": {"passwordCredentials": \
 {"username":"demo", "password":"devstack"}, \
 "tenantName":"demo"}}' -H "Content-type: application/json" | \
 python -m json.tool
```

SIDEBAR **Debug CLI / Expose API**

Every CLI command will output its API command if the debug flag is set. To enable debugging for a specific CLI command, pass the `--debug` argument before any other variables as shown here:

```
devstack@devstack:~$ nova --debug image-list

REQ: curl -i 'http://10.163.200.32:5000/v2.0/tokens'
-X POST -H "Content-Type: application/json"
-H "Accept: application/json"
-H "User-Agent: python-novaclient"
-d '{"auth": {"tenantName": "admin", "passwordCredentials":
{"username": "admin", "password": "devstack"}}}'

...
```

Now that you understand the mechanics of the OpenStack CLI and API, you're ready to put these skills to use. In the next section we'll walk through creating a new tenant (project) using the CLI. This is an operational function you'll perform for each new department, user, or project you want to separate from a more general tenant.

3.3 Tenant model operations

OpenStack is natively multi-tenant-aware. As previously mentioned in chapter XREF ch02, you can think of your OpenStack deployment as a hotel. A person can't be a resident of a hotel unless they have a room, so you can think of *tenants* as hotel rooms. Instead of beds and a TV, Hotel OpenStack provides computational resources. Just as a hotel room is configurable (single or double beds, a suite or a room, and so on), so are tenants. The number of resources (vCPU, RAM, storage, and the like), images (tenant-specific software images), and the configuration of the network are all based on tenant-specific configurations. Users are independent from tenants, but users may hold roles for specific tenants. A single user might hold the role of administrator in multiple tenants. Every time a new user is added to OpenStack, they must be assigned a tenant. Every time a new instance (VM) is created, it must be created in a tenant. Management of all OpenStack resources is based on the management of tenant resources.

Because your access to OpenStack resources is based on tenant configurations, you must understand how to create new tenants, users, roles, and quotas. In chapter

XREF ch02 you used DevStack, which created a few sample tenants and users for you. In the next few subsections you'll walk through creating a new tenant and all related objects that go with it from scratch. As an OpenStack administrator, this will be a common task. A department or even a project might be a new tenant. Tenants will be the fundamental way that you divide and manage configurations and resources in OpenStack.

3.3.1 The tenant model

Before you start the process of creating tenant and user identity objects, you need to get an idea of how these items are related. Take a look at figure 3.1, which shows the interplay among tenants, members, and roles in OpenStack. You can see that *roles* are designations independent of tenants until a user is assigned a role in a tenant. You can see that *User* is an `admin` in the `General` tenant and a `Member` in the `Another` tenant. Notice that both users and roles have a one-to-many relationships with tenants. As you move forward in this chapter's examples, you'll create several of the components shown in this figure.

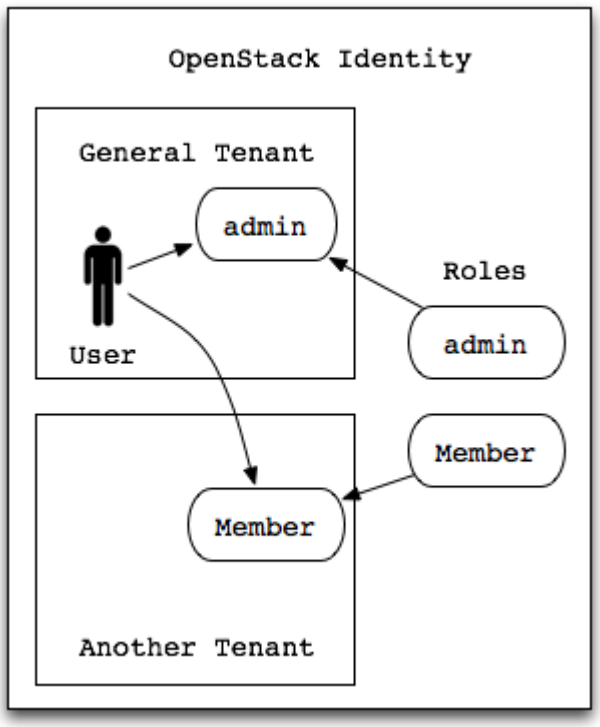


Figure 3.1 Tenant identity relations

As you can see in figure 3.1, tenants are the way OpenStack organizes role assignments. In OpenStack all resource configuration (users with roles, instances, networks, and so on) are organized based on tenant separation. In OpenStack

jargon, the term *tenant* can be used synonymously with *project*, so think of using a tenant for a particular project or organizational division. It's worth noting that roles are defined outside of tenants, but users are created with an initial tenant assignment. It would be reasonable for a user to be created in departmental tenant (for example, John Doe is created in the `General` tenant) and be assigned a role in another tenant (John Doe is `Member` of `Another` tenant). This means that every tenant can have a specific user with the role `Member`, but that specific user would only have one home tenant.

We'll use the OpenStack CLI to demonstrate examples in this section. We could just as easily use the Dashboard, but demonstrations using the CLI can often be more clearly explained because the CLI forces you to direct your request to the specific CLI application that deals with that function. When using the Dashboard, it's hard to tell which component is controlling what. Once you understand the process through the CLI, using the Dashboard will be trivial.

NOTE **Using DevStack from chapter XREF ch02**
The examples covered in this chapter will be executed on the OpenStack instance deployed by DevStack, as described in chapter XREF ch02. If you already have your OpenStack instance from chapter XREF ch02 set up, you're ready to run the examples in this chapter.

Earlier in this chapter, in listing 3.1, you set your environment variables to represent the `Demo` user in the `Demo` tenant. Because you'll be creating a new tenant, you'll need to set your environment variables to represent the `Admin` user in the `Admin` tenant, as shown in the following listing.

Listing 3.5 Prepare your shell session as admin

```
source /opt/stack/python-novaclient/tools/nova.bash_completion  (1)
source openrc admin admin  (2)
```

(1) Set variables for shell "completion" (so that pressing Tab after entering "something /bo" completes "something /boot".
(2) Run this command from the ~/devstack directory. This command will set environmental variables so that when you run OpenStack CLI commands, your identity will be <user: admin> in <tenant:admin>.

WARNING **Admin or demo**

In the previous subsection you set environmental variables to refer to the `demo` user. Under Linux, running as the *root* user when it's not necessary is considered bad practice because it's too easy to make possibly disruptive changes by accident. You can consider unnecessarily running commands as the `admin` user to be a similarly bad practice in OpenStack.

3.3.2 Creating tenants, users, and roles

In this section you'll create a new tenant and user. You'll then assign a role to your new user in your new tenant.

CREATING A TENANT

Use the command shown in listing 3.6 to create your new tenant.

Listing 3.6 Creating a new tenant

```
keystone tenant-create --name General
```

When you run the command, you'll see output like the following:

```
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |                                  |
|   enabled   |               True               |
|      id     | 9932bc0607014caeab4c3d2b94d5a40c |
|     name    |             General              |
+-------------+----------------------------------+
```

You've now created a new tenant that will be referenced when creating other OpenStack objects. Take note of the tenant ID generated in this process; you'll need this ID for the next steps. Figure 3.2 illustrates the tenant you just created. The `admin` and `Member` roles were created as part of the DevStack deployment of OpenStack. The sidebar explains how to list all tenants and roles for a particular OpenStack deployment.

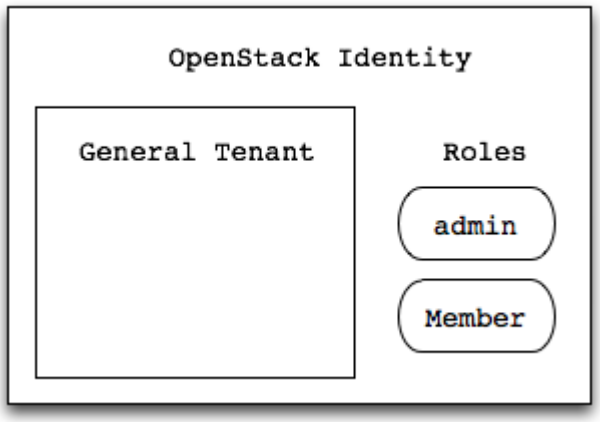


Figure 3.2 The created tenant

SIDEBAR

Listing tenants and roles

You can list all tenants on the system as follows:

```
devstack@devstack:~/devstack$ keystone tenant-list
+----------------------------------+--------------------+---------+
|                id                |        name        | enabled |
+----------------------------------+--------------------+---------+
| 9932bc0607014caeab4c3d2b94d5a40c |      General       |   True  |
| b1c52f4025d244f883dd47f61791d5cf |       admin        |   True  |
| 166c9cab0722409d8dbc2085acea70d4 |      alt_demo      |   True  |
| 324d7477c2514b60ac0ae417ce3cefc0 |        demo        |   True  |
| fafc5f46aaca4018acf8d05370f2af57 | invisible_to_admin |   True  |
| 81548fee3bb84e7db93ad4c917291473 |      service       |   True  |
+----------------------------------+--------------------+---------+
```

You can similarly list all roles on the system with the following command:

```
devstack@devstack:~/devstack$ keystone role-list
+----------------------------------+---------------+
|                id                |      name     |
+----------------------------------+---------------+
| 4b303a1c20d64deaa6cb9c4dfacc33a9 |     Member    |
| 291d6a3008c642ba8439e42c95de22d0 | ResellerAdmin |
| 9fe2ff9ee4384b1894a90878d3e92bab |    _member_   |
| 714aaa9d30794920afe25af4791511a1 |     admin     |
| b2b1621ddc7741bd8ab90221907285e0 |  anotherrole  |
| b4183a4790e14ffdaa4995a24e08b7a2 |    service    |
+----------------------------------+---------------+
```

CREATING A USER

Now that the tenant has been created, you can go ahead and create a new user, as shown in the following listing.

Listing 3.7 Creating a new user

```
keystone user-create  (1)
--name=johndoe  (2)
--pass=openstack1  (3)
--tenant-id 9932bc0607014caeab4c3d2b94d5a40c  (4)
--email=johndoe@testco.com  (5)
```

(1) Tell OpenStack Identity (Keystone) to create a new user
(2) Set user name as johndoe
(3) Set password for johndoe as openstack1
(4) Set the default tenant for johndoe to General
(5) Set johndoe's email to johndoe@testco.com

When you run the command, you'll get output like the following:

```
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |        johndoe@testco.com        |
| enabled  |               True               |
|    id    | 21b27d5f7ba04817894d290b660f3f44 |
|   name   |             johndoe              |
| tenantId | 9932bc0607014caeab4c3d2b94d5a40c |
+----------+----------------------------------+
```

You've now created a new user. Take note of the user ID generated in this process because it will be needed in the next step. Figure 3.3 now includes the user you just created in the `General` tenant.

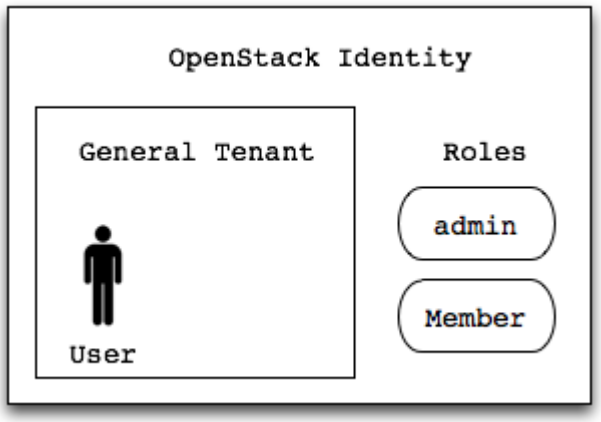


Figure 3.3 The created user

SIDEBAR **Listing users in a tenant**

You can list all users in a tenant with the following command:

```
devstack@devstack:~/devstack$ keystone user-list \
 --tenant-id 9932bc0607014caeab4c3d2b94d5a40c   (1)
+-------------+---------+---------+--------------------+
|      id     |   name  | enabled |       email        |
+-------------+---------+---------+--------------------+
| 21b2...3f44 | johndoe |   True  | johndoe@testco.com |
+-------------+---------+---------+--------------------+
```

(1) Specify the tenant-id for your tenant. In this case you're checking the General tenant.

You now need to add a role to your new user in your new tenant. We'll do that next.

ASSIGNING A ROLE

In order to assign a role to a user in a specific tenant, you need to define the `tenant-id`, `user-id`, and `role-id`.

You can use the `General` tenant, which was created at the beginning of this section and the `johndoe` user, which was created in the previous step. You want allow the user `johndoe` to be able to create instances in the `General` tenant, and to do this you must assign the `Member` `role-id` to `johndoe` in the `General` tenant.

To find the `Member` `role-id`, you need to query OpenStack Identity roles, as shown in the following listing.

Listing 3.8 Listing OpenStack roles

```
keystone role-list
```

❶ **List all roles on this OpenStack deployment**

The list of roles will look something like this:

```
+----------------------------------+---------------+
|                id                |      name     |
+----------------------------------+---------------+
| 4b303a1c20d64deaa6cb9c4dfacc33a9 |     Member    |
| 291d6a3008c642ba8439e42c95de22d0 | ResellerAdmin |
| 9fe2ff9ee4384b1894a90878d3e92bab |    _member_   |
| 714aaa9d30794920afe25af4791511a1 |     admin     |
| b2b1621ddc7741bd8ab90221907285e0 |  anotherrole  |
| b4183a4790e14ffdaa4995a24e08b7a2 |    service    |
+----------------------------------+---------------+
```

You now have all of the information you need to assign your newly created user as `Member` of your newly created tenant. Run the command shown in the following listing, substituting the appropriate IDs for your system.

Listing 3.9 Adding a role

```
keystone user-role-add \   ❶
--tenant-id 9932bc0607014caeab4c3d2b94d5a40c \   ❷
--user-id 21b27d5f7ba04817894d290b660f3f44 \   ❸
--role-id 4b303a1c20d64deaa6cb9c4dfacc33a9   ❹
```

❶ Tell OpenStack Identity (Keystone) to add a role for a user to a tenant
❷ Assign tenant-id
❸ Assign user-id
❹ Assign role-id

This command will generate no output if it’s successful.

You’ve now assigned a role to a user in a tenant. Figure 3.4 illustrates the role you just assigned to the `johndoe` user in the `General` tenant.

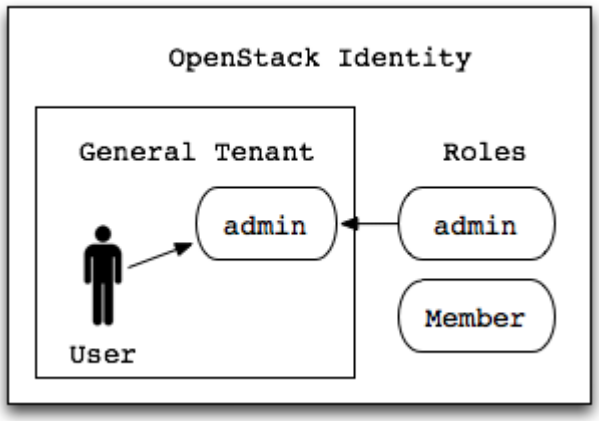


Figure 3.4 The assigned role

At this point you can access the OpenStack Dashboard and log in using the `johndoe` user and `openstack1` password. When you do, you'll be taken to the `General` tenant/project management screen. If you try to create a new instance, you'll notice that no networks exist. You'll create a new tenant network next.

3.3.3 Tenant networks

OpenStack Networking (Neutron) is both loved and hated. To make sure you have more of the former experience than the latter, you should get your feet wet as soon as possible. In this section, you'll run through some simple tenant network configurations.

First, though, you need to understand the basic differences between how traditional "flat" networks are configured for virtual and physical machines and how OpenStack Networking will be demonstrated. The term *flat* refers to the absence of a virtual routing tier as part of the virtual server platform; in traditional configurations, the VM has direct access to a network, as if you plugged a physical device into a physical network switch. Figure 3.5 illustrates a flat network connected to a physical router.

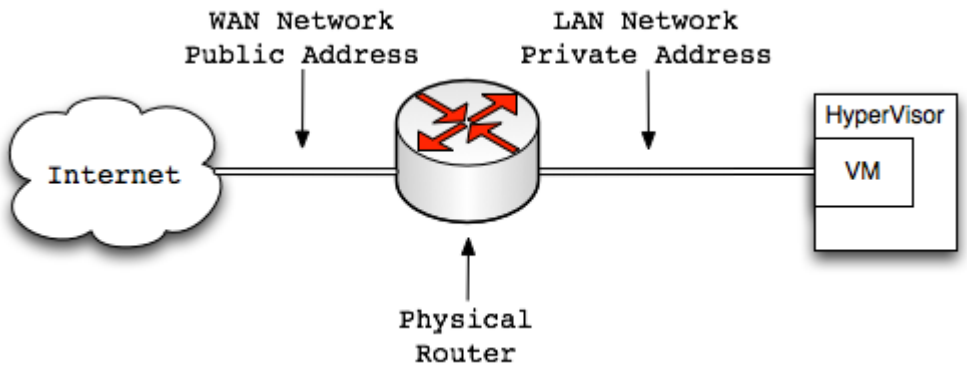


Figure 3.5 Traditional routed network

In this type of deployment, all network services (Dynamic Host Configuration Protocol (DHCP), load balancing, routing, and so on) beyond simply switching (Open System Interconnection (OSI) Model, Layer 2), must be provided outside of the virtual environment. For most systems administrators, this type of configuration will be very familiar, but this is not how we'll demonstrate the power of OpenStack. You can make OpenStack Networking behave like a traditional flat network, but that approach will limit the benefits of the OpenStack framework.

In this section, you'll be building an OpenStack tenant network from scratch. Figure 3.6 illustrates the differences between a more traditional network and the type of network you'll be building.

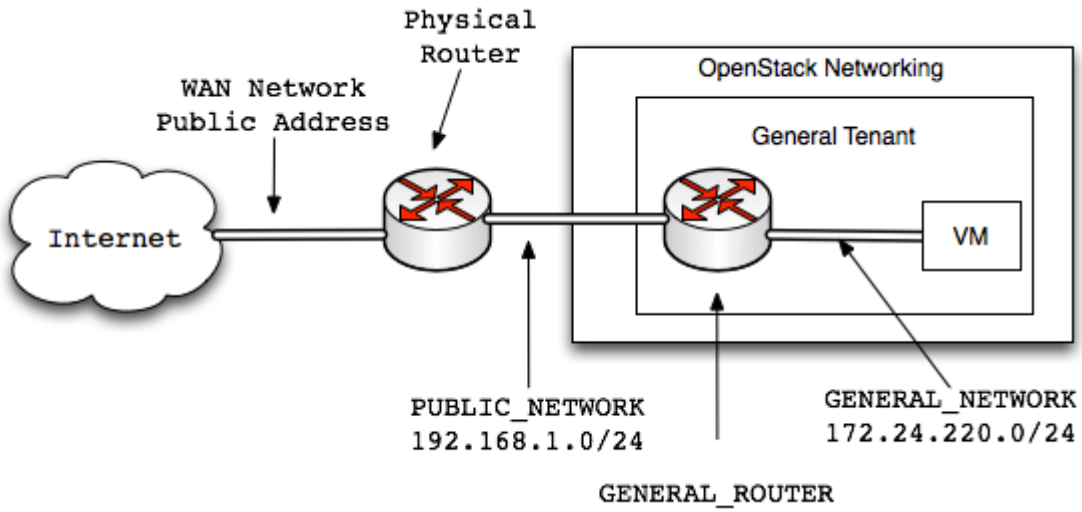


Figure 3.6 OpenStack tenant network

Note that compared to the traditional flat network, the OpenStack tenant network includes an additional router that resides within the virtual environment. The addition of the virtual router in the tenant separates the *internal network*, shown as *GENERAL_NETWORK* , from the *external network*, shown as *PUBLIC_NETWORK*. VMs communicate with each other using the internal network, and the *virtual router*, shown as *GENERAL_ROUTER*, uses the external network for communication outside the tenant.

SIDEBAR **Set your environment variables**

The configurations in the following subsections require that the OpenStack CLI environmental variables are set. To set the environment variables, execute the following commands in your shell:

```
source /opt/stack/python-novaclient/tools/nova.bash_completion
source openrc admin admin
```

NETWORK (NEUTRON) CONSOLE

Neutron commands can be entered through the Neutron console (which is like a command line for a network router or switch) or directly through the CLI. The console is very handy if you know what you're doing, and it's a natural choice for those familiar with the Neutron command set. But for the sake of clarity, we'll demonstrate each action as a separate command using CLI commands. There are many things you can do with the Neutron CLI and console that you can't do in the Dashboard.

The distinction between the Neutron console and Neutron CLI will be made clear in the following subsections. While the examples in this chapter will be executed using the CLI, you'll still need to know how to access the Neutron console. As you can see from the following listing, it's a simple matter of using the `neutron` command.

Listing 3.10 Accessing the Neutron console

```
devstack@devstack:~/devstack$ neutron
(neutron) help

Shell commands (type help <topic>):
===================================
...
(neutron)
```

1. **Using the neutron command without arguments will take you to the console**
2. **All of the subcommands will be listed here**

You can now access the Neutron interactive console. Any CLI configurations can be made in the interactive console or directly on the command line. That's how you'll create a new network.

CREATING INTERNAL NETWORKS

The first step in creating a tenant-based network is to configure the internal network that will be used directly by instances in your tenant. The internal network works on ISO Layer 2 (L2), so for the network types this is the virtual equivalent of providing a network switch to be used exclusively for a particular tenant. Listing 3.11 shows the code used to create a new network for your tenant.

Listing 3.11 Creating an internal network

```
devstack@devstack:~/devstack$ neutron net-create \   ❶
--tenant-id 9932bc0607014caeab4c3d2b94d5a40c \   ❷
GENERAL_NETWORK   ❸
```

❶ Tell OpenStack Networking (Neutron) to create a new network
❷ Specify the tenant where the network should be created
❸ Specify the name of the tenant network

You'll see output like the following when you create the network.

```
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 35a387fd-892f-47ad-a226-e8d0f2f0636b |
| name                      | GENERAL_NETWORK                      |
| provider:network_type     | local                                |
| provider:physical_network |                                      |
| provider:segmentation_id  |                                      |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 9932bc0607014caeab4c3d2b94d5a40c     |
+---------------------------+--------------------------------------+
```

Figure 3.7 illustrates the `GENERAL_NETWORK` created for your tenant. The figure shows the network connected to a VM, which will be accurate once you create a new instance and attached the network you just created.

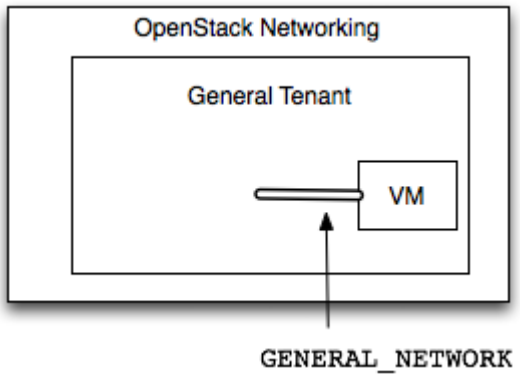


Figure 3.7 The newly created internal network

You've now created an internal network. The next step is to create an internal subnet for this network.

CREATING INTERNAL SUBNETS

The internal network you just created inside your tenant is completely isolated from other tenants. This will be a strange concept to those who work with physical servers, or even to those who generally expose their virtual machines directly to physical networks. Most people are used to connecting their servers to the network, and network services are generally provided on a data center or enterprise level. We don't typically think about networking and computation being controlled under the same framework.

As previously mentioned, OpenStack can be configured to work in a flat network configuration, but there are many advantages to letting OpenStack manage the network stack. In this section you'll create a subnet for your tenant; this can be though of as a ISO Layer 3 (L3) provisioning of your tenant. You might be thinking to yourself, "What are you talking about? You can't just provision L3 services on the network!" Or perhaps, "I already have L3 services centralized in my data center. I don't want OpenStack to do this for me!" By the end of this section, or perhaps by the end of the book, you'll have your own answers to these questions. For the time being, just trust that the OpenStack experience includes benefits that are either enriched or not otherwise possible without the advanced network virtualization provided by OpenStack Networking.

What does it mean to create a new subnet for a specific network? Basically, you describe the network you want to work with, and then describe the address ranges you plan on using on that network. In this case, you'll assign the new subnet to the `GENERAL_NETWORK` in the `General` tenant. You must also provide an address range for the subnet. In this context, the term subnet refers to both an OpenStack

subnet, which is defined as part of the OpenStack network, and they IP subnet which is defined as part of the OpenStack subnet creation process. You can use your own address range as long as it doesn't exist in the tenant or a shared tenant. One of the interesting things about OpenStack is that you could use the same address range for every internal subnet in every tenant. The following listing shows the command used to create a subnet.

Listing 3.12 Creating an internal subnet for a network

```
neutron subnet-create \   ❶
--tenant-id 9932bc0607014caeab4c3d2b94d5a40c \   ❷
GENERAL_NETWORK \   ❸
172.24.220.0/24   ❹
```

❶ Tell OpenStack Networking (Neutron) to create a new subnet
❷ Specify the tenant where the subnet should be created
❸ Specify the name of the network where the subnet should be created
❹ Specify the subnet range to be used on the internal network in CIDR notation

When you run this command, you'll see output like the following:

```
Created a new subnet:
+------------------+-----------------------------------------------------+
| Field            | Value                                               |
+------------------+-----------------------------------------------------+
| allocation_pools | {"start":"172.24.220.2","end":"172.24.220.254"} |
| cidr             | 172.24.220.0/24                                     |
| dns_nameservers  |                                                     |
| enable_dhcp      | True                                                |
| gateway_ip       | 172.24.220.1                                        |
| host_routes      |                                                     |
| id               | 40d39310-44a3-45a8-90ce-b04b19eb5bb7                |
| ip_version       | 4                                                   |
| name             |                                                     |
| network_id       | 35a387fd-892f-47ad-a226-e8d0f2f0636b                |
| tenant_id        | 9932bc0607014caeab4c3d2b94d5a40c                    |
+------------------+-----------------------------------------------------+
```

SIDEBAR **CIDR notation**

Classless Inter-Domain Routing (CIDR) is a compact way to represent subnets.

For internal subnets, most people use a private class C address range, which was actually a class C of the original public classful ranges. In the case of a class C range, 8 bits are used for the subnet mask, so there are 2^8 = 256 addresses, but CIDR is expressed in the form <First address>/<Size of host bit field>, where 32 bits – 8 bits = 24 bits.

This might seem confusing, but luckily there are many online subnet calculators you can use if you aren't up on your binary math.

You now have a new subnet assigned to your `GENERAL_NETWORK`. Figure 3.8 illustrates the assignment of the subnet to the `GENERAL_NETWORK`. This subnet is still isolated, but you're one step closer to connecting your private network with a public network.

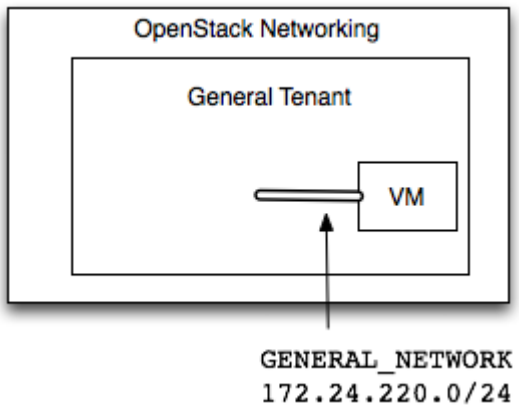


Figure 3.8 The newly created internal subnet

Next you need to add a router to the subnet you just created. Make a note of your subnet ID, because it will be needed in the following sections.

CREATING ROUTERS

Routers, put simply, route traffic between interfaces. In this case you have an isolated network on your tenant and you want to be able to communicate with other tenant networks or networks outside of OpenStack. The following listing shows how to create a new tenant router.

Listing 3.13 Creating a router

```
neutron router-create \   ❶
--tenant-id 9932bc0607014caeab4c3d2b94d5a40c \   ❷
GENERAL_ROUTER   ❸
```

❶ Tell OpenStack Networking (Neutron) to create a new router
❷ Specify the tenant where the subnet should be created
❸ Specify the name of the router

When you create the router, you'll see output like the following:

```
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | df3b3d29-104f-46ca-8b8d-50658aea3f24 |
| name                  | GENERAL_ROUTER                       |
| status                | ACTIVE                               |
| tenant_id             | 9932bc0607014caeab4c3d2b94d5a40c     |
+-----------------------+--------------------------------------+
```

Figure 3.9 illustrates the router you created in your tenant.

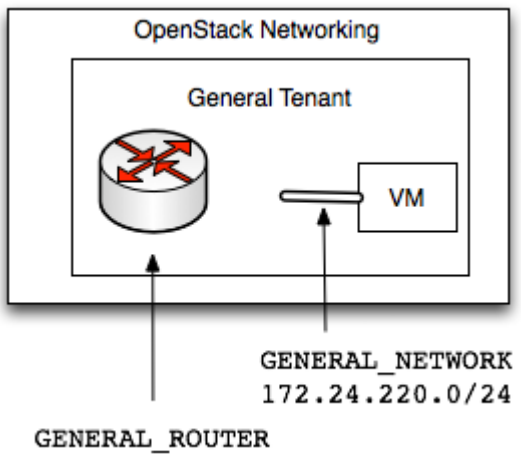


Figure 3.9 The newly created internal router

You have a new router, but your tenant router and subnet aren't yet connected. The following listing shows how to connect your subnet to your router.

Listing 3.14 Adding a router to the internal subnet

```
neutron router-interface-add \   ❶
df3b3d29-104f-46ca-8b8d-50658aea3f24 \   ❷
40d39310-44a3-45a8-90ce-b04b19eb5bb7   ❸
```

❶ Tell OpenStack Networking (Neutron) to add the internal subnet to the router
❷ Specify the router ID
❸ Specify the subnet ID

When the router is created, you'll see output like (Ids are auto generated so yours will be unique) the following:

```
Added interface 0a1a97e3-ad63-45bf-a55f-c7cd6c8cf4b4 to
router df3b3d29-104f-46ca-8b8d-50658aea3f24
```

Figure 3.10 illustrates the new GENERAL_ROUTER router connected to your internal network, GENERAL_NETWORK.

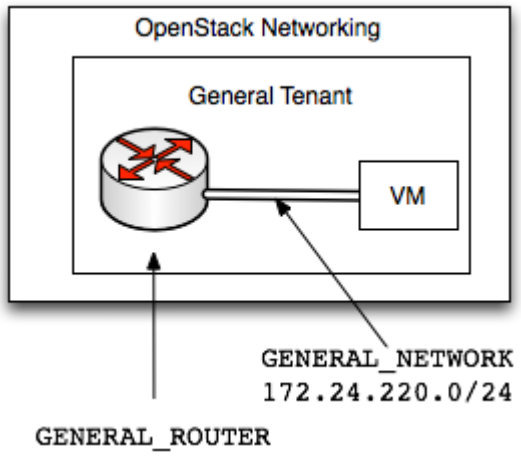


Figure 3.10 The new router connected to the internal network

The process of adding a router to a subnet will actually create a *port* on the local virtual switch. You can think of a port as a device plugged into your virtual network port. In this case the device is the GENERAL_ROUTER, the network is the GENERAL_NETWORK, and the subnet is 172.24.220.0/24.

NOTE **DHCP agents**

In past versions of OpenStack Networking, you had to manually add Dynamic Host Configuration Protocol (DHCP) agents to your network—the DHCP agent is used to provide your instances with an IP address. In current versions, the agent is automatically added for you the first time you create an instance, but in advanced configurations it's still handy to know that agents (of all kinds) can be manipulated through Neutron.

The router will use the address specified during subnet creation (defaulting to the first available address), and unless you have already created an instance on this network, this will be the first port (device) on this network. If you create an instance, you should be able to communicate with the router address of 172.24.220.1, but you won't yet be able to route packets to other networks. A router isn't much good when it's only connected to one network, so your next step is to connect the router to a public network.

CONNECTING A ROUTER TO A PUBLIC NETWORK

Before you can add a public interface, you need to find it. In previous steps, you created an internal network, internal subnet, and router, so you knew their ID values. If you're working with the OpenStack deployment produced by DevStack in chapter XREF ch02, a public interface will already exist, and the following listing shows how you can list your external networks.

Listing 3.15 Listing external networks

```
neutron net-external-list
```

The preceding command will produce output like the following. If no external networks exist, jump ahead to section 3.5.3.6 and create a new external network.

```
(neutron) net-external-list
+---------------+--------+---------------------------------------+
| id            | name   | subnets                               |
+---------------+--------+---------------------------------------+
| 4eed3f..34b23d | public | e9643dc8...df4d34099109 192.168.1.0/24 | ❶
+---------------+--------+---------------------------------------+
```

❶ The ID and subnet fields have been shortened to fit on the page

You've now listed all public networks; make a note of the network ID. In the example there will be a single public network, but in a production environment there could be many. You can select the appropriate network ID, based on the desired subnet in the listing. The network ID will be used along with the previously referenced router ID to add the existing public network to your router.

Previously, in listing 3.14, you used the command `router-interface-add` to connect your internal network to your router. You could use the same command to add the public network, but we're going to designate this public network as the router gateway so we'll use the `router-gateway-set` command. The router gateway will be used to translate (route) traffic from internal OpenStack networks to external networks.

Go ahead and add the public network as the gateway for the router using the following command.

Listing 3.16 Add existing external network as router gateway

```
neutron router-gateway-set \   ❶
df3b3d29-104f-46ca-8b8d-50658aea3f24 \   ❷
4eed3f65-2f43-4641-b80a-7c09ce34b23d   ❸
```

❶ Tell OpenStack Networking (Neutron) to add the existing external network as a router gateway
❷ Specify the ID of the tenant router
❸ Specify the ID of the existing external network

The preceding command will produce the following output:

```
Set gateway for router df3b3d29-104f-4
6ca-8b8d-50658aea3f24
```

Figure 3.11 illustrates the external `PUBLIC_NETWORK` network you added to `GENERAL_ROUTER` as a network gateway.

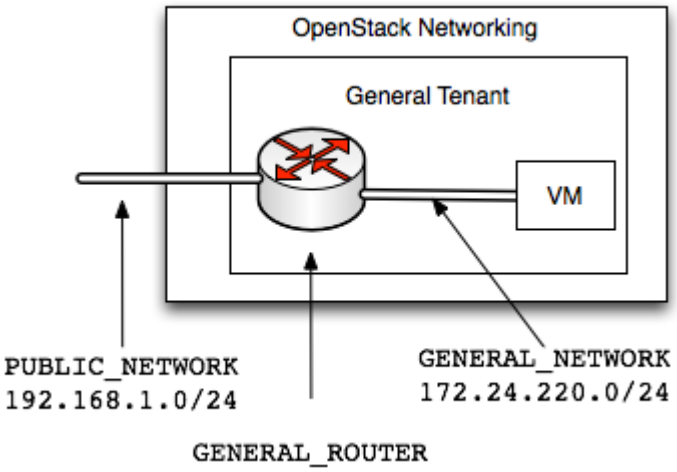


Figure 3.11 An existing network assigned as a router gateway

CREATING AN EXTERNAL NETWORK

The configurations in the following subsections require that OpenStack CLI environmental variables are set, as shown below.

```
source /opt/stack/python-novaclient/tools/nova.bash_completion
source openrc admin admin
```

In section 3.5.3.2 you created a network that was specifically for your tenant. In this subsection you'll create a public network that can be used by multiple tenants. This public network can be attached to a private router as a network gateway, as described in the previous section.

NOTE **Using the DevStack network from chapter XREF ch02**

The examples covered in this section will be executed on an OpenStack instance deployed by DevStack, as described in chapter XREF ch02. The deployment performed in that chapter provided the necessary network configuration to allow the addition of external networks. If you already have your OpenStack instance from chapter XREF ch02, you're ready to complete the examples in this section. In upcoming chapters you'll learn to manually make the network configurations that the previous DevStack deployment made for you.

In listing 3.15 you saw how to list networks that were designated as "external." If you're working from the DevStack deployment from chapter XREF ch02, an

external network will already exist. This isn't a problem, because you can have many external networks in OpenStack Networking.

Only the `admin` user can create external networks, and if not specified, the new external network will be created in the `admin` tenant. Go ahead and create a new external network as shown in the following listing.]

Listing 3.17 Creating an external network

```
neutron net-create \    ❶
new_public   ❷
--router:external=True   ❸
```

❶ Tell OpenStack Networking (Neutron) to create a new network
❷ Specify the network name
❸ Designate this network as an external network

When the network is created, you'll see output like this:

```
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 8701c5f1-7852-4468-9dae-ff8a205296aa |
| name                      | new_public                           |
| provider:network_type     | local                                |
| provider:physical_network |                                      |
| provider:segmentation_id  |                                      |
| router:external           | True                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | b1c52f4025d244f883dd47f61791d5cf     |
+---------------------------+--------------------------------------+
```

NOTE **Confirming the network's tenant**
If you want to confirm that the network was created in the `admin` tenant, you can retrieve all tenant IDs as shown in the "Listing tenants and roles" sidebar in section 3.5.2.1.

You now have a network that's designated as a external network, as shown in figure 3.12. This network will be in the `admin` tenant, and would not currently be visible in the `General` tenant.

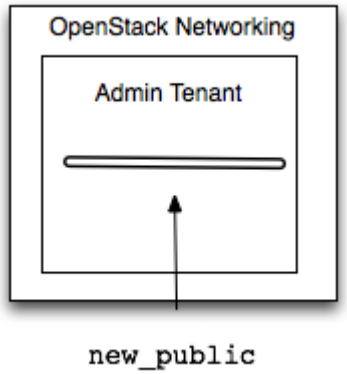


Figure 3.12 Created external network

Before you can use this network as a gateway for your tenant router, you must first add a subnet to the external network you just created. You'll do that next.

CREATING AN EXTERNAL SUBNET

You must now create an external subnet, as shown in the following listing.

Listing 3.18 Creating an external subnet

```
neutron subnet-create \   (1)
--gateway 192.168.2.1 \   (2)
--allocation-pool start=192.168.2.2,end=192.168.2.254 \   (3)
new_public \   (4)
192.168.2.0/24 \   (5)
--enable_dhcp=False   (6)
```

(1) Tell OpenStack Networking (Neutron) to create a new subnet
(2) Set the gateway address to the first available address
(3) Define the range of addresses available for allocation in the subnet
(4) Define the external network where the subnet will be assigned
(5) Define the subnet in CIDR format
(6) Specify that OpenStack shouldn't provide DHCP services for this subnet

When the subnet is created, you'll see output like the following.

```
Created a new subnet:
+------------------+-----------------------------------------------------+
| Field            | Value                                               |
+------------------+-----------------------------------------------------+
| allocation_pools | {"start": "192.168.2.2", "end": "192.168.2.254"} |
| cidr             | 192.168.2.0/24                                      |
| dns_nameservers  |                                                     |
| enable_dhcp      | False                                               |
| gateway_ip       | 192.168.2.1                                         |
```

```
| host_routes       |                                                      |
| id                | 2cfa7201-d7f3-4e0c-983b-4c9f3fcf3caa                 |
| ip_version        | 4                                                    |
| name              |                                                      |
| network_id        | 8701c5f1-7852-4468-9dae-ff8a205296aa                 |
| tenant_id         | b1c52f4025d244f883dd47f61791d5cf                     |
+-------------------+------------------------------------------------------+
```

You now have the subnet 192.168.2.0/24 assigned to the external `new_public` network. The subnet and external network you just created, as shown in figure 3.13, can now be used by an OpenStack Networking router as a gateway.

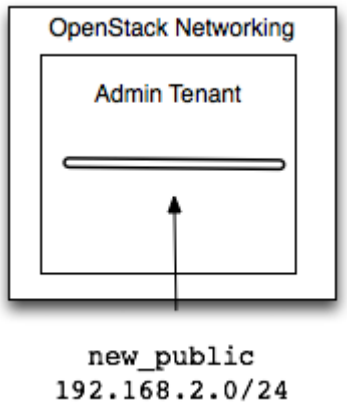


Figure 3.13 The newly created external subnet

Figure 3.14 shows your current state, assuming you followed the examples in the previous subsections.

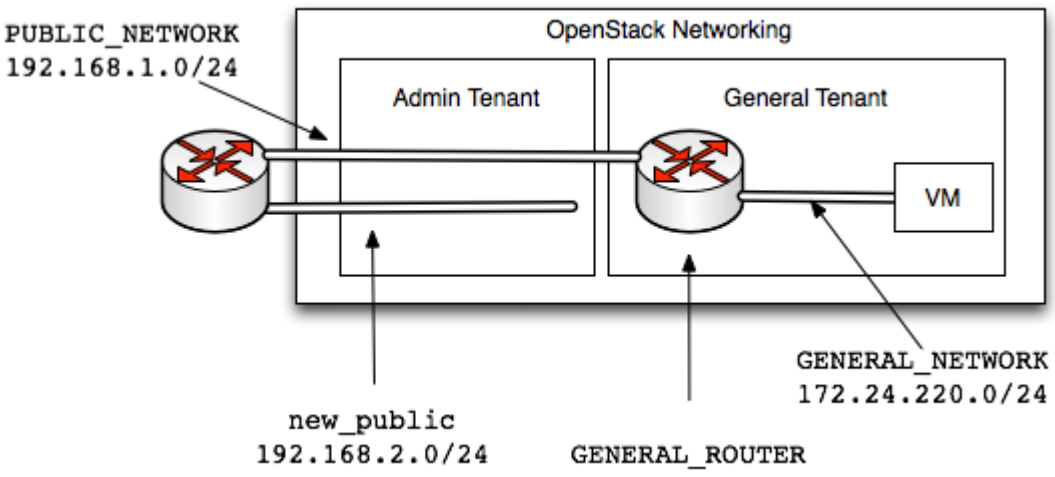


Figure 3.14 External gateways for tenants

Even if your network doesn't resemble the figure, the main idea here is to illustrate that there are two possible external networks that could be assigned as gateways for the tenant router. The `new_public` and `PUBLIC_NETWORKS` are separate virtual networks and have been assigned diffrence subnets. Currently the `PUBLIC_NETWORK` is assigned as the gateway for your `GENERAL_ROUTER`. This means that any instance network traffic that's not directly network-connected to your tenant (such as access to the internet or other tenants) will use this (gateway) network as its link to the outside world.

SIDEBAR **Listing routers**

To list all the routers in the system, use the `neutron router-list` command, like this:

```
devstack@devstack:~/devstack$ neutron router-list
+--------+----------------+-----------------------------------------
| id     | name           | external_gateway_info
+--------+----------------+-----------------------------------------
| df..24 | GENERAL_ROUTER | {"network_id": "4e..3d", "enable_snat": ..
+--------+----------------+-----------------------------------------
```

WARNING **Remove floating addresses**

Floating addresses are external IP addresses that have a one-to-one relationship with internal IP addresses assigned to instances. Floating IP addresses must be removed from instances and be deallocated before removing the router gateway. Floating addresses are directly associated with the external network, so any attempt to remove an external network with these associations still in place will result in failure.

Let's assume you want to change the current gateway from `PUBLIC_NETWORK` to `new_public`. You must first remove the old gateway and then add the new one. The following listing shows how to clear the existing gateway.

Listing 3.19 Clearing a router gateway

```
neutron router-gateway-clear \   ❶
df3b3d29-104f-46ca-8b8d-50658aea3f24   ❷
```

❶

1. Tell OpenStack Networking (Neutron) to clear the currently assigned gateway from a router
2. Specify the router ID where you want the gateway removed

When the gateway is removed, you'll get the following confirmation.

```
Removed gateway from router df3b3d29-104f-46ca-8b8d-50658aea3f24
```

Once the existing gateway has been removed, your tenant will be configured in the state shown in figure 3.15, where no external networks are connected to the tenant.

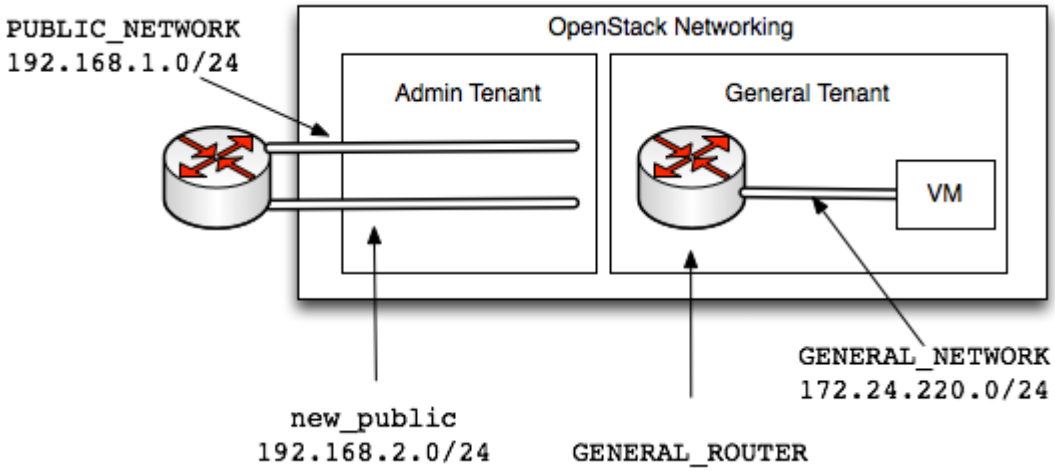


Figure 3.15 Removed router gateway

Your tenant network configuration is now back to where is was before you added the existing external network as a gateway in the previous section. You can now add the `new_public` external network as a gateway instead of `PUBLIC_NETWORK`. The following listing shows the commands required to complete the network configuration.

Listing 3.20 Adding a new external network as the router gateway

```
neutron router-gateway-set \   (1)
df3b3d29-104f-46ca-8b8d-50658aea3f24 \   (2)
8701c5f1-7852-4468-9dae-ff8a205296aa   (3)
```

1. Tell OpenStack Networking (Neutron) to add the new external network as a router gateway
2. Specify the ID of the tenant router
3. Specify the ID of the external network to be assigned as the gateway

When the gateway is set, you'll see output like the following.

```
Set gateway for router
df3b3d29-104f-46ca-8b8d-50658aea3f24
```

Figure 3.16 illustrates the assignment of the new network, `new_public`, as the gateway for the `GENERAL_ROUTER` in the `General` tenant. You can confirm this setting by running the `neutron router-show <router-id>` command, where the `<router-id>` is the ID of the `GENERAL_ROUTER`. The command will return the `external_gateway_info`, which lists the currently assigned gateway network. Optionally, you can log into the OpenStack Dashboard and look at your tenant network. The `PUBLIC_NETWORK` will no longer be there, and it will be replaced by the `new_public` network.

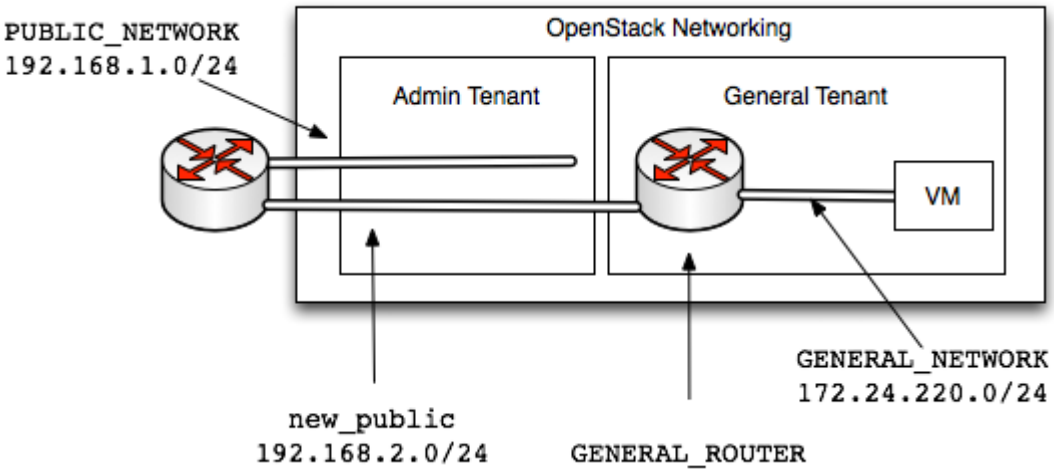


Figure 3.16 The new network assigned as a router gateway

You've now learned how to create tenants, users, and networks. The tenant model allows multiple users to operate under the same environment without affecting each other. As a OpenStack administrator, you wouldn't want a single tenant to be able to use more resources than their share, so OpenStack implements a quota system for several of the major components, including Compute, Storage, and Networking services. We'll look at that next.

3.4 Quotas

Quotas are applied on the tenant and tenant user level to limit the amount of resources any one tenant can use. When you create a new tenant, a default quota is applied. Likewise, when you add users to your tenant, the tenant quota is applied to that user. By default, all users will have the same quota as the tenant quota, but you have the option of reducing a user's quota in a tenant independently of the overall tenant quota.

Consider the case where you have an application administrator and database administrator sharing the same tenant for a project. You might want to assign half of the tenant resource to each user. On the other hand, if you increase a user's quota in a tenant in excess of the tenant quota, the tenant quota will increased to match the new user value.

Quota management is an important operational component for an OpenStack administrator to understand. In the rest of this subsection, you'll work through some CLI exercises to display and update quotas for tenants and tenant users via the Compute component. As with the majority of tenant configurations, you can also make these changes through the Dashboard or directly with the API.

3.4.1 Tenant quotas

In order modify quota settings, you'll need to know the tenant ID you want to work with and a user ID that is currently in that tenant. The following example below shows how you can list all tenants on the system and find the tenant ID.

```
devstack@devstack:~/devstack$ keystone tenant-list
+----------------------------------+--------------------+---------+
|                id                |        name        | enabled |
+----------------------------------+--------------------+---------+
| 9932bc0607014caeab4c3d2b94d5a40c |      General       |   True  |
| b1c52f4025d244f883dd47f61791d5cf |       admin        |   True  |
| 166c9cab0722409d8dbc2085acea70d4 |      alt_demo      |   True  |
| 324d7477c2514b60ac0ae417ce3cefc0 |        demo        |   True  |
| fafc5f46aaca4018acf8d05370f2af57 | invisible_to_admin |   True  |
| 81548fee3bb84e7db93ad4c917291473 |      service       |   True  |
+----------------------------------+--------------------+---------+
```

Listing 3.21 shows the command you can use to show the quota settings for a particular tenant.

Listing 3.21 Showing the Compute quota for a tenant

```
nova quota-show  \   ❶
 --tenant 9932bc0607014caeab4c3d2b94d5a40c   ❷
```

❶ Tell OpenStack Compute (Nova) to show the quota information
❷ Specify the tenant ID for the quota query

The quota information will be displayed as follows. Consult the OpenStack operations manual for current quota items and unit values: http://docs.openstack.org/openstack-ops/content/projects_users.html

```
+-----------------------------+-------+
| Quota                       | Limit |
+-----------------------------+-------+
| instances                   | 10    |
| cores                       | 10    |
| ram                         | 51200 |
| floating_ips                | 10    |
| fixed_ips                   | -1    |
| metadata_items              | 128   |
| injected_files              | 5     |
| injected_file_content_bytes | 10240 |
| injected_file_path_bytes    | 255   |
| key_pairs                   | 100   |
| security_groups             | 10    |
| security_group_rules        | 20    |
+-----------------------------+-------+
```

You now know the quota for a particular tenant. To retrieve the default quota assigned to new tenants, simply omit the tenant ID from the command (`nova quota-show`).

Now suppose you're an OpenStack administrator and you've created a tenant for a department in your company. This department has just been tasked with deploying a new application, which will require resources that exceed their existing tenant quota. In this case, you'd want to increase the quota for the entire tenant. The command for doing this is shown in the following listing.

Listing 3.22 Updating the Compute quota for tenant

```
nova quota-update \   ❶
 --<quota_key> <quota_value> \   ❷
 <tenant-id>   ❸
```

❶ Tell Compute that you want to update your quota values
❷ Assign a new quota value to a specific quota item
❸ Specify the tenant whose quota you want to update

A list of quota keys can be obtained by displaying the quota values, as shown previously in listing 3.21. In the following example, the `cores <quota_key>` is updated.

```
devstack@devstack:~/devstack$ nova quota-update \
 --cores 20 \   ❶
 9932bc0607014caeab4c3d2b94d5a40c   ❷
```

❶ Set <quota-key> to cores and <quota-value> to 20
❷ Specify the ID for the General tenant

You have now successfully updated your tenant quota, and your users can now start assigning additional resources. If you rerun the command shown in listing 3.21, you'll see that the quota has been updated.

Next we'll look at how you can work with quotas on the tenant user level. As you'll soon see, the ability to apply quotas on the user level for a specific tenant can be very useful in managing resource utilization.

3.4.2 Tenant user quotas

In some cases there might be only a single user in a tenant. In these cases, you'd only need quota management on the tenant level. But what if there are multiple users all in the same tenant? OpenStack provides the ability to manage quotas for individual users on a tenant level. This means that an individual user can have separate quotas for each tenant for which they are a member.

Suppose that one of your users with a role on a specific tenant is only responsible for a single instance. Despite being responsible for only one instance, this user has on several occasions added additional instances to this tenant. The additional instances count against the overall tenant quota, so although the user in question *should* only have one instance, they might actually have several. To

prevent this from happening, you can adjust the user's quota for the tenant, and not the entire tenant quota. The following listing displays the existing quota for a particular user.

Listing 3.23 Showing the Compute quota for a tenant user

```
nova quota-show \
 --user <user_id> \   (1)
 --tenant <tenant_id>   (2)
```

(1) Specify the user ID in a tenant for the query
(2) Specify the tenant ID for the query

The following example shows the user ID related to `johndoe`, which you created in section 3.5.2.2, and the tenant ID related to the `General` tenant, which you created in section 3.5.2.1. Your actual IDs will differ from the examples listed here.

```
devstack@devstack:~/devstack$ nova quota-show \
 --user 21b27d5f7ba04817894d290b660f3f44 \   (1)
 --tenant 9932bc0607014caeab4c3d2b94d5a40c   (2)
+-----------------------------+-------+
| Quota                       | Limit |
+-----------------------------+-------+
| instances                   | 10    |
| cores                       | 10    |
| ram                         | 51200 |
| floating_ips                | 10    |
| fixed_ips                   | -1    |
| metadata_items              | 128   |
| injected_files              | 5     |
| injected_file_content_bytes | 10240 |
| injected_file_path_bytes    | 255   |
| key_pairs                   | 100   |
| security_groups             | 10    |
| security_group_rules        | 20    |
+-----------------------------+-------+
```

(1) Specify the user ID for johndoe
(2) Specify the tenant ID for General

As you can see, the user quotas are the same size as the original tenant quota. By default, users added to a tenant can use all resources assigned to that tenant. For

this tenant, you updated the *cores* value in a previous example, but that only updated the tenant quota, which, as you can see, doesn't automatically increase a user's tenant quota.

Assume that user `johndoe` is a problem user that you want to restrict to running a single instance in the `General` tenant. Listing 3.24 shows the command you can use to do this.

Listing 3.24 Updating the Compute quota for a tenant user

```
nova quota-update \
 --user <user_id> \   (1)
 --<quota_key> <quota_value> \   (2)
 <tenant_id>   (3)
```

(1) Specify the user's ID
(2) Assign a new quota value to a quota item for a specific user
(3) Specify the tenant ID

In the following example, the user *johndoe* is configured an instance quota of one instance in the *General* tenant.

```
devstack@devstack:~/devstack$ nova quota-update \
 --user 21b27d5f7ba04817894d290b660f3f44 \   (1)
 --instances 1 \   (2)
 9932bc0607014caeab4c3d2b94d5a40c   (3)
```

(1) Specify user-id related to johndoe user.
(2) Set <quota-key> to "instances" and <quota-value> to 1 instance.
(3) Specify tenant-id related to General tenant.

You have now restricted the user `johndoe` to running a single instance. You might further want to restrict the resources this user can utilize for their individual instance, such as limiting the number of cores to 4.

3.4.3 Additional quotas

There are additional quota systems for OpenStack Storage and Networking. The arguments for these quota systems are more or less the same, but the quota keys will be different.

You need to access each quota system through the CLI command assigned to the related system component. For OpenStack Compute the command is `nova`, for

Storage it's `cinder`, and for Networking it's `neutron`. The following two listings illustrate accessing quota information for OpenStack Storage and Networking.

Listing 3.25 Showing Storage tenant quota

```
devstack@devstack:~/devstack$ cinder quota-show \   ❶
 9932bc0607014caeab4c3d2b94d5a40c   ❷
+-----------+-------+
|  Property | Value |
+-----------+-------+
| gigabytes |  1000 |
| snapshots |   10  |
|  volumes  |   10  |
+-----------+-------+
```

❶ Tell Storage (Cinder) to show the current quota
❷ Specify the tenant for the query

The following example demonstrates how to display the current Neutron quota for a specific tenant.

Listing 3.26 Showing Networking tenant quota

```
devstack@devstack:~/devstack$ neutron quota-show \   ❶
 9932bc0607014caeab4c3d2b94d5a40c   ❷
+---------------------+-------+
| Field               | Value |
+---------------------+-------+
| floatingip          | 50    |
| network             | 10    |
| port                | 50    |
| router              | 10    |
| security_group      | 10    |
| security_group_rule | 100   |
| subnet              | 10    |
+---------------------+-------+
```

❶ Tell Networking (Neutron) to show the current quota
❷ Specify the tenant for the query

As you can see, the CLI commands for Storage and Networking quotas are very similar to the ones you used with Compute. You can also use the Dashboard for quota configuration.

3.5 Summary

- The Dashboard is intended for end users.
- The CLI and APIs are intended for administration, scripting, and repetitive tasks.
- Anything you can do with the Dashboard, you can do with the CLI or APIs.
- The CLI can be configured to output the API-level calls that are used for a specific command.
- Resources managed by OpenStack are reserved and provisioned based on tenants (projects).
- The terms *tenant* and *project* are used interchangeably in OpenStack, but projects related to resources, users, and rights shouldn’t be confused with OpenStack projects like Compute, Network, and so on.
- Roles determine the rights of a user in a specific tenant.
- Users are assigned a home tenant, but they might hold many roles in other tenants.
- Tenant networks and subnets are generally isolated private networks for a specific tenant.
- Public networks and subnets are generally shared between tenants and are used for external (public) network access.
- Layer 3 services (DHCP, metadata services, and the like) can be provisioned on networks.
- Virtual routers are used to route network traffic from tenant (private) networks to public networks.
- Quotas are assigned to both tenants and to specific users in a tenant.

9 *Architecting your OpenStack*

This section covers

- Using OpenStack to replace existing virtual server platforms
- Why you should build a private cloud
- Choices to make when building your private cloud

In the first part of this book you dipped your toes into OpenStack through the use of DevStack. This part of the book was intended as an introduction to how and why OpenStack is used. In addition to introductions, the goal of the first part of the book was to make you interested enough in OpenStack framework, that you wanted to gain a deeper understanding of how things work under the covers.

In the second part of the book you went through a manual deployment of several core OpenStack components. While it is important that you understand the underlying component interactions that make up OpenStack, the second part of the book was not intended as a blue print for OpenStack deployment. The goal of the second part of the book was to build your confidence in the underlying system, through low-level exposure of the components and configurations. While this level of understanding is useful, the intent of this part of the book was not to prepare you for manual installation of components in a production environment.

The third and final part of the book covers topics related to deploying and utilizing OpenStack in production environments. Specifically, enterprise environments where the typical systems administrator might take care of a wide verity of both infrastructure and applications. Often in enterprise environments, systems engineers drive infrastructure design, deployment, and adoption. The chapters contained the third part of the book are intended to help you develop a successful OpenStack deployment for your environment.

Chapter 9, *Architecting your private cloud*, will cover decisions, both architectural, financial, and operational that you will need to make as you plan your deployment. This chapter is not intended as a cookbook, but rather as a starting reference used in the development of a successful architecture. For a prescriptive approach in developing your architecture, consult the OpenStack Architectural and Design Guide *http://docs.openstack.org/arch-design*. Once you determine the type of OpenStack deployment is right for you the online design guide can be a valuable asset for configuration and sizing.

This chapter is divided into top-level sections, which are related to the intended usage of your OpenStack deployment.

9.1 Replacement of existing virtual server platform(s)

In a December 2013 enterprise survey, Gartner (technology research firm) reported that respondents considered vSphere (VMware) their primary hypervisor, and 48% named Hyper-V (Microsoft) as their secondary hypervisor. While this survey is several years old, chances are you are using one or both of these hypervisors in your environment. This sub-section covers how OpenStack can be used as either a replacement or augmentation of your existing VM environment.

Likely your traditional virtual environment was designed to provide virtual machines in a way that operationally mimics physical machines. There is also a good chance that virtualization was introduced into your environment as an infrastructure cost savings measure for existing workloads. Often as workloads were moved from physical servers to their virtual equivalents, through a process knows as Physical to Virtual (P2V), an exact clone of the physical server was made by the P2V tool. More often than not, physical and virtual servers operated on the same networks, which allowed the P2V migration to take place often without service interruption. For many environments the process of consolidating workloads on virtual servers resulted in significant financial savings. In addition to resources being used more efficiently, new capabilities like master images and virtual machine image snapshots, became part of the software deployment and upgrade processes, mitigating many types of software and hardware failures. Despite many of the new capabilities virtual environments offered users system administrators were still managing both the OS and application levels of virtual machines, much in the same way they had physical machines.

If your intent is to treat your virtual environment the way you do a physical environment, as described in the previous paragraph, then the benefits of OpenStack in your environment will be limited. This is to say, if your operational

practice is the manual deployment of virtual services through a central group without automation, then you must evaluate how cloud frameworks such as OpenStack can be effectively adopted in your environment.

Suppose you have been using VMware vSphere as your server virtualization platform and you are interested in adopting OpenStack as VMware replacement for cost reduction purposes. If you think of OpenStack as simply a "free" alternative to VMware, then you might be heading down the wrong path. While in most cases OpenStack can be deployed in a way that provides feature parity for many competing virtual environments, this has to be done in a way that is compatible with your operational practices. Consider again the previous VMware replacement example, where you want to move all existing VMware workloads to OpenStack. While OpenStack Storage can deal with VMDK (VMware image format) files, there is no graphical Virtual to Virtual (V2V) migration tool like what is provided by VMware, and likely there should not be. Now, consider the process most often used to build VMware-based machine images, where a CD/DVD is virtual mounted by a remote host in order to install a system-image, this to does not exist on the OpenStack Dashboard. One should not consider these things as an indication that OpenStack is incomplete, but as an indication that it is intended to be used differently than traditional virtual server environments.

Where might OpenStack be a good replacement for VMware and other commercial hypervisors? To answer this you must first consider strategically how you want to interact with infrastructure resources. The table below list the possible impact of OpenStack based on your infrastructure manage strategy.

Table 9.1 OpenStack impact based on environment

Name	Description	Impact
Siloed and manual VM	Hardware management is siloed and resources are shared. Virtual hardware is manual provisioned by IT staff to end-users, where end-users are responsible for OS-level operation.	Low to Negative
Siloed and automated VM	Hardware management is siloed and resources are shared. Virtual infrastructure is deployed using automated methods by IT staff, where end-users are responsible for application-level operation.	Medium
Application specific back-end	Hardware is dedicated and managed by the cloud framework. Infrastructure and applications deployment are automated by IT staff for a specific application.	High to Very High
Private Cloud	Hardware is dedicated and managed by the cloud framework. Applications and standard (size and OS) VMs are provided to end-users using automated self-service methods.	Very High

As described in the previous subsection and listed under the previous table as *Siloed and manual VM*, a central group manually provisioning virtual machines without automation, will likely view OpenStack as either incomplete or unnecessary. This is to say, without the addition of automation to their processes, OpenStack does not provide them with anything of value to do their current jobs. Now suppose the central group manually provisioning virtual machines is repositioned as infrastructure and/or application resource consultants, as shown in the table as *Application specific back-end*. Suppose that through this strategic shift not only is automation used in the infrastructure, but also for application-level provisioning request. Suppose further, that resource allocations are allotted to tenants and departmental-level personnel were able to provision their own resources, as shown in the table as *Private cloud*. In this mode of operation the

central group is enabling departmental agility by brokering services, not proscribing them. In many respects operating in this way allows you to change your thinking about the role of infrastructure in your environment. Automation on application and infrastructure levels removes the need for P2V and V2V tools, so there is no need to move images around. In this mode of operation infrastructure resources are more transient functioning more as a application capability than a static allocation.

The real value of OpenStack is in the automation and platform abstractions provided by the framework. The following sub-sections discuss architectural considerations that you must make as you develop your OpenStack design.

9.1.1 Making deployment choices

If you are used to supporting virtual server platforms like VMware vSphere or Hyper-V, you might need to rethink how you purchase and support hardware. While it is less common than in the past, often in the enterprise physical resources like network switches and central pools of storage are shared between physical and virtual resources. Even if a resources is exclusively assigned to a virtual server platform, you typically think of provisioning resources to be used by the platform, not the platform its self managing the resources. For instance, it is common to assign a new VLAN or make a shared LUN available to a group of hypervisors. However, if you need to create a new VLAN or a new shared LUN, the administrator of those systems must go through their own provisioning process. It is also very common that the "network person" do all network configurations, the "storage person" did all storage assignments and the "VM person" tied the resources together with a physical server to produce VMs. Each person in this process had to perform manual provisioning steps along the way, often without visibility into how their resource played into the complete infrastructure.

Deploying OpenStack from slivers of shared central infrastructure is generally the wrong path to take. OpenStack detects, configures, and provisions infrastructure resources, not the other way around. Even if your shared central infrastructure provides multi-tenant (not to be confused with OpenStack tenants) operation, which would isolate OpenStack automation, one would still need to consider the effects of making OpenStack-provisioned resources dependent on shared resources. For instance, software upgrades for reasons outside of OpenStack operations could impact services. In addition, resource utilization outside the scope of OpenStack resources could impact performance, while providing no indication to OpenStack services that problems exist.

In many cases your virtual environments were not designed to leverage the benefits of an infrastructure that can be managed programmatic. In these cases, your operational practices have been developed around vertical management of siloed resources like compute, storage, network, load balancers, etc. However, OpenStack was designed with the complete abstraction of physical infrastructure in mind. In general, you can save yourself lots of trouble by assigning resources exclusively to OpenStack, and through plugins and services let the framework manage resources, instead of the other way around.

In the following sub-sections it is assumed you wish to augment or add new services using OpenStack to manage your resources. In your environment you want to leverage the management capabilities of OpenStack, you even want OpenStack to manage your hardware, but you want the end product to resemble what you are providing now. Specifically, you are willing to change your operational and deployment practices for the sake of efficiency, but your primary interest is to deploy VMs, like what you might be doing now with VMware or Microsoft. Section 9.2 will discuss taking a more progressive approach to IaaS.

WHAT KIND OF NETWORK ARE YOU

If you want to take advantage of OpenStack, but you don't want OpenStack to manage L3 services, such as routing, DHCP, VPN, etc., then you must evaluate your options based on management of L2 (switching) services. The following example shows a VM directly connected to a public L2 network.

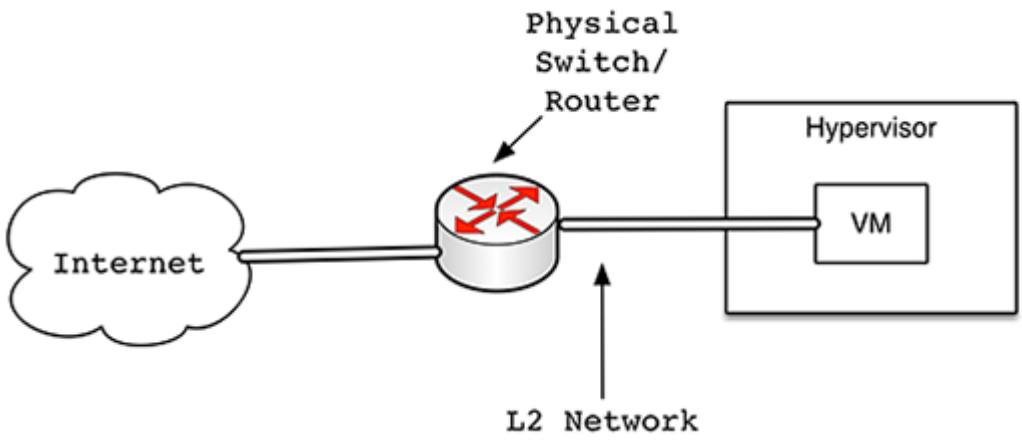


Figure 9.1 L2 network with VM and hypervisor

The example is not specific to OpenStack and would be representative of similar network deployments for many virtual server platforms, including VMware vSphere and Microsoft Hyper-V. In this network deployment scenario, the job of

the hypervisor is to direct L2 network traffic to a switch, which is typically a physical switch outside the control of the hypervisor. Unlike many of the network examples in this book, there is no concept of an "internal" or hypervisor network, since no L2 services are being provided by the virtual server platform. In this deployment type all L3 services are provided by systems outside the hypervisor. As you might imagine, separating the majority of network services from the virtual server platform limits the benefits of the platform, but the simplification is not without benefit. Based on your IT strategy, existing resources, and support structure, this mode of operation might be best for you.

The majority of this book has focused on OpenStack Networking (Neutron) providing L2 and L3 services. As discussed in previous chapters, Neutron was built to mange complex networks and services within OpenStack environments, not simply push L2 traffic to external networks. However, the OpenStack Compute (Nova) project, which predates Neutron, does provide basic L2 services. If you want to limit your OpenStack deployment to L2 services only, then you want to use Nova for networking not Neutron.

SIDEBAR **Network Hardware**

If you are using Nova for networking, there is very little reason to worry about the integration of OpenStack with network hardware. Early on in the OpenStack project, hardware vendors were writing drivers for Nova integration in their hardware, but most development has since moved to Neutron.

Nova network can operate in three different topologies *flat*, *flat-DHCP*, and *VLAN*. In this sub-section the flat topology was described, where all network services are obtained external from OpenStack. The flat-DHCP topology is similar to the flat topology, with the exception that OpenStack provides a DHCP server to assign addresses to VMs. The VLAN topology works in the same way as the flat topology, but allows for VLAN segmentation of the network based on VLAN id. Simply, with the flat network all VM traffic is sent to the same L2 network segment. Using the VLAN topology, you can assign a specific L2 network segment to a particular VM.

The next section will cover choices in storage.

WHAT TYPE OF STORAGE ARE YOU

If you are coming from a traditional virtual server platform environment, you may have no management integration between your hypervisor and storage subsystem. If you are using VMware vSphere, you typically attach large shared host volumes to your hypervisors, as shown in the figure below.

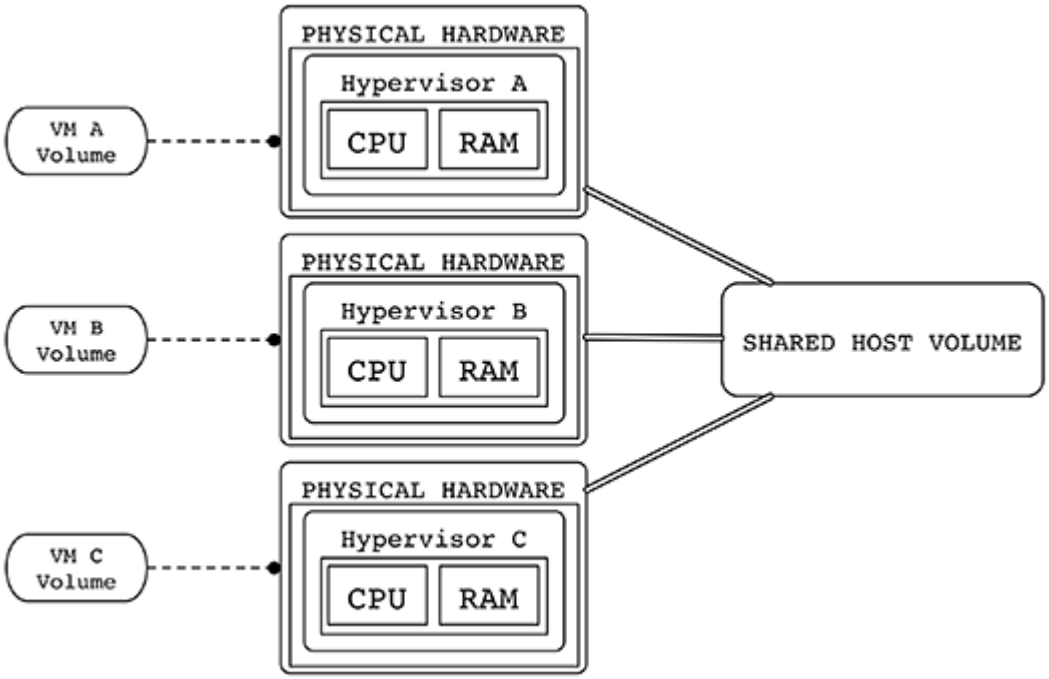


Figure 9.2 Shared volume across hypervisors

In the figure you see a single host volume shared across all hypervisors. These shared host volumes are formatted with a cluster-aware file system, which allows *Hypervisor A* to use the same underlying host volume to store data for *VM A*, as *Hypervisor B* would for *VM B*. If *VM B* was to be migrated to *Hypervisor A*, no storage data would need to be transferred, since the data is already accessible by *Hypervisor A*. In this scenario VM volume management is done on the shared host volume level, so from the standpoint of the underlying storage subsystem nothing is managed beyond the large shared host volume attached to the hypervisors. In contrast, Microsoft Hyper-V promotes a "shared nothing" model, where each hypervisor maintains its own storage and the storage for its VMs. A independent host volume model is shown in the figure below.

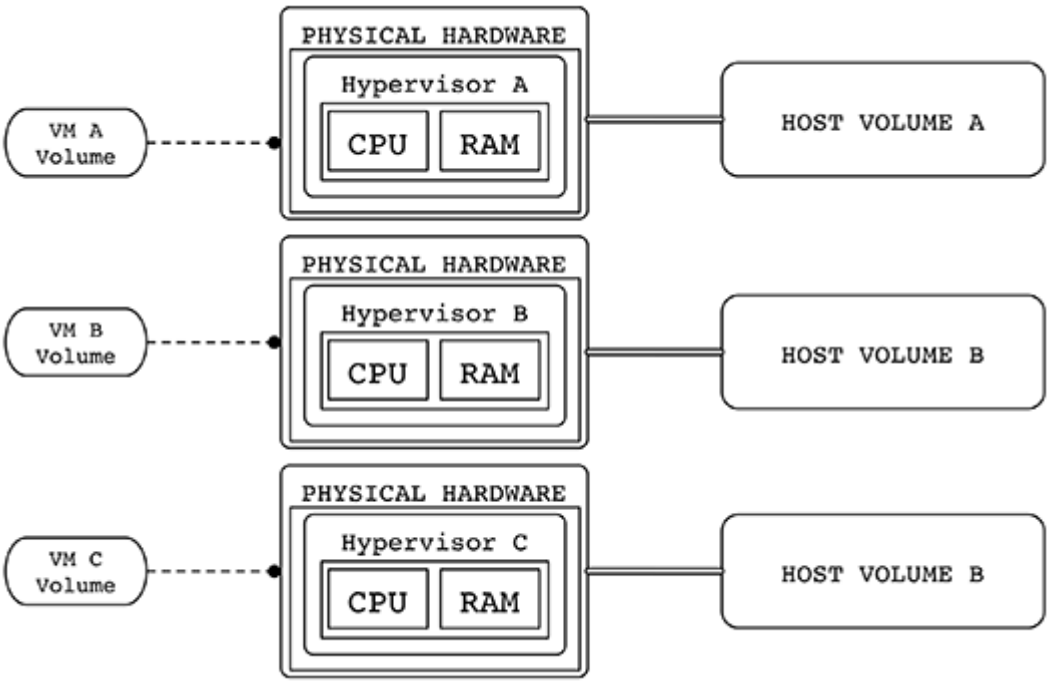


Figure 9.3 Independent host volumes

In the figure, *Hypervisor A* uses *host volume A* to store data for *VM A* . If *VM B* was migrated to *Hypervisor A*, volume information would need to be migrated to the new hypervisor. The benefit of the shared nothing architecture is that the failure domain is reduced, but the migration cost are increased. As with the shared host volume model, the hypervisor is managing volumes for the VMs it maintains, so the storage subsystem is still not actively managed as part of the virtual server platform. This is not to say that there are no storage vendor integrations with vSphere and Hyper-V, simply that a high-level of integration is not fundamental to their operation.

As discussed in previous chapters, there are two types of storage in the OpenStack world, object and block. OpenStack Swift provides object storage services, which can be used to provide back-end storage for VM images and snapshots. If you have been working as a virtual server platform administrator, you might not have ever worked with object storage. While object-based storage is very powerful, is not required to provide virtual infrastructure, and has been outside the scope of this book. However, block storage is a required VM component and has been covered in several chapters.

As with Neutron in previous sub-section, the majority of this book has been devoted to working with block storage using OpenStack Storage (Cinder). Using the OpenStack Compute Service (Nova), it is possible to boot a VM without using

Cinder. However, the volume used to boot the VM is *ephemeral*, which means that when the VM is terminated, data on the VM volume is also removed. In contrast to ephemeral storage is *persistent* storage, which and be detached from a VM and reassigned to another VM. The relationship between the hypervisor, persistent VM volume, Cinder, and the storage subsystem is shown in the figure below.

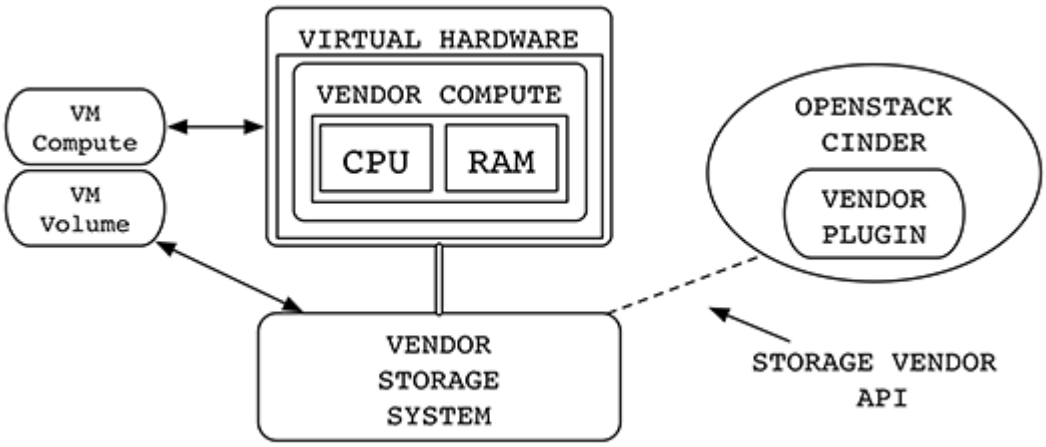


Figure 9.4 OpenStack VM volumes

As shown in the figure, the VM not the host, communicates directly with an underlying storage subsystem. In contrast, both in the case of VMware vSphere and Microsoft Hyper-V, VM storage is provided by the hypervisor. This fundamental difference in operation forces a higher level of storage subsystem management for OpenStack. Where other virtual server platforms might manage VM volumes on the hypervisor level, Cinder interfaces with hardware and software storage systems to provide functions like volume creation, expansion, migration, deletion, etc. A list of storage systems and supported functions can be found on the Cinder support matrix: *https://wiki.openstack.org/wiki/CinderSupportMatrix*. Aside from a few corner cases, most OpenStack deployments will use Cinder to manage volume storage. However, the question remains, what type storage hardware or software platform should Cinder manage?

The question of underlying storage subsystem depends on the intersection of several factors including, what is your current storage vendor(s), are you willing to delicate a storage system to OpenStack, what is your tolerance for risk in your environment. For instance, suppose you wish to mimic the operations of vSphere or Hyper-V, and your storage is provided by a large centralized Storage Area Network (SAN). In this case you might not want Cinder to communicate directly with your shared central system, but you do want to use storage from this system.

In this scenario you can abstract the underlying storage subsystem by attaching independent volumes directly to compute or storage node, similar to what was shown in figure 9.3. You would then use Logical Volume Manager (LVM) to manage your independent host volume. LVM would then be managed by Cinder, thus abstracting the underlying storage subsystem. Managing an independent volume using LVM not invalidate the storage model shown in figure 9.4, in fact LVM was used as the underlying storage subsystem for Cinder in Chapter 7. There are of course other options where centralized storage software and hardware can be used directly, but LVM is a common choice for people using shared central services.

In the next sub-section will cover choices in servers.

WHAT KIND OF SERVER ARE YOU

The past few subsections have covered choices you need to make in providing network and storage resources for your VM. For the most part the choices that you make regarding networking and storage are based on your current and intended future state of operations. In both cases there was no mention of changing the underlying configuration of network or storage hardware and software in a way that was fundamentally different than what you are doing now. In fact, this section has describe an architecture that can be used to mimic a traditional virtual server platform environment. When it comes to OpenStack Compute (Nova) the support matrix (https://wiki.openstack.org/wiki/HypervisorSupportMatrix) does not list server hardware vendors, it list supported hypervisors.

SIDEBAR **Bare-metal and containers**

While outside the scope of this book, using OpenStack to provision bare-metal servers and LXC/Docker containers is also an option. The use of containers in the OpenStack environment is especially interesting for those interested in using OpenStack to provide applications.

Commoditization in the server hardware market around the x86_64 instruction set all but guarantees that if you purchase a server with an Intel or AMD x86_64 processor, any hypervisor will work. While some hardware configurations and vendors provide advantages over one another, OpenStack Compute is hardware agnostic. The real question you are facing is, what hypervisor do I want to use?

You must first consider your motivations for deploying OpenStack in the first place. If your intent is to replace an existing commercial virtual server platform in

a way the mimics the operation of that platform, then you likely don't want to maintain the license cost of the commercial hypervisor.

SIDEBAR **Free versions of VMware ESXi and Microsoft Hyper-V**

Recently VMware and Microsoft have released free versions of their core visualization platforms. The changes in licensing have generated a great deal of community interest in using these hypervisor with OpenStack. However, there are drawbacks including limited initial community support compared to KVM. In addition, hypervisors like ESXi lack the full OpenStack functionality of there commercially licensed versions.

Based on OpenStack user surverys, Kernel-based Virtual Machine (KVM) is the hypervisor used in the vast majority of OpenStack deployments. The majority of the community support will be based around using KVM as your hypervisor. In summary, select server hardware based on current business practices, and use KVM until you have a reason to use something else.

This section covered the architectural decisions around deploying OpenStack to replace an existing virtual server environment. The next section will cover the architecture around deploying a "green field" environment for a On-premise private cloud.

9.2 Why build a private cloud

OpenStack is used in some very large public cloud services including, RackSpace Public Cloud, HP Helion Public Cloud, Dreamhost DreamCompute, and many others. These companies make use of OpenStack projects, along with their own custom integration services, to manage resources on a much larger scale than most enterprise customers. The ratio of servers to administrators wildly varies based on the size and complexity of an organizations infrastructure and related workloads. For example, it is common to have a 30:1 or lower ratio of physical servers to administrators for small and medium-sized business, while a medium to large enterprise might have 500:1 virtual servers to administrator. However, when you consider that the Amazons and Googles of the word achieve a ratios 10000:1 physical servers to administrators, you can start to appreciate the infrastructure management efficiencies that large scale providers have developed. When an enterprise provides public clould-like services from resources that are exclusive to the enterprise, we call this private cloud. By adopting technologies and operational practices born out of large scale providers for private clouds, enterprises are able to develop hybrid cloud strategies based on workload. So, why do private clouds exist? Why are all workloads not in the public cloud? A detailed study of IT strategy concerning public, private, and hybrid clouds is beyond the scope of this book. This section will present several arguments for deploying a private cloud for your enterprise and adopting a hybrid cloud strategy.

9.2.1 Public cloud economy of scale myth

The idea of cloud computing is often described as the computational equivalent of the electrical power grid. Considering the economic definition of utility is the ability of a commodity to satisfy human wants, it is easy to see how cloud computing gained this reputation. Often the cloud is compared to the power grid, but there exist a fundamental flaw in this comparison. The power grid much like cloud computing produces a commodity that must be consumed in real-time, but this is where the comparison ends. The basis for this contradiction is the vast difference between the economies of scale related to production of the commodities. Bulk power generation from nuclear facilities are orders of magnitude more cost effective than what could be produced by a cloud of consumer grade power generators. Given there does not yet exist a quantum or any other type of computer capable of producing computational power with a cost benefit greater than commodity clusters, we must conclude they are not comparable. In fact, the profit margins across commodity servers is so low that enterprise and public provider cost for the same hardware is negligible.

This is not to say that there are not advantages for large scale providers. For instance, a large diverse group of workloads balanced across a resources, should be more efficient than a small scale unoptimized counterpart. However, the point is enterprise customers can take advantage of the same fundamental components as providers, and often near the same price point.

9.2.2 Global scale or tight control

Public cloud providers offer a wide verity of services beyond IaaS, but for the sake of argument restrict your evaluation of public cloud to IaaS. Consider IaaS as providing VMs comprised of discrete components (CPU, RAM, Storage, and Network). Public cloud consumers are unaware of the physical infrastructure used to provide public IaaS offerings. Specifically, users have no way of knowing if they are paying for the latest and greatest or yesterdays technology. To complicate matters further, the consumer has no way to determine the level of oversubscription for a particular type of shared service. Without knowledge of the underlying platform and the number of shared users, there is no passive (quantitative) way to measure IaaS value between public providers. Consider a case where provider A has a cost per unit of X and an over subscription of 20:1, while provider B has a cost per unit of 2X and rate of 10:1, the total cost is clearly the same, but in the eyes of the customer provider B is simply 2X the cost of provider A.

In many industries SLAs (Service Level Agreements) are defined so consumers can evaluate the expected quality of a service provider. Typically, public cloud SLAs are based on uptime not performance. No doubt large consumers of public cloud resources develop performance SLAs with public providers, but this is no common in the small business to medium enterprise space. Without a quantitative approach you are left with qualitative approaches based on active measurement. However, while there is no shortage of benchmarks in the computing industry, there has yet to emerge an accepted workload measurement standard for cloud services.

In the absence of clearly defined SLAs and verification methods, is if difficult to compare the value of public cloud offering between providers. In addition, value comparisons can change over time as workloads change across providers. For many workloads the benefits of global on-demand IaaS, far outweigh resource performance variability. However, for other workloads tightly control performance provided by a private cloud are necessary.

9.2.3 Keeping data gravity private

Dave McCrory coined the term data gravity to describe how applications and other services were drawn to sources of data, analogous to the attraction of physical objects in the universe proportional to their mass. Public cloud providers recognize this phenomenon and typically make it much more financially attractive to move data into their services that out. For instance, there is no charge to move data into an Amazon EC3 service, but there is a tiered pricing structure to move data out of the service. A similar pricing structure exist for Amazon EC2 instances and other IaaS provider offerings, to entice moving data into a specific cloud provider and keeping it there.

Cloud providers can exploit the data gravity phenomena through their transfer rate pricing structure, to create cloud vendor lock on consumers. Consider the case where an organization determines that based on unit price of resources (not accounting for transfer) it is cost effective to move all of its storage and related computation to a public cloud provider. Even if the majority of data was generated outside the cloud provider, there would be no transfer penalty to continuously add data to storage maintained by the public provider. Now suppose this organization wants utilize a secondary public cloud provider for redundancy. While the new provider might not charge transfer for incoming data, the existing provider will charge for outgoing data, which could greatly increase cost. The same hold true if you want to process information locally or if you want to take advantage of processing on a lower cost provider. When the preponderance of your data is maintained in the public cloud, it is hard for services to escape the force of gravity pulling you your data provider. Keeping the majority of your data in a private cloud allows you to only move data in and out of public clouds as needed. For many workloads and organizations the ability to consume services from several providers, including local resources, outweighs the benefits of pure public cloud offerings.

9.2.4 Hybrid moments

The principle of pay-as-you-go is a key differentiator between public and private clouds. It is easy to understand the economic benefits of the spending 1 hour on 1000 computers versus the 1 computer for 1000 hours when timely information is essential. At first glance, the economics of purchasing cloud services does not seem viable when you assume 100% service usage 24x7x365. The idea of usage-based pricing allows for the redirection of otherwise committed capital to be repositioned as strategic investments. The cost of capacity planning risk must also be factored into the equation. Given the natural elasticity of cloud much of the capacity risks are transferred to the cloud provider. Private cloud costs related to over-provisioning are understood as the cost of building for the peak usage regardless of peak duration. In most cases peak workloads exceed average workloads by a factor of 5 to 1. Public cloud has been adopted broadly across the enterprise. However, enterprise adoption of public cloud is rarely an exclusive adoption. Based on enterprise surveys public cloud services are adopted for specific strategic workloads, while private clouds provide services for a more diverse range of services.

The majority of IT organizations have adopted a hybrid cloud strategy, making use of both public and private cloud resources. The real economic benefits of public cloud in the enterprise can be realized if organizations find the right balance between their provide clouds and public cloud.

For the service provider OpenStack can provide project components than can be used to construct massively global clouds of resources. For the enterprise, the OpenStack framework can be used to deploy private cloud services. From an integration standpoint, API compatibility between public and private OpenStack-based providers allows the enterprise to optimally consume resources based on workload requirements.

9.3 Building a private cloud

This book has focused approaching OpenStack from the enterprise prospective, not as a virtual server platform replacement, but as a cloud management framework. The previous section covered the benefits of deploying a private cloud. In addition, the benefits of adopting a hybrid cloud strategy where resources could be managed based on a common (OpenStack API) control set were covered.

This section will tie together what you have learned in previous chapters and will prepare you for the rest of the chapters in Part 3 of the book.

9.3.1 OpenStack deployment tools

The deployment tool you choose will be based on existing vendor relationships, current operational strategy, and future "cloud" direction. There are three approaches you can take when deploying OpenStack. The first approach, a manual deployment, was covered in Part 2 of this book. Manual deployments offer the most flexibility of any options, but have obvious problems at scale. The second approach, general orchestration tools, uses tools like *Ansible*, *Chef*, *Juju*, *Puppet*, *Vagrant*, etc. that are used to deploy a wide range of systems and applications. Being well versed in a collection of general orchestration tools allows you to not only deploy OpenStack, but also deploy applications using OpenStack resources. The drawback of these systems is that each tool has its role to play, so you end up using a wide range of general purpose tools. This is a benefit for the aforementioned reasons, but constitutes a training and operational challenge for the organization adopting this strategy. The third approach, standalone OpenStack deployment and management tool, is a familiar approach for those in the enterprise. OpenStack deployment platforms like *HP Helion*, *Mirantis Fuel*, and *RedHat RDO* not only provide easy to use tools for deploying OpenStack, they provide their own validated OpenStack versions and deployment methods. You can think of this the same way you think of Linux distributions. Like the Linux kernel there is one OpenStack source repository (many branches) for community development. Enhancements and fixes recognized by Linux and OpenStack communities make there way into their respective code repositories. However, just like the Linux community, vendors exist to validate community work for specific use cases, for which they provide support. Just as in you don't technically pay for the Linux kernel when you pay for a supported Linux distribution, you are not paying for OpenStack when you purchase a commercially supported OpenStack distribution. As with many Linux distributions, commercial OpenStack vendors typically provide community supported versions of their deployment tools. Once such tool *Mirantis Fuel*, is covered in Chapter 11.

SIDEBAR **Research, Big Data, and OpenStack**

For those that deal in the areas on research computing consolidated infrastructure management options are emerging. Traditional High Performance Computing (HPC) niche vendors like *Bright Computing* and *StackIQ* (API level) are getting into the Hadoop and OpenStack game. These vendors and likely others are adapting their HPC deployment and management platforms to provide a single pane of glass management across HPC, Hadoop, and OpenStack deployment.

In many IT shops the title *Systems Programmer* is still used to describe roles that have not had much to do with programming since the days of the mainframe. However, In some organizations the System Programmer role has been reborn as part of the *DevOps* (systems administrators that also write code and scripts) movement. A system administrator that is accustomed to manually double-clicking their way through VM and application deployments is not likely going to be comfortable general orchestration tools that they have to code or script together. Likewise, someone with automation experience will feel very restricted will a standalone OpenStack deployment tool.

Based on the strategic direction of your origination you should pick a direction that not only deploys OpenStack, but a direction that is sustainable. For some this direction will be purchasing a commercially supported distribution and assigning existing resources to work with the supporting vendor. Some organizations will choose to develop DevOps teams that will not only be able to deploy OpenStack, they will be able to orchestrate resources and applications across private and public cloud providers.

9.3.2 Networking in your private cloud

In sub-section 9.3.1.1 of this chapter, Nova networking was discussed. When using Nova networking your choice of networking hardware is not of great importance since OpenStack does very little to manage your network. However, throughout most of this book networking is discussed in the context of OpenStack Networking (Neutron). When using Neutron your choice of network hardware and software is very important, since OpenStack will be managing many aspects of your network.

At the time of writing, few (See: https://www.openstack.org/marketplace/drivers/) vendor provided L3 services exist. Since L3 services will likely be provided by OpenStack, the focus of this sub-section will be on choices related to L2.

SIDEBAR **Neutron Distributed Virtual Routing (DVR)**

One of the many goals of the Neutron DVR sub-project is to provide distributed routing with compute nodes, integration with routing hardware, and routing service migration between nodes. While the project is fairly new, the DVR project will likely serve as the primary integration point for most advanced L3 vendor services.

From a Neutron L2 prospective you have two choices. The first choice is to use a community or vendor provided monolithic network plugin. These plugins are considered monolithic since all L2 OpenStack services must be implemented by the driver, as shown in the following figure.

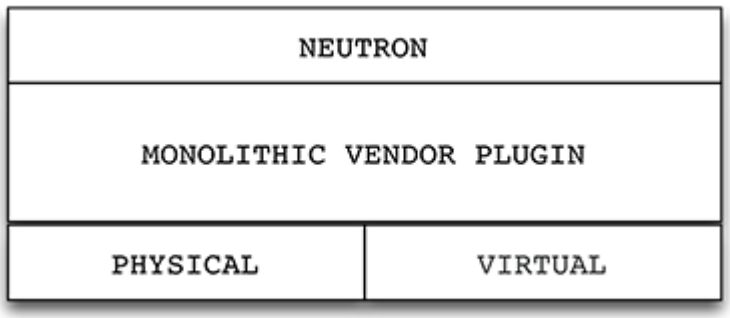


Figure 9.5 Monolithic plugin archatecture

Initially, monolithic plugins were the only method to integrate vendor hardware and software with OpenStack Networking. Several monolithic plugins have been developed for vendor hardware including, Arista, Cisco, Melinox, VMware, and others. The issue with this approach is that plugin code must be modified with subsequent OpenStack releases, even if nothing has changed on the vendor side. Separating out vendor specific code from OpenStack code lead to the second choice in Neutron L2 networking, the Modular Layer 2 (ML2) plugin, previously covered in Chapter 6 and shown in the figure below.

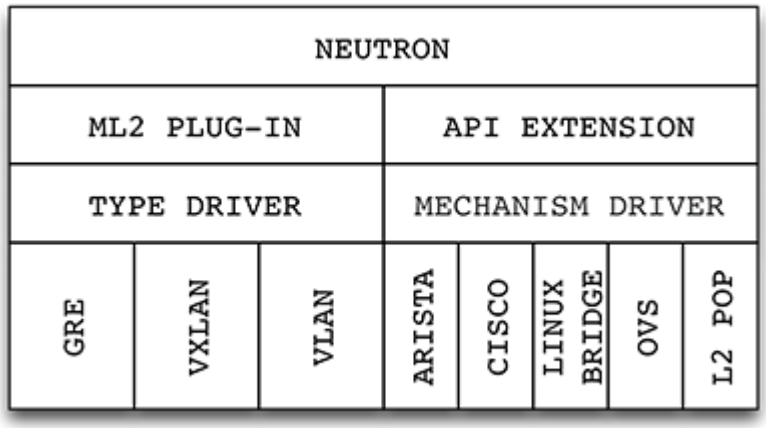


Figure 9.6 ML2 plugin archatecture

The ML2 plugin framework allows communities and vendors to provide L2 support much easier than with monolithic plugins. The majority of vendors, even those that had previously developed monolithic plugins, are now adopting the ML2 plugin by writing *Mechanism Drivers* for their specific technology.

So why should you care about vendor support of the network anyway? Based on the OpenStack user survey *Open vSwitch (OVS)* is the most commonly used network driver (interface used between standalone hardware and software package and OpenStack) used in OpenStack deployments. Due to it's popularity OVS was used as a network driver in both Part 1 and 2 of this book. Specifically, using ML2 terminology, the ML2 plugin was configured to use the *GRE Type Driver* and *OVS Mechanism Driver*. By using a combination of an overlay (GRE) type driver and a software switch (OVS), we simplified the switch hardware configuration down to simple connectivity between compute and network nodes. The hardware configuration is simple in this context, since OVS is using the physical transport as a method to communicate between other OVS switches using network overlays (traffic isolation on OVS level). There are many benefits to using network overlays like *GRE* and *VXLAN*, including scale and flexibility. However, there are performance cost related to using network overlays and software switches (OVS) in general. In Chapter 11, *Automated HA OpenStack Deployment with Fuel*, the *VLAN Type Driver* will be used with the *OVS Mechanism Driver*. This approach requires more configuration on the side of the switch hardware. In this context OVS is still the network driver, but instead of overlays connecting OVS instances, OVS makes use of a range of VLANS. The VLANS used by OVS in the Chapter 11 example must be manually configured on the switch. In this context, some switching load is offloaded to the hardware switch and some remains in OVS. The

next progression in offloading network load from software to hardware is to use a mechanism driver that deals with all L2 operations on the hardware (this could be a hybrid hardware and software) device. In this configuration OpenStack Network operations are translated by the network driver into vendor specific operations. This does not mean that you have to use VLAN as you type driver when using a hardware vendor mechanism driver. In fact, there are many vendor managed types including very powerful VXLAN type drivers that are offloaded in hardware.

As with most things in OpenStack, and technology in general, generalize solutions (in software) come at the cost of performance (through hardware). You must determine if the performance of software switching and overlay is acceptable or can your private cloud benefit from the performance gained through tight integration of OpenStack Networking and vendor hardware.

9.3.3 Storage in your private cloud

In Chapter 7, you walked through the deployment of OpenStack Storage node using Cinder. The purpose of the storage node was to provide block storage to VMs. Just as Neutron uses network drivers to communicate with underlying software and hardware network resources, Cinder uses storage drivers (See: https://www.openstack.org/marketplace/drivers/) to communicate with storage resources.

In Chapter 7, a LVM configured volume was managed by Cinder. In this context, a LVM storage driver was used by Cinder to interface with the underlying LVM subsystem. Now, where did the storage device that was used by the LVM volume come from? As previously discussed in sub-section 9.3.1.2 of this chapter, the LVM volume could have been a local disk or it could have been provided from an external source like a SAN. From the standpoint of LVM and by relation Cinder, as long as the device shows up as a block storage device in the Linux kernel, it can be used. However, this is similar to OVS using network hard as a physical transport. Through the abstraction of the underlying storage device by LVM, you are loosing many of the advanced storage features the underlying storage subsystem might offer. Just as with OVS, OpenStack is unaware of the capabilities of the underlying physical infrastructure and storage functions are offloaded to software. Luckly, there are many Cinder storage drivers for OpenStack including drivers for storage systems provided by Ceph, Dell, EMC, Fujitsu, Hitachi, HP, IBM, and many others. As with OpenStack Networking,

integrating OpenStack Storage with hardware and software storage subsystems through the use of vendor storage drivers allows OpenStack to make use of the advanced features of the underlying system.

Based on OpenStack user surveys, the Ceph storage system is used in the major of OpenStack deployments. Due to its popularity in the OpenStack community and inclusion in many standalone OpenStack deployment tools, Chapter 10 is dedicated to walking through a Ceph deployment.

Much as with vendor decisions related to OpenStack networking, storage decisions need to be based on your current capabilities and future direction. While extremely popular with the OpenStack community, building out support for a Ceph storage cluster might not be the right choice if the rest of the storage in your enterprise is EMC. Likewise, with many advanced storage features previously only found in high-end arrays, now found in Cinder or not needed because of some other aspect of private cloud operation, purchasing a high-end array might not be necessary.

As you continue on through the remaining chapters in Part 3 of the book, think about the type of environment you want to construct. For some a purpose built system with deep vendor integration will be the best fit. For others a general purpose deployment that is flexible is the right choice. Regardless of the path you take make sure OpenStack is the right tool for the job, and most importantly your origination is well position to take advantage of the benefits of the OpenStack framework.

11 *Automated HA OpenStack Deployment with Fuel*

This section covers

- Preparing your environment for Fuel
- Deploying Fuel server
- Deploying OpenStack using Fuel

This chapter will demonstrate an *automated high-availability (HA)* deployment of OpenStack using *Fuel*.

The deployment type is described as automated since, you will prepare your hardware for automated deployment, describe your environment, and the automation tool will perform all steps required to deploy OpenStack in your environment, including the deployment of OpenStack components as covered in Chapters 5-8 and Ceph, covered in Chapter 10. The term HA refers to the implemented architectural design, where multiple OpenStack controllers are used.

In Chapter 2, *Taking an OpenStack TestDrive*, you were introduced to the automation tool *DevStack*. Recall that this tool performed automation tasks related to OpenStack deployment, but was intended as a development tool, not for deploying production environments. A production-focused automation tool must provide more than simply configuration and installation of OpenStack, it must also deal with environment preparations, like OS installation, and server-side network configurations. In fact, while the tool demonstrated in this chapter does not go this low in the stack, some automation tools will actually configure network hardware as well. a production-focused automated tool must be auditable, repeatable, stable, and provide the option of commercial support.

The term HA refers to the ability of the deployment to continue operation even when specific components fail. So far the deployments described in Chapter 2 and

Part 2 of this book, covered deployments with a single controller. In these type of deployments if the controller server or one of its dependencies (MySQL DB) fails, your OpenStack deployment has failed. Specifically, until the single controller is returned to operation, no changes could be made to your infrastructure. In the HA deployment demonstrated in this chapter, the deployment will continue to operate as normal, even if a controller is disabled. Controllers are aware of each others state, so if a controller failure is detected, services are redirected to another controller.

The tool that will be demonstrated for OpenStack deployment in this chapter is called Fuel. Fuel was developed and later open-sourced by Mirantis Inc. *https://www.mirantis.com*. Several production OpenStack automation tools exist, and many more are in active development. Fuel was chosen for demonstration based on the maturity of the tool, number of production enterprise deployments, and the availability of commercial support. While this chapter will demonstrate a HA deployment of OpenStack using Fuel, many of the steps would be the same regardless of tool.

First you will prepare your environment, next you will deploy the Fuel tool, and finally you will use Fuel to deploy your OpenStack environment.

11.1 Preparing your Environment

Claims of "turn-key" automation typically provide a vision of a car buying experience. You sign on the dotted line, the sales person tosses you the keys, and free as a bird you set out on the open road. The problem with this vision is that when is IT ever "turn-key," at least if you are the one responsible for doing the work. Keeping with the car analogy, our current-state technology car is more like a custom-built pre-assemblyline vehicle. Some might run on electric, others on fossil fuels, or even steam. The point is, no standard assembly-line deployment of OpenStack exists. As a consequence, no standard support model exists across the OpenStack domain for the deployment of the framework. For example, a Windows or Linux administrator on at least a basic level, will understand a Windows or Linux instance installed anywhere. The rules of these systems apply universally. These universal terms, from a deployment standpoint, do not exist for OpenStack the "Cloud Operating System". This of course is the intended benefit of this book: Understand the framework well enough, so that when things fail, or when automation tools evolve, you have some idea of what is going on under the covers.

The preparation of an automated deployment environment and configuration might well take longer than deploying a small environment manually. However, it

is in the opinion of the author that for the enterprise, the use of automation tools even in small deployments is very helpful. Helpful, in that the tasks you completed in Part 2 of the book can be repeated, support can be purchased, and actions can be tracked and if necessary audited. In this chapter you will walk through the process of an automated HA OpenStack deployment.

11.1.1 Network Hardware

As you might have experienced in Chapter 2 and Chapter 6, the network component of OpenStack can be the most confusing and difficult. Admittedly, there are many moving pieces to manage and configure to use OpenStack Networking. This difficulty is amplified if you are limited in your operational understanding of router and switch configuration. Even if you have a complete understanding of routing and switching, you might not have direct access to network hardware to make changes. Even if you have direct access to the network hardware for your OpenStack deployment, you might not have the ability to assign your own address and Virtual Local Area Networks (VLANs) or configure up-stream network hardware. As you work though this chapter, you will need the ability to do these things, or have them done for you.

NOTE **VLANS Untagged vs Tagged**

The IEEE 802.1Q networking standard provided the ability to designate a virtual network by adding information to the Ethernet frame. Virtual Local Area Networks VLANS are considered a OSI L2 function and allow administrators to do things like divide up a switch into separate L2 networks and assign multiple VLANS to a single physical trunk interface. When it is said that a VLAN is *tagged*, the Ethernet frame contains at 802.1Q header specifying its VLAN. Likewise, when a Ethernet frame does not contain a 802.1Q header ,it is said to be *untagged*. It is common for switches to communicate between each other through tagged frames (many VLAN same port), while generally servers use untagged frames (one VLAN per port).

As you learned in Chapter 6, in OpenStack your server is acting like a switch, and like a switch could be using tagged or untagged VLANs. It's important you understand this concept as you move forward. The examples in this book use tagged VLANs on both the physical switch and physical network interfaces on the server.

CONFIGURING DEPLOYMENT NETWORK

You want to accomplish three things in this section:

- You need to make sure that you can communicate with your automation (Fuel) server.
- The automation server can contact all host servers.
- You have out-of-band (OOB) access to both your automation and host servers.

In the deployment demonstrated in this chapter makes use of two separate physical networks: the automation administration network and OOB network. The administration network is used by the automation system to manage host servers on operating system and OpenStack levels. The OOB network is used to access and configure servers on the hardware level.

SIDEBAR **Love your OOB as you love yourself**

Working with OpenStack or any large system without OOB is like working on an aircraft engine while in the air. The importance of OOB for automated deployments can't be overstated. You might be able to run between consoles in your existing environments or even in a manual OpenStack deployment. However, the processes used in a automated deployment all but guarantee that you must have access to your servers from outside of an OS level. There is no greater fear in an admins heart than the loss of access to OOB, this means a trip to the data center.

Just how these networks will be used will be discuss in the following subsections, but for now it is sufficient to know the example will use two untagged (VLANs) networks on two separate network interfaces for administrative and OOB networks, for each server.

The switch interface and VLAN configuration for the demonstration OpenStack management switch, a *Force10 S60*, is shown in the example below.

Listing 11.1 Out of Band and Administration

```
interface GigabitEthernet 0/1                                    ❶
 description "Uplink Port VLAN 95,96"
 no ip address
 switchport
 no shutdown
!
interface GigabitEthernet 0/2                                    ❹
 description "OOB Server 0"
 no ip address
 switchport
 no shutdown
!
interface GigabitEthernet 0/3                                    ❺
 description "Admin Server 0"
 no ip address
 switchport
 no shutdown
!
...
interface Vlan 95                                                ❷
 description "OOB Network 10.33.1.0/24"
 no ip address
 tagged GigabitEthernet 0/1
 untagged GigabitEthernet 0/2
 no shutdown
!
interface Vlan 96                                                ❸
 description "Admin Network 10.33.2.0/24"
 no ip address
 tagged GigabitEthernet 0/1
 untagged GigabitEthernet 0/3
 no shutdown
!
...
```

CONFIGURING SWITCH UPLINK PORTS

In order to actually use you management switch you will need to configure an "uplink" to an existing network. The physical network topology for the demonstration deployment is shown in the figure below.

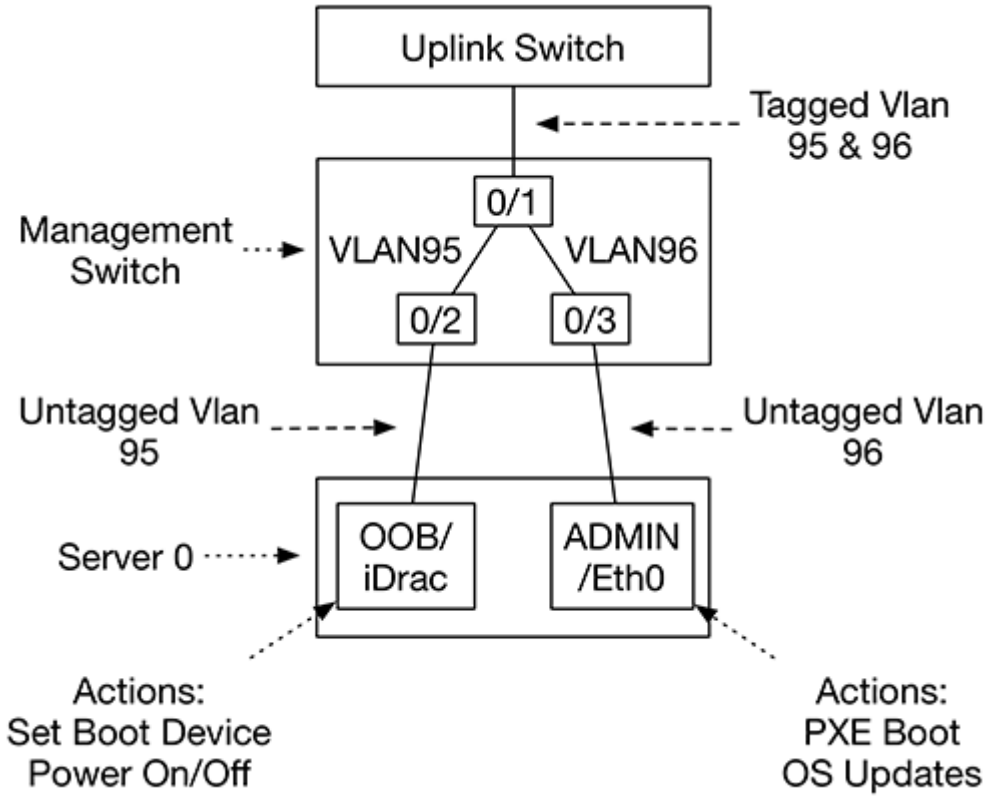


Figure 11.1 Network Hardware Topology for Single Host Management

The network topology shown in the previous figure, shows the relation between the uplink and management switches and a single OpenStack host. In the example, port 1❶ contains the two VLANS 95❷ and 96❸. You will notice that these VLANs are tagged, which allows them to both exist on the single port 1. Likewise, ports 2❹ and 3❺, which are destined for the server are untagged. The server ports are untagged, because there will be only a single VLAN assigned per port. In addition, low-level functions related automated deployment are often complicated or impossible using tagged VLANs, since the software or hardware used in the deployment process might not support VLAN tagging. It's worth noting that the described networks will not be used directly by OpenStack, but only in the deployment and management of the underlying hardware and OS. These networks will continued to be used for system administrative purposes.

Of course there will be much more network configuration going forward specific to OpenStack Networking (networking that OpenStack controls), but this will be sufficient to get started. The next sub-section will discuss preparations that must be made to the hardware.

11.1.2 Server Hardware

Earlier in this section a network topology was described that would provide OOB and automation administration for each server, both participating in the OpenStack deployment and the automated deployment (Fuel) server.

At this point every server has two physical cables plugged into a management switch(s). One cable is for the OOB and the other is for administration, as previously shown in Figure 11.1. In this sub-section we will prepare the server hardware to use the two networks.

CONFIGURE THE OOB NETWORK

From a deployment and ongoing management standpoint the OOB network is critical. OOB capabilities include:

- Management of software-configurable aspects of server and network hardware.
- Remotely access virtual representation of the hardware console.
- Remotely mount virtual media to be used in software installation.
- Programmatic access to hardware operations (script reboot, boot devices, etc.)

The assumption this point is that your servers are unconfigured ,but have been placed in the rack, have physical connections to the previously described networks, and have power. Your next step is to establish OOB connectivity to all your servers. OOB management can be though of as your lifeline to the physical hardware. This interface is separate from the operating system and often a physically separate add-on device.

There are several ways to accomplish the next step, some automated, and others quite manual. In some cases, initial configuration tasks might be performed by data center operations, in other situations this falls to the systems administrator. Likely, the size of your deployment and your enterprise operating policy will be determining factors into which process is good for you.

Typically, establishing OOB connectivity is done only once and changes are very infrequent, so unless you have an established method for automating this process, it might be easier to manually configure OOB on each server. Better yet, find an intern and ask them to do it.

WARNING	**Automation: Measure once cut twice** The process of racking, stacking, cabling, and most importantly documenting the physical environment is extremely important. Likely for many of you this will be well understood through painful experiences like: "What do you mean the rack/row was on the same feed/network/unit". Through automation we can easily deploy very complicated configurations across large environments. This effectively creates a configuration "needle in a haystack," when underling infrastructure is not properly configured. The physical deployment and documentation is the basis for your system, so make sure everything is *exactly* as it is documented before proceeding. The upfront rigor will be rewarded in the end.

To manually configure OOB, you must physically access your server hardware console with monitor and keyboard (optionally a serial interface). Typically a system configuration screen can be accessed by interrupting the server boot process, by pressing designated keys when prompted.

The figure below shows the system configuration screen related to OOB management. The demonstration system is a *Dell* server, which contain *iDRAC* OOB management cards. While the OOB management cards are vendor specific, the general configuration and desired result will be the same across vendors. In the figure, you can see where a *static* address, gateway, and subnet mask have been assigned. Once saved this information will persist across reboots.

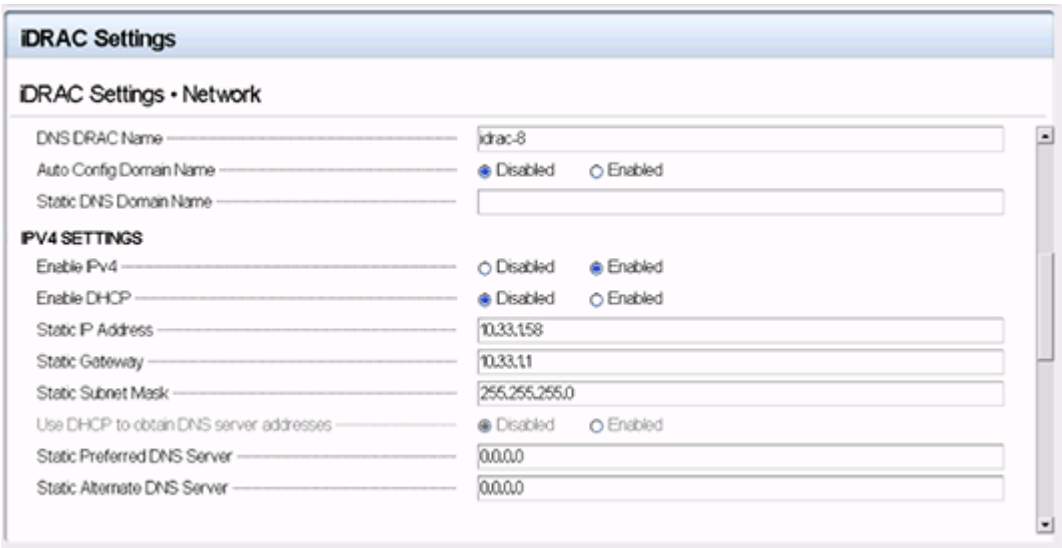


Figure 11.2 OOB Management Interface

NOTE	**DHCP and OOB Management** Dynamic Host Configuration Protocol (DHCP) can be used to configure OOB management. While beyond the scope of this book, in larger deployments this additional degree of automation is all but necessary.

Once your OOB has been configured, you'll likely have several methods to access your server including, Web interfaces and Secure Shells (SSH) are discussed next.

ACCESS AN OOB WEB INTERFACE

The figure below shows the OOB web interface for the demonstration server.

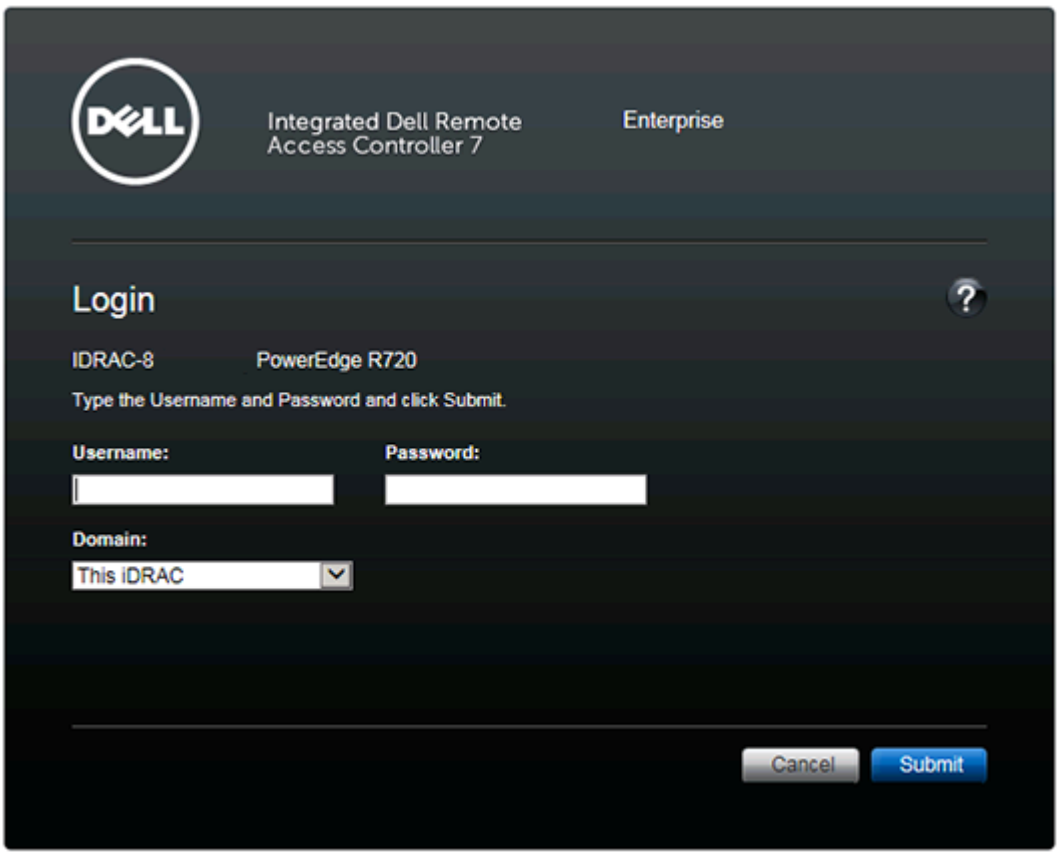


Figure 11.3 OOB Management Interface

From the web interface you are able to access the virtual console and mount virtual media. The virtual console will displays exactly what you would see on the physical console if you were standing in front of the server.

OOB web interfaces are notorious for requiring specific and typically out of date web browsers. The good news is that most of the functions you are likely

interested in from the OOB prospective can be configured through an OOB SSH console.

ACCESSING OOB USING SSH

In addition to the graphical web interface, OOB management systems typically include a SSH and/or Telnet interface. While the web-based interface is convenient, it is not easily used from a programmatic prospective. The *iDRAC* in the demonstration system provides an SSH interface, which allows for the scripted manipulation of OOB managed hosts. These manipulations range from a simple reboot, to complete hardware reconfigurations. For example, for each type of server (compute, storage, etc.) you can create a scripted hardware level (from system software level, RAID, etc.) configuration. Your role specific configuration can then be applied through your OOB management interface.

Making use of the non-interactive ssh password provider software *sshpass*, the example below demonstrates the process of configuring hardware over SSH. The application *sshpass*, allows you to script access

Listing 11.2 Interface OOB Management over SSH

```
sshpass -p 'mypassword' ssh -o StrictHostKeyChecking=no \
 root@10.33.1.58 racadm config -g cfgServerInfo \
 -o cfgServerBootOnce 0                                    (1)

sshpass -p 'mypassword' ssh -o StrictHostKeyChecking=no \
 root@10.33.1.58 racadm config -g cfgServerInfo \
 -o cfgServerFirstBootDevice PXE                           (2)

sshpass -p 'mypassword' ssh -o StrictHostKeyChecking=no \
root@10.33.1.58 racadm serveraction powercycle             (3)
```

WARNING **Scripted Passwords**

In general, keeping clear-text passwords in scripts and using them with SSH is considered bad practice. Obviously, low-level credentials can obviously gained if the script was accessed, but this is also the case in other forms of authentication. Automated login of SSH is designed to use public/private key encryption. An unintended consequence of scripting SSH access could be passwords showing up in console history and other places where interactive login information is typically redacted.

This being said, your ability to use public/private keys and other forms of authentication will be limited by your operating platform.

In the example, three separate commands are sent to the OOB management interface that was configured in the previous figure. The command-set for your OOB management interface will vary by vendor. In the *iDRAC* demonstration, the command *racadm* is used in both the configuration (*-g cfgServerInfo*) and management actions (*serveraction*) The first command ❶ specifies that the following boot configuration is intended as a permanent configuration. The second command ❷ specifies the first devices to be used in the boot process. The final command ❸ restarts the server.

NOTE

Configuring First Boot Device as Network

Automation frameworks must have a way to discover new devices. Discovery is typically done through the network using Preboot eXecution Environment (PXE) booting. Servers configured to *PXE boot* will try and boot using the network before accessing any operating system component that might be found on attached storage.

The PXE boot device will transmit a DHCP request and the DHCP server will return an address assignment along with a location of executable code that can be used to boot the server. Once the DHCP address has been assigned to the server, the boot code will be transmitted over the network and the server will boot based on this information.

The automated deployment described in this chapter makes use of PXE booting.

WARNING

Automation Bug: Network

You might have several devices that are capable of PXE boot on your server. In addition, the server management software might have a PXE boot option as well. In the example environment the PXE boot agent provided by the system-level Unified Extensible Firmware Interface (UEFI) was disabled and the PXE boot agent from the network device itself was used.

This was done due to PXE boot incompatibilities between the Fuel PXE boot environment and the PXE agent provided by the server software.

In the following sub-section you will cover configuring your storage hardware.

CONFIGURING SERVER STORAGE

While private cloud technologies like OpenStack, have started to change things in the enterprise, the norm for a typical server is still to use centralized storage through the use of proprietary Software Area Networks (SAN). There is nothing wrong with this and as previously discussed in Chapter 9, OpenStack can easily make use of SANs provided many vendors. However, many OpenStack strategies make use of open-source server-based storage solutions, like Ceph, described in Chapter 10. In fact, based on OpenStack community surveys, server-based open-source storage solutions are used in the majority of OpenStack deployments. Likely as a result, most automation frameworks natively support open-source storage solutions as part of a fully automated deployment. For these reasons, this book will cover an automated deployment that makes use of the open-source storage software Ceph.

SIDEBAR **Local storage on compute nodes**

Regardless of the role of your server, you will need to configure its internal storage. While it is possible to PXE boot servers, often PXE boot is only used during deployment and upgrade phases. Unless part of your overall operational strategy, you likely don't want to rely on PXE boot availability for normal operation.

Local disk configuration will be specific to your hardware and the intended purpose of your deployment. However, there are some role-based recommendations that can be made based on the demonstrated automation framework and environment.

- Controllers (3 nodes): The framework (Fuel) used in this chapter places the majority of the administrative and some of the operational (MySQL, Network functions, Storage Monitor, etc.) load on the controllers. For this reason servers designated as controllers should have fast system volumes (OS), if possible SSD disks. In HA environment performance should be valued (for many environments) over redundancy for controllers, since you already have redundancy through duplication of your controllers. A SSD RAID-0 system volume will be used in the example.
- Compute (5 nodes): Your VM storage will be located on separate Storage nodes, so once again you only have to worry about the system volume. However, since the system volume is required to provide the operating environment for the virtual servers, it should be resilient. Unless you plan on an environment where RAM is expected to be highly over-provisioned (RAM swapping pages to disk), the system volumes to not require SSD performance. A SAS RAID-10 system volume will be used in the example.
- Storage (3 nodes): Your VM storage, images, and all storage related to resources provided from OpenStack will be stored here. As previously stated, Ceph will be used to manage these resources, but they must devices must first be provisioned on the device

level. Ceph will work with storage on the device level, as presented by the hosting OS. Of course, you must have an OS in the first place, so a redundant system volume is needed. Disk used by Ceph can be separated into *Journal* and *Data* volumes. You want your fastest disk as *Journal* volumes and your largest disk as *Data* volumes. In both cases Ceph volumes should be configured as Just A Bunch Of Disk (JBOD) or RAID-0 if your disk controller does not support JBOD. A SAS RAID-10 system volume was used, along with 4xSSD RAID-0 *Journal*, and 16XSAS RAID-0 *Data* volumes.

WARNING **Automation Bug: Disks**

In the example environment the *Data* volumes on the Storage nodes were combined in pairs to limit the total number of storage devices on the server to 13.

The servers actually contain 24 disk devices, which if you list device and path for all disks creates a long string. Early deployment attempts were failing when 24 disk were used. Failure was due to a string size limited that was exceeded during the automated OS deployment phase, related to the OS itself, not the automation framework. Reducing the number of volumes reduced the configuration string down to an acceptable length.

You might wonder: "What about the automation server?" Well, it really does not matter. Technically, the automation server could be run from a VM or even a laptop. In practice, as will be discussed in the next section, you will likely want a physical automation server, since this server will be both booting (at least initially) your host servers and maintaining their configuration. In addition, you don't want to spread your failure domain if you must use another system to provide a virtual administration node. Performance is less of an issue on admin nodes, but you do want something that can function independent from any other system.

The next sub-section covers the Automation/Administration Network, which will be used to deploy and manage your cluster.

CONFIGURE THE AUTOMATION/ADMINISTRATION NETWORK

From a deployment and management standpoint, the automation/administration network is second only to the OOB network, in terms of importance. This network will be used as the way the automation framework communicates with your hosts. This network will be used for the following functions:

- PXE boot network used during install/upgrade.
- Administrative traffic between Automation (Fuel) server and managed nodes.
- OS level network communication such as, outgoing Network Time Protocol (NTP) and incoming SSHD traffic on managed hosts.

The good news is that like the OOB network, the configuration of this network should be simple. On each server in to be used in the deployment, pick a OS accessible (not a OOB hardware) interface and assign them to the same network. Do yourself a favor and select the same network interface for all servers. Using the same interface for all servers, allows you to group all the servers together when you configure the interfaces in the automation framework. In the example environment, the first on-board network interface (*eth0*) will be designated as the automation/administration network interface.

Assignment, in this context, means the designated physical server ports will be connected to switch ports that have been configured with the untagged VLAN 96. These switch ports will not be assigned any other VLANs and the traffic for VLAN 96 will not contain VLAN tags, as previously described. From the prospective of the end-point device (server) VLAN 96 does not exist, and is only used by the switch to isolated traffic to ports internally designated on VLAN 96. The figure below shows how this network is used during the PXE boot process.

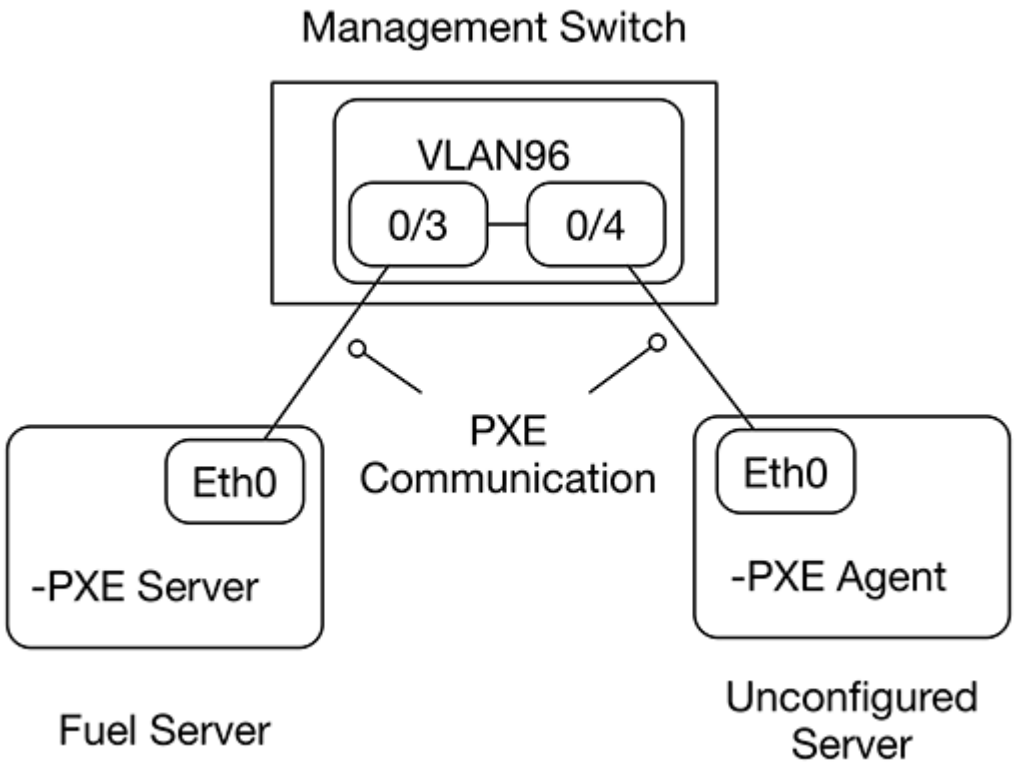


Figure 11.4 Automation/Administration Network in PXE boot

In the figure you see a configured automation (Fuel) server and a unconfigured server on the same untagged network. The *Fuel Server* boots from its local disk

and starts listening for DHCP/PXE boot request on its *Eth0* interface. The node labeled *Unconfigured Server* is a server that has at least been configured with an OOB network (not shown) and set to PXE boot. During the PXE boot process the *Unconfigured Server* will broadcast a request for boot information. Since both servers are on the same network, the *Fuel Server* will receive the request. The *Fuel Server* will then respond to the broadcast with both network address information and additional network booting instructions. The *Unconfigured Server* will then precede with the boot process. The figure below shows how the network is used once the server has been deployed.

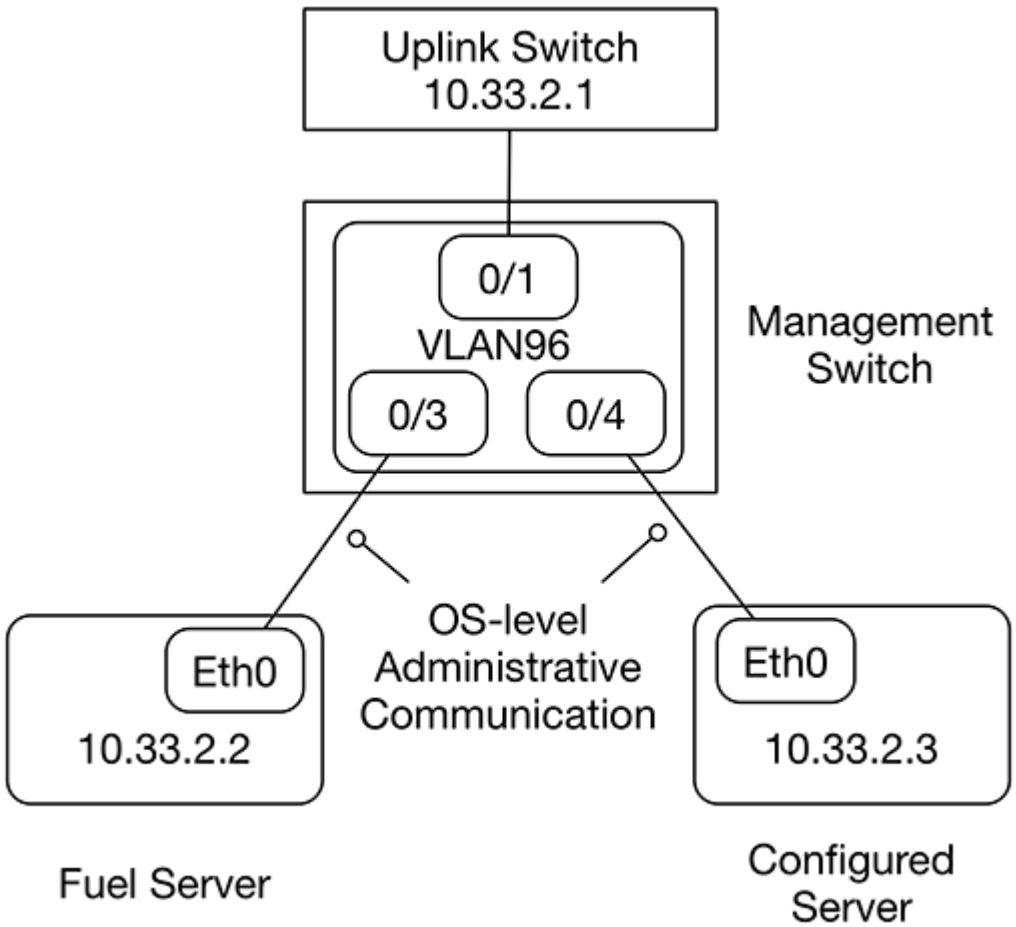


Figure 11.5 Automation/Administration Network in OS Communication

As you can see in the figure, the same network that was used in the PXE boot process will be used for OS-level administration, once the node has been deployed. In addition, Fuel server will continue to use this network to both discover new servers and configuration management on existing servers.

You've now have completed all the necessary steps to prepare the environment

for the automation server deployment. The next section will focus on the deployment of the automation (Fuel) server.

11.2 Deploying Fuel

Ok. You are now setup with Fuel, but before you start the Fuel deployment process make sure that you have OOB connectivity to all servers, including the designated automation (Fuel) server. In addition, make sure that all servers have been assigned an untagged automation/administration network to the same interface, such as *eth0* in the previous examples.

Go ahead and download the Fuel 6.0 community edition ISO from the Fuel Wiki, and start the installation process, described in the next sub-section.

11.2.1 Installing Fuel

At this point you should have OOB or direct console connectivity to the server designated as your automation server, as previously shown in Figure 11.1. Follow the steps below to start the Fuel installation process.

- Using the OOB management capabilities of on your deployment server (or manually), mount the Fuel 6.0 ISO on your designated automation server. This process will be specific to your OOB management tool.
- Reboot your automation server, and based on the vendor-specific instructions for your server, boot from the Fuel 6.0 ISO.
- As the server starts to boot from the mounted Fuel 6.0 ISO, repeatedly press the tab key to interrupt the boot process. To be clear, you want to boot from the ISO, but you want to interrupt the boot loader process on the ISO. If the server continues to boot, repeat the process until you see the screen shown in the figure below.

SIDEBAR **Why are we stopping the boot process?**
We are stopping the boot process in order to make modifications to Fuel installation settings. The next version of Fuel 6.1, currently in development, provides a more convenient installation settings menu.

If you successfully stopped the boot process, you will see the figure like the one shown below.

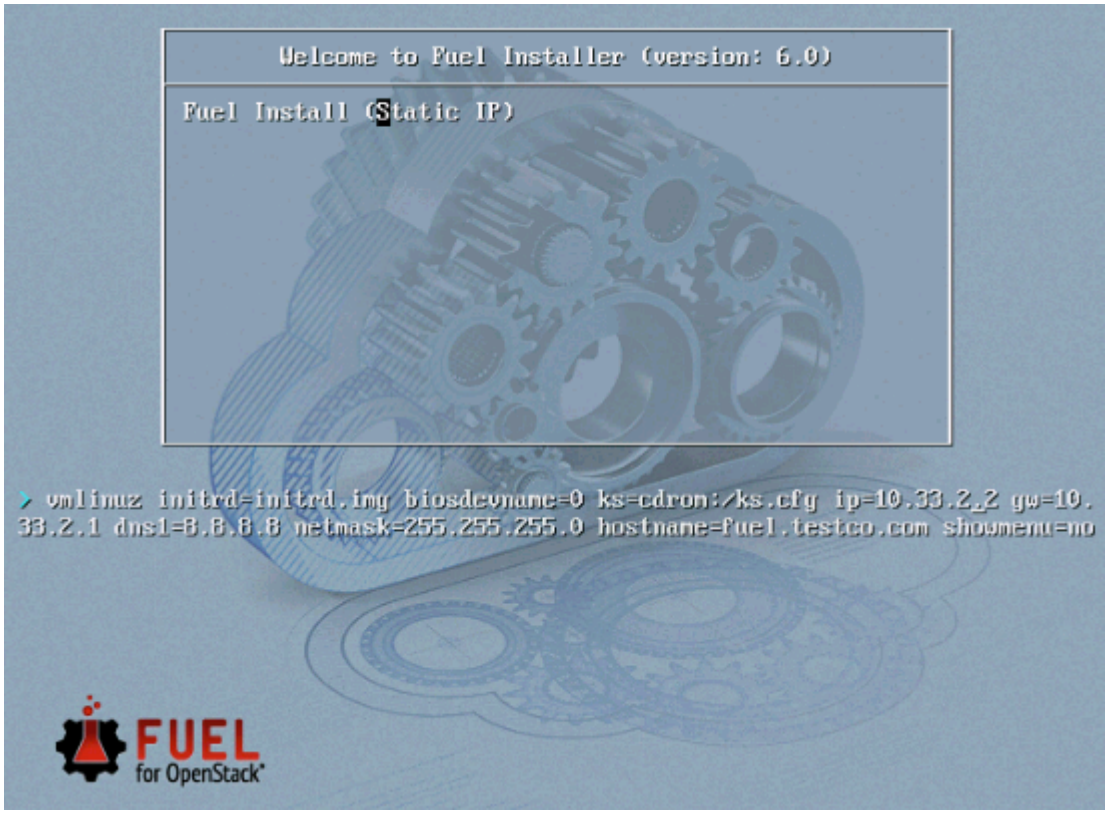


Figure 11.6 Edit Settings on the Fuel Installer Screen

You'll notice that the network settings on your screen are not the same as the one shown in the figure. My default Fuel uses an automation/administration address range of 10.20.0.0/24. The address range for the demonstration environment is 10.33.2.0/24. In order to use a different range, you must use your cursor to modify the following settings to match your environment:

- ip= The address to be used by the Fuel server for both PXE and administration.
- gw= The network gateway address to be used by both the Fuel server and all host managed hosts.
- dns1= The name server address to be used by both the Fuel server and all host managed hosts.
- netmask= The subnet mask to be used by both the Fuel server and all host managed hosts.
  hostname= The hostname to be used by the Fuel server.

When you've made all necessary changes, press the <*Enter*> key to continue the installation. If the Fuel installer detects existing partitions on the local disk you will be prompted to overwrite the partition, as shown in the figure below:

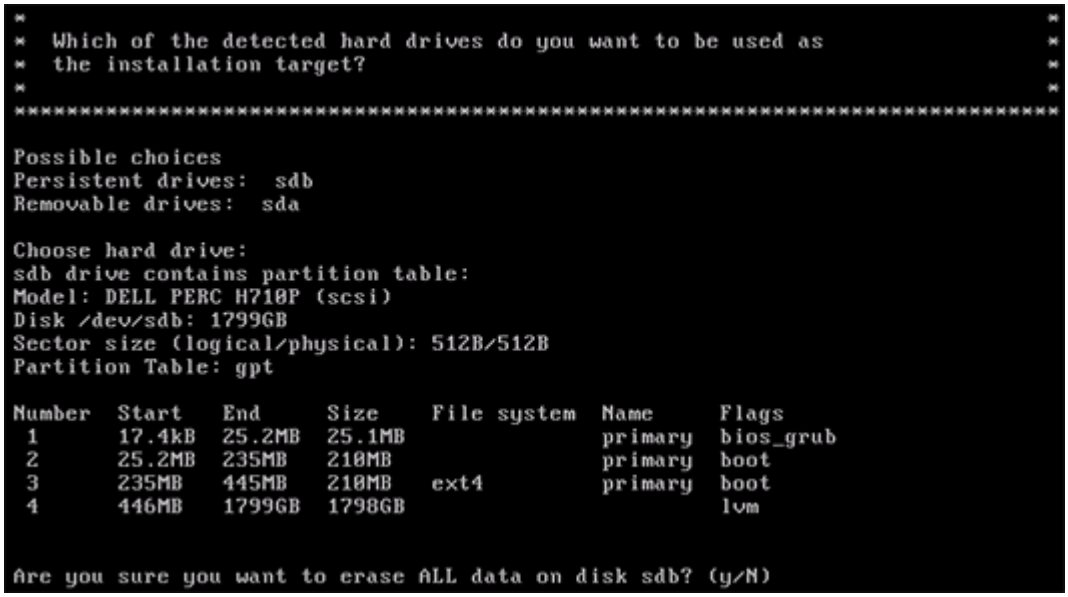


Figure 11.7 Edit Settings on the Fuel Installer Screen

If you are sure that you want to erase the existing data, follow the directions and proceed with the installation. From this point forward in the installation you should not be prompted again. If the install is successful your server will be rebooted and your automation server console should look like the figure below. You want to make sure that you change the root password at this point.

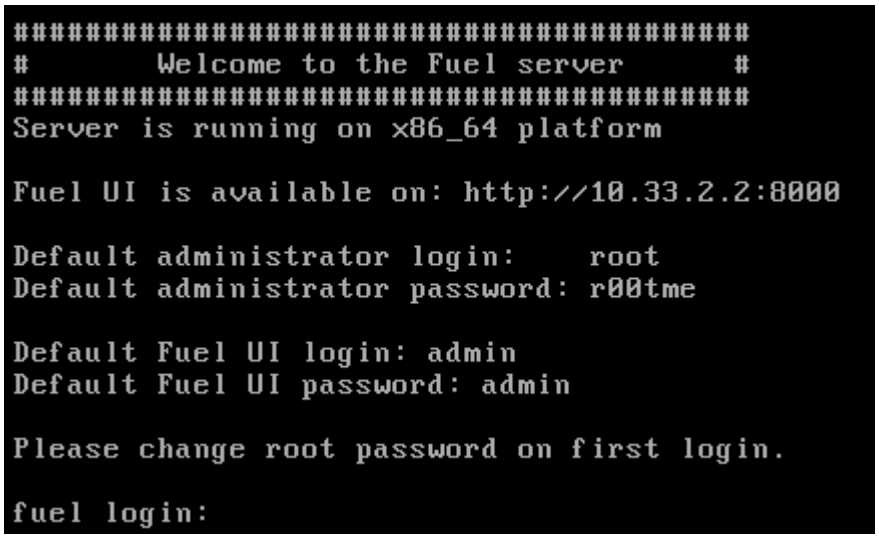


Figure 11.8 Edit Settings on the Fuel Installer Screen

Ok. You Fuel server is now installed! In the next section you will walk through a basic automated HA deployment using the web-interface. In the section after that, you will walk through a more advanced console-based installation.

11.3 Web-based Basic Fuel OpenStack Deployment

You are going to start working with Fuel using the web-interface. There is also a way to deploy OpenStack using Fuel from a CLI, but the web interface is easier to get started with. Go ahead and access the Fuel IU by navigating your web browser to *http://<fuel server ip>:8000*. For instance, in the demonstration enviorment the UI address would be: *http://10.33.2.2:8000*. On your browser you should see the UI as shown below.

Figure 11.9 Login to the Fuel Web-interface

Using the default UI login and password of *admin/admin* login to the Fuel UI. Once logged in you will be taken to the Environments screen as shown in the figure below.

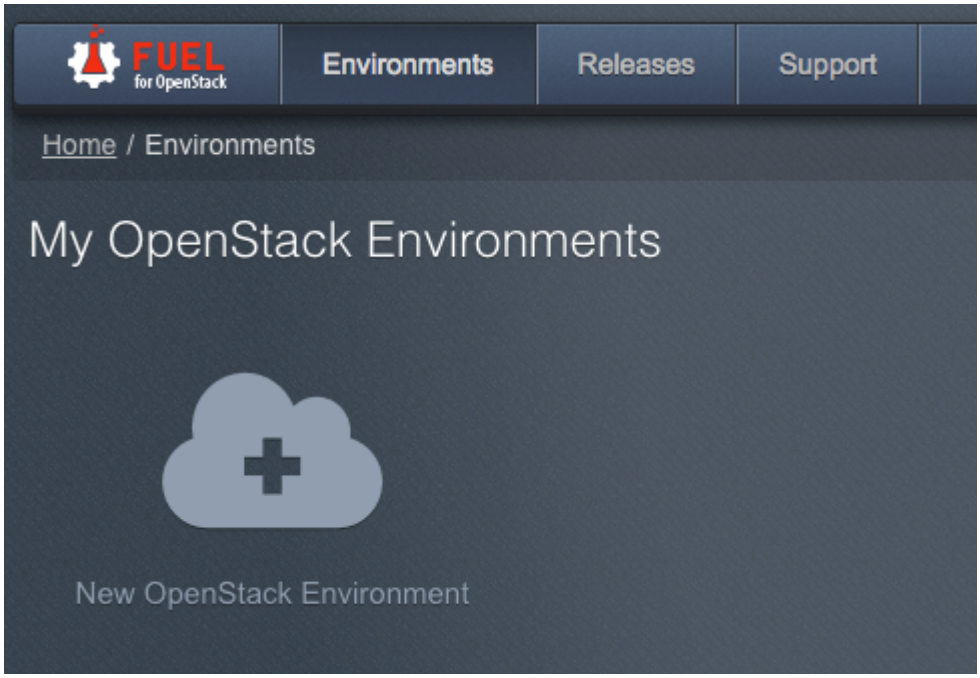


Figure 11.10 Fuel Environments Screen

Of course at this point, you haven't configured any environments, and more importantly, you don't have not discovered any servers to be used for a deployment. Continue on to the next sub-section to discover your servers.

11.3.1 Server Discovery

You might be used to management tools that perform server discovery after a service has already been deployed. Often this is done by a network address scan, an embedded agent, or simply as list of hosts. In this case your servers don't have an address to discover or even an operating system to assign an address.

Server discovery in Fuel is accomplished through a lightweight agent that is placed on each unconfigured server after an initial successful PXE boot. Which means that you must first configure all unconfigured (excluding the Fuel server) servers, to boot using PXE, as previously discussed. Then each server should be rebooted, so Fuel can manage the PXE boot process. The steps for configuring your server hardware for PXE boot and the rebooting process will be specific to your vendor hardware. Discovery is accomplished in the following steps:

- Set *Unconfigured Servers* to use PXE as first boot device.
- Restart *Unconfigured Servers*.
- *Unconfigured Servers* receive DHCP/PXE information from Fuel server.
- *Unconfigured Servers* boot using management *bootstrap* image provided by Fuel server.
- Agent running under the *bootstrap* image reports back to Fuel server once *Unconfigured Server* has booted and hardware inventory has been collected.

- *Unconfigured Server* is reported by the Fuel server as an *Unallocated Server*.

If the discovery process is successful, all of your previously *Unconfigured Servers* will now be reported as *Unallocated Servers* by the Fuel UI, as shown in the figure below. If a server does not show up as discovered, access your virtual OOB console and check the status of the host. It is not uncommon to force a cold restart from OOB if the server is in a hung state.

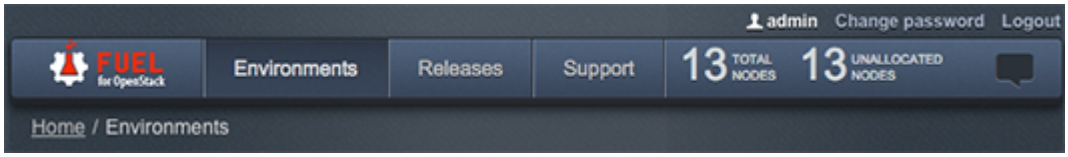


Figure 11.11 Fuel UI with 13 Unallocated Nodes

In the next sub-section you will build a new environment the be used to deploy OpenStack on your *Unallocated Servers*. This environment describes how you want your servers configured for use with OpenStack.

11.3.2 Create a Fuel Deployment Environment

At this point you are ready to describe the environment that Fuel will deploy for you. From the *Environments* tab click on the *New OpenStack Environment* icon. You will be presented with a dialog box and a series of screens like the one shown in the figure below.

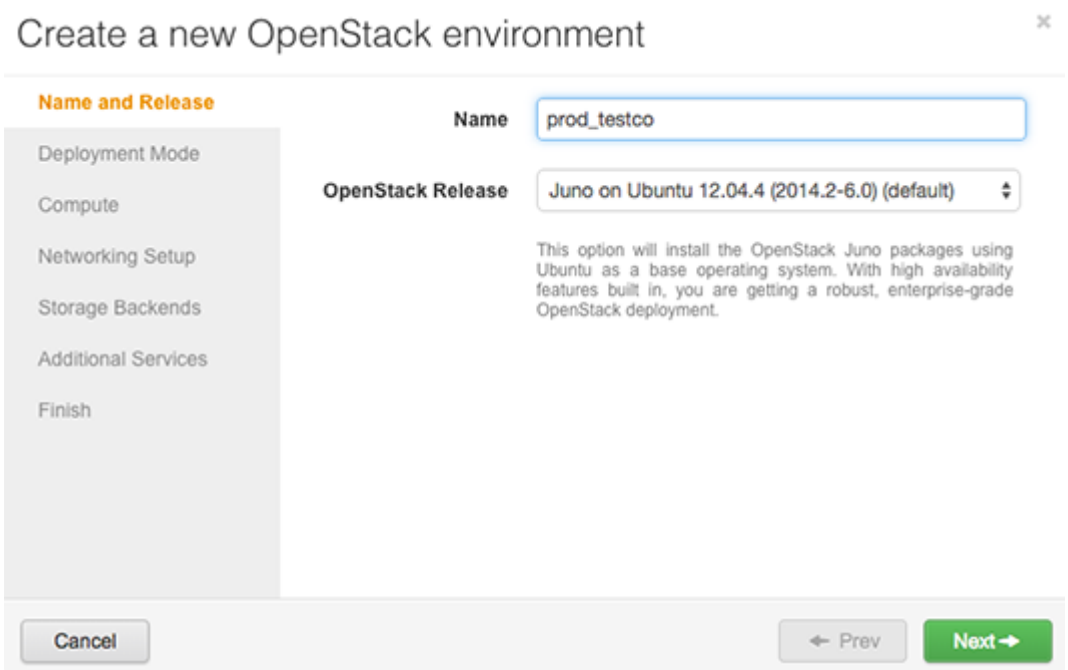


Figure 11.12 Creating a new OpenStack environment using Fuel UI

You will be asked the following series of questions:

- Name and Release: Provide the name of the deployment and the OpenStack release you wish to deploy. The name is purely cosmetic, but the OpenStack release and OS platform selection will determine the underlying OS and release platform for the entire deployment.
- Deployment Mode: If you have limited hardware you can select Multi-node. In the chapter demonstration Multi-node with HA will be selected.
- Compute: If you are running on hardware, like in the demonstration environment, select KVM. However, if you are running everything virtually you should select QEMU.
- Networking Setup: There are several choices for network type. Likely you will want to use Neutron for networking, which narrows it down to using GRE or VLAN for network segregation. The demonstration environment will use VLAN segregation, since this is a common choice in production environments, for performance reasons.
- Storage Backends: You can select either Linux LVM as the storage backend or Ceph, both of which will be configured for you. In the demonstration environment Ceph will be selected for both Cinder and Glance storage.
- Additional Services: You can optional install additional services provided by the OpenStack framework.
- Finish: When you have completed you configuration click on the <*Create*> button to build your environment configuration.

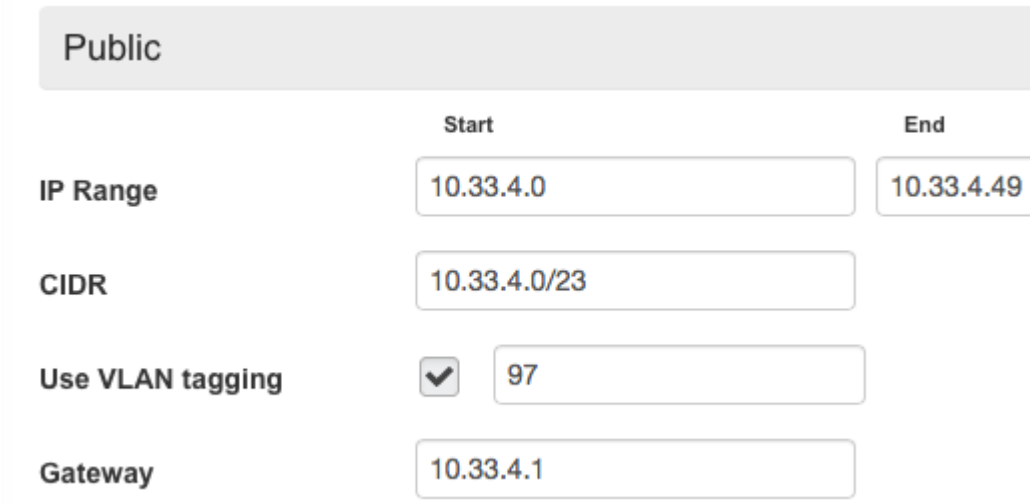
You have now created a new deployment environment configuration. Continue to the next sub-section to configure your network environment.

11.3.3 Configure the Network for the Environment

Before you assign any *Unallocated Servers*, you want to first configure the network for your environment. Completing this step first will make interface configuration during node assignment easier to understand, since the networks will already be defined by the time you assign them to your hosts.

Click on the Networks tab in your environment configuration screen. From the networks screen you will create the network configuration that will both be applied to the underlying hosts operating systems, and will also used to configure OpenStack. The following networks configuration will be based your environment. In the demonstration, VLAN segmentation was selected, so example settings will reflect that option.

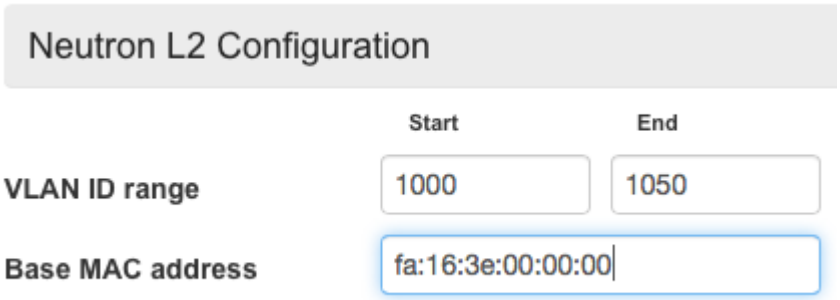
- Public: Network to be used for external VM communications. The *IP Range* is the range of addresses to be reserved for OpenStack operations, such as external router interfaces. The *CIDR* is the full subnet used for all external (OpenStack + Floating) addresses. The *Gateway* is the network gateway for the subnet. In the demonstration environment this network will used the tagged VLAN 97.



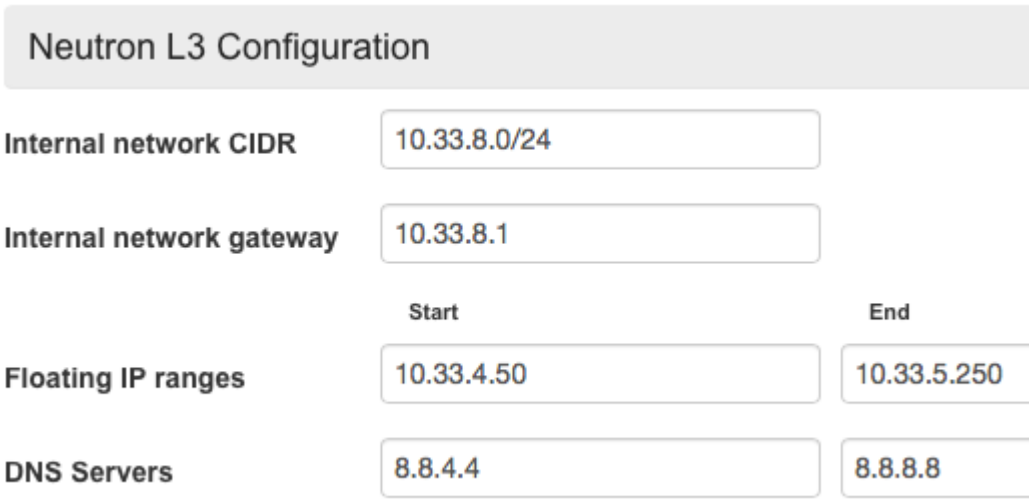
- Management: Network used by OpenStack nodes to communicate on the API level. This should be considered an internal network for OpenStack components.

- Storage: This network will carry the storage traffic from the Ceph nodes to the Compute and Glance nodes.

- Neutron L2: This is the internal or private VM network. Set a range of VLANs to be used for VM-to-VM communication between Compute nodes.



- Neutron L3: This *CIDR* and *Gateway*will be used internally when creating internal OpenStack networks. The *Floating IP ranges* are the range of addresses reserved from the *Public* network to be available to VMs as floating external addresses.



When your configuration is complete click on the <*Save Settings*> button. In the next section you'll allocate nodes to your environment. Once nodes are assigned to the environment you will return to this screen and verify your network configuration.

11.3.4 Allocating Host to your Environment

You have configured your new environment and you have a pool of discovered, yet unassigned resources. However, no physical resources have been assigned roles in the environment. The next step in the process is to assign roles to the pool of *Unallocated Servers*. While in the configuration screen for your environment click on the <*+Add Nodes*> button. A dialog screen will be presented, which list the available roles along with *Unallocated Server* candidates. The node assignment screen should look like the figure shown below.

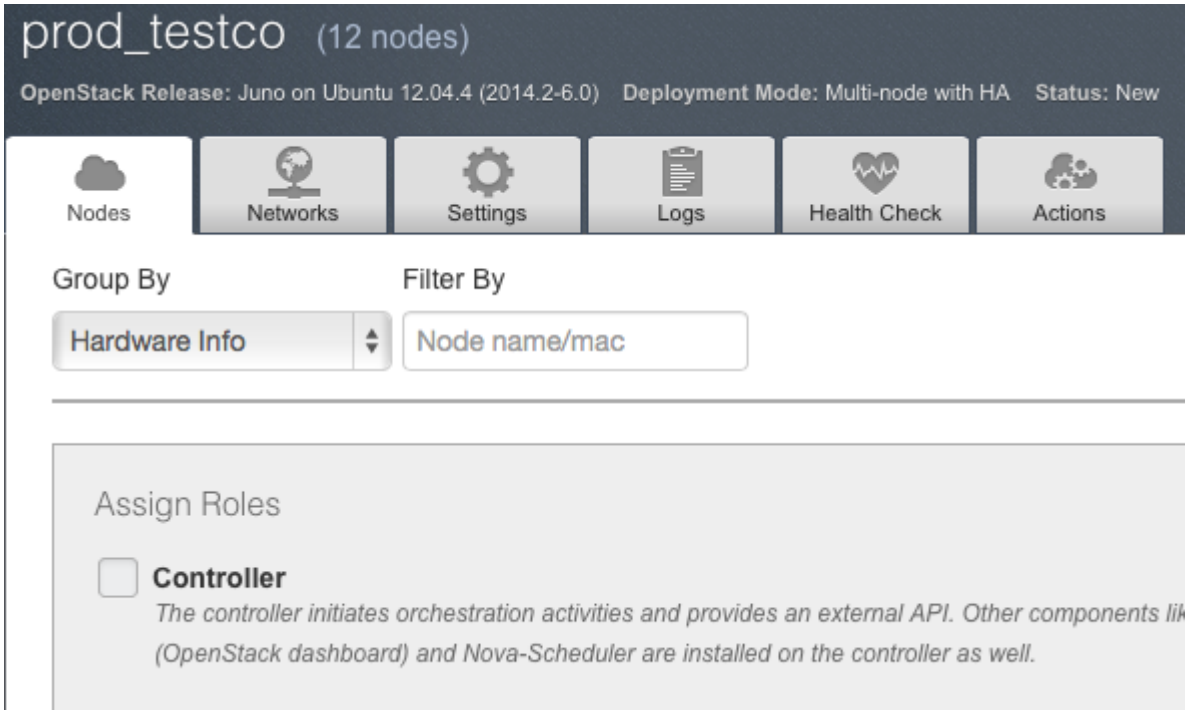


Figure 11.13 Assigning Nodes to OpenStack Roles

In the demonstration environment the following assignments were made:

- Controller: 3 x Servers /w 64G Ram
- Compute: 5 x Server /w 512G Ram
- Storage - Ceph (OSD): 3 x Server /w 48G Ram and 16.5TB Disk

In the following sub-sections disk and network will be configured from the allocation screen.

CONFIGURE INTERFACES

If you have assigned all interfaces on all nodes to be used for the same purpose, then this step is easy, otherwise you will have to repeat the interface configuration process, for every group of hosts that have unique interface configurations. If interfaces vary based on server or role, simply repeat the described configuration for each node. From the nodes tab in your environment, toggle the *[Select All]* check-box and then click the <*Configure Interfaces*> button. The node interface dialog should appear, like the one shown in the figure below.

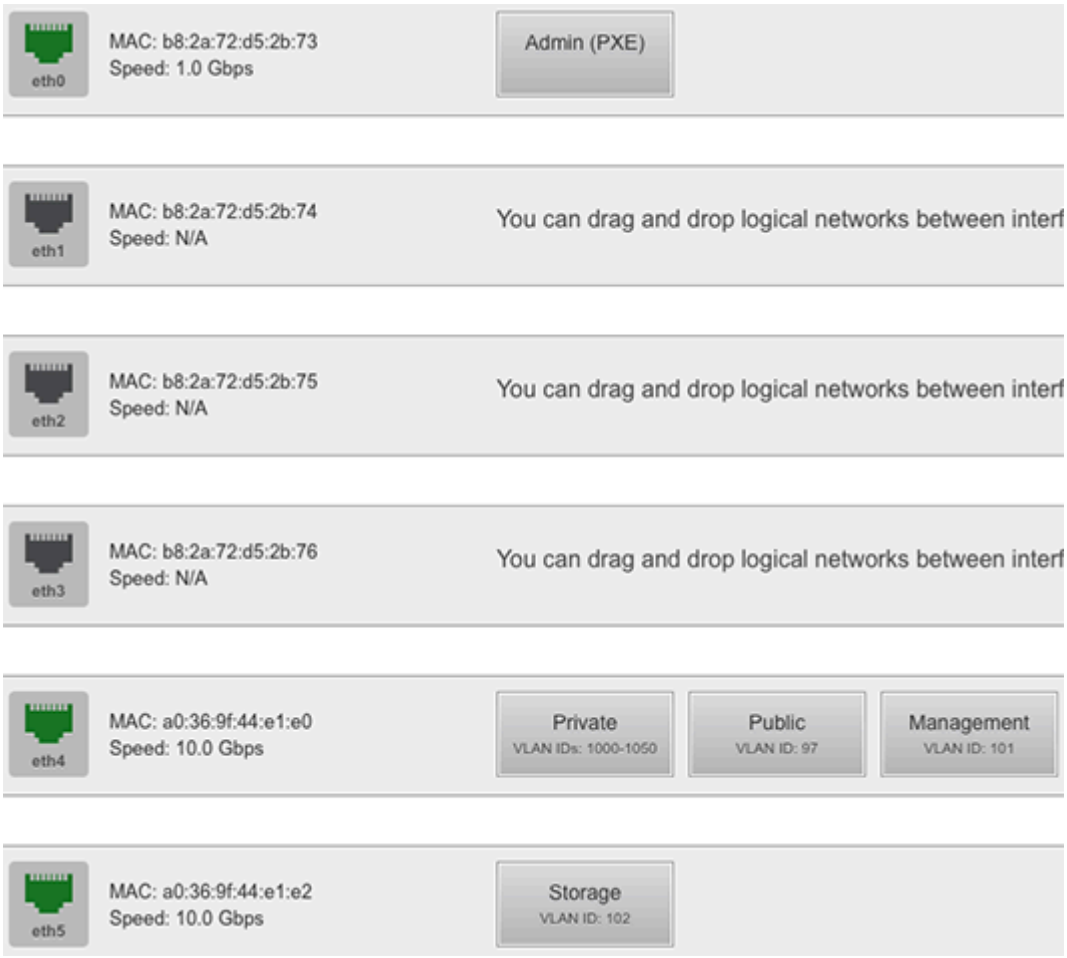


Figure 11.14 Assigning Node Interface Configuration

The networks you configured in the previous section, like the *Public* network shown in Figure 11.1, must be assigned to physical interfaces on your servers. The interfaces shown in the screen will be specific to your deployment. However, the following networks assignments must be made regardless of deployment configuration:

- Admin (PXE): This should be assigned to the automation/administration interface. In the demonstration this will remain on interface *eth0*.
- Private: This should be assigned to the interface that will carry VM-to-VM traffic internal to the OpenStack deployment. In the demonstration VLAN isolation was selected, so the interface *eth4* is expected to have tagged access to VLANs in which to transmit this traffic between nodes.
- Public: This should be assigned to the interface that will carry traffic external to the OpenStack environment.
- Management: This should be assigned to the interface that will be used to carry traffic internal to OpenStack components.
- Storage: This interface should be assigned to the interface that is connected to the storage network.

Once your interfaces have been assigned click <*Apply*> to save settings. In the next sub-section you will configure your disks.

CONFIGURE DISKS

Unlike the interface configuration, the physical disk on the separate servers are not expected to the be the same. If you do have a group of servers that have the same disk configuration, you can select the group and then click on <*Configure Disk*>. For instance, to configure the disk on a group of Ceph OSD nodes, you would select the group to access the disk configuration screen shown below.

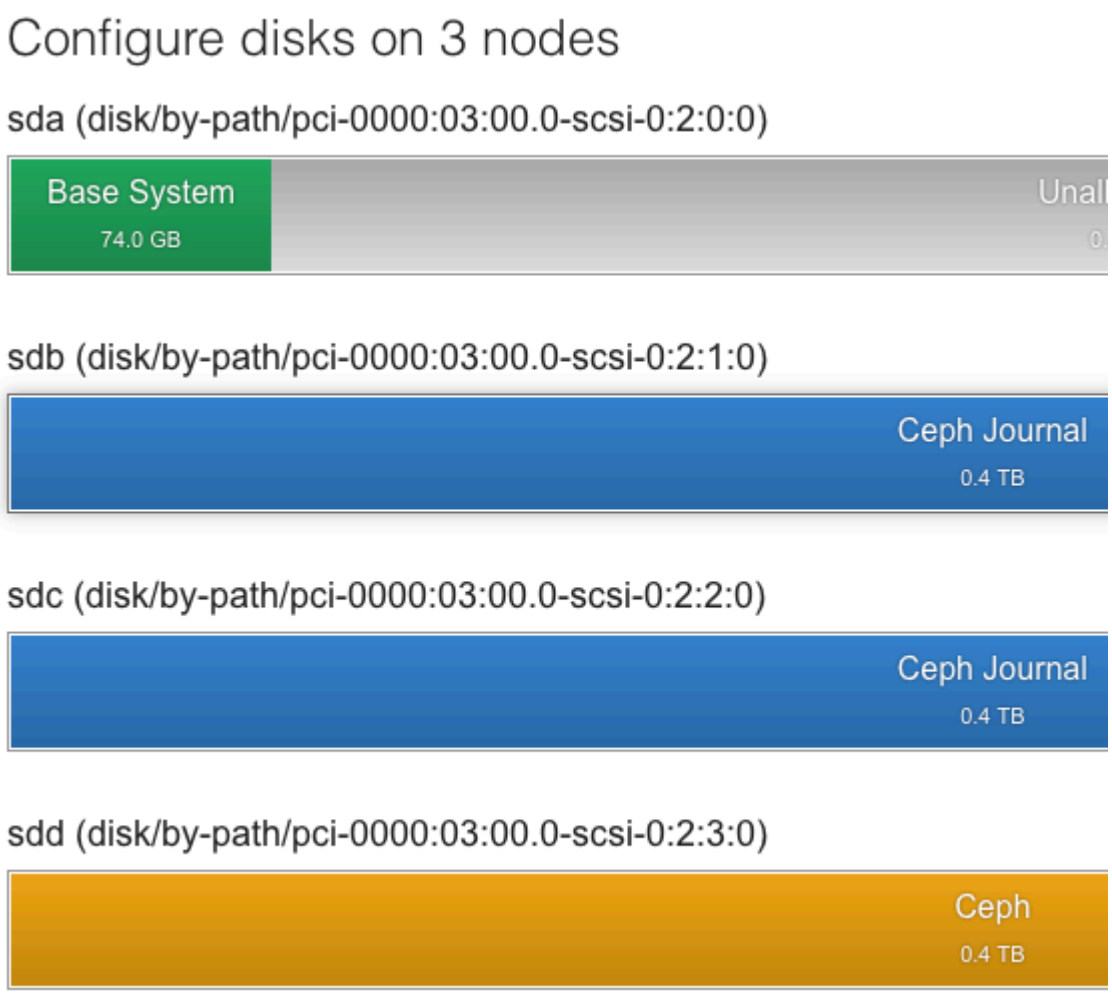


Figure 11.15 Assigning Node Disk Configuration

In the figure, you can see that some disk have been assigned as Ceph Journal devices and other have been assigned as Ceph Data devices. The configuration will be assigned for all nodes selected in the group.

Once you have configured the disk on all of your nodes, continue to the next section where you will make final configuration settings and verify your network.

11.3.5 Final Settings and Verification

Click on the Settings tab in your environment configuration screen. Under this tab you will find your existing configuration based on questions you answered during environment creation. From this screen you can more finely tune your deployment.

At a minimum we recommend that you change any passwords related to your deployment. In addition, if you are using Ceph for your storage, you might consider assigning Ceph as the backend for both Nova and the Swift API. Once you have made all changes click on the <*Save Settings*> button.

Once again, click on the Networks tab in your environment configuration screen. Scroll down to the bottom of the screen and click the <*Verify Networks*> button. If all your network settings are correct you will see *Verification succeeded*, as shown in the figure below.

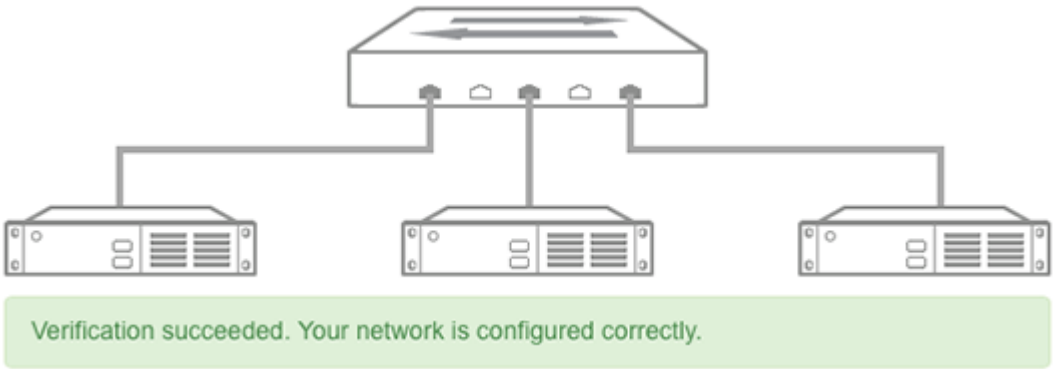


Figure 11.16 Verified Network Configuration

NOTE	**Help! My network is not documented!** Things can admittedly get complicated in this process. Something as simple as determining what server is plugged into what switch a port can get complicated. If you get completely lost, go back to the basics. No idea where the server is plugged in? You can either start chasing cables in the data-center, or you could use the Link Layer Discovery Protocol (LLDP). Consider booting the server(s) in question using a liveCD, which contains support for lldpd. Consult your switch documentation for how to enabled lldp on your network hardware. Running lldp on the switch and server will allow both devices to report link-level location. Know where the server is attached, but VLAN verification is failing? Once again, boot a server and confirm connectivity from an untagged VLAN. This can be done by assigning an IP address on the switch and the server, then use ping to confirm communication between devices. Once communication is confirmed, repeat the process using tagged VLANS.

WARNING	**Stop! Verify or Else** Do not proceed until your network configuration can be verified. You will be guaranteed to have major problems if your network configuration is not validated on all participating nodes.

11.3.6 Deploy changes

Ok. Do you see the blue button labeled <*Deploy Changes*>? If you are confident in your hardware configuration and your environment settings, go ahead an click it! This will start the deployment process, which includes OS installation and OpenStack deployment. Deployment progress will be indicated by the green progress bar on the top right hand of the screen. If you are interested in deployment details you can click on the *Logs* tab, or on the paper icon next to the individual server deployment progress bar.

When the process completes successfully, you will be provided with the address of your deployments Horizon web address as shown below.

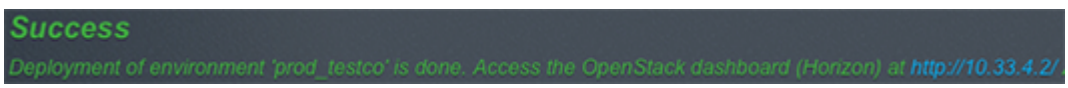


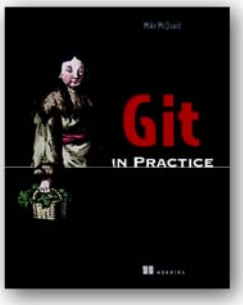
Figure 11.17 Successful Deployment

Go ahead and click on the *Health Check* tab and follow the instructions to run a full test on your environment. Some test require OpenStack tenant interactions, such as importing specific images.

If you need to add a node, simply click <+*Add Nodes*> and follow the deployment process.

To redeploy simply click the <*Reset*> button under the *Actions* tab and then <*Deploy Changes*>. If you want to completely start over, click <Reset> then <Delete> the environment. Enjoy your new HA OpenStack environment.

RELATED MANNING TITLES

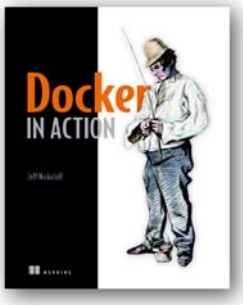


Git in Practice
by Mike McQuaid

ISBN: 9781617291975
272 pages, $39.99
September 2014

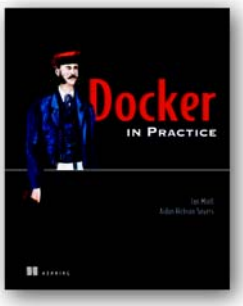
Learn Git in a Month of Lunches
by Rick Umali

ISBN: 9781617292415
375 pages, $39.99
August 2015



Docker in Action
by Jeff Nickoloff

ISBN: 9781633430235
300 pages, $49.99
December 2015



Docker in Practice
by Ian Miell and Aidan Hobson Sayers

ISBN: 9781617292729
275 pages, $44.99
November 2015

For ordering information go to www.manning.com