1
Keystone OpenStack Identity Service

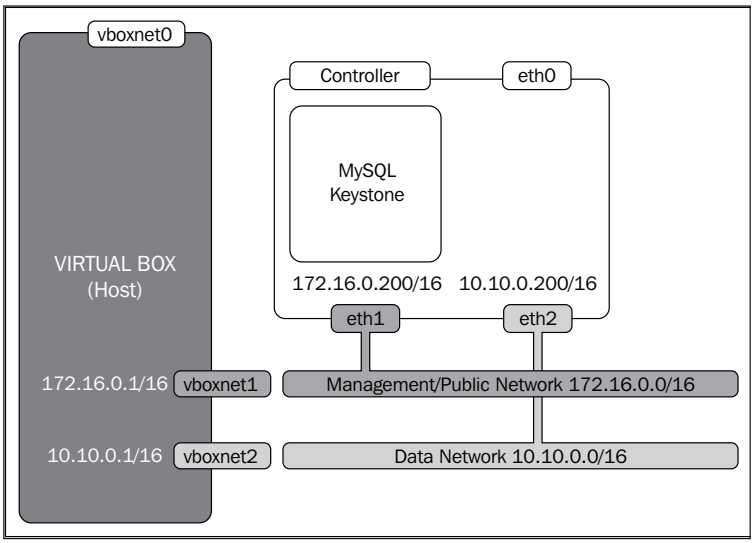
In this chapter, we will cover:

- Creating a sandbox environment using VirtualBox and Vagrant
- Configuring the Ubuntu Cloud Archive
- Installing OpenStack Identity Service
- Creating tenants
- Configuring roles
- Adding users
- Defining service endpoints
- Creating the service tenant and service users

Introduction

The OpenStack Identity Service, known as **Keystone**, provides services for authenticating and managing user accounts and role information for our OpenStack cloud environment. It is a crucial service that underpins the authentication and verification between all of our OpenStack cloud services and is the first service that needs to be installed within an OpenStack environment. Authentication with OpenStack Identity Service sends back an authorization token that is passed between the services, once validated. This token is subsequently used as your authentication and verification that you can proceed to use for that service, such as OpenStack Storage and Compute. As such, configuration of the OpenStack Identity Service must be done first and consists of creating appropriate roles for users and services, tenants, the user accounts, and the service API endpoints that make up our cloud infrastructure.

At the end of this chapter, we will have the following environment setup:



Creating a sandbox environment using VirtualBox and Vagrant

Creating a sandbox environment using VirtualBox and Vagrant allows us to discover and experiment with the OpenStack Compute service. VirtualBox gives us the ability to spin up virtual machines and networks without affecting the rest of our working environment, and is freely available at `http://www.virtualbox.org` for Windows, Mac OS X, and Linux. Vagrant allows us to automate this task, meaning we can spend less time creating our test environments and more time using OpenStack. Vagrant is installable using Ubuntu's package management, but for other operating systems, visit `http://www.vagrantup.com/`. This test environment can then be used for the rest of this chapter.

It is assumed that the computer you will be using to run your test environment has enough processing power and has hardware virtualization support (for example, Intel VT-X and AMD-V support with at least 8 GB RAM). Remember we're creating a virtual machine that itself will be used to spin up virtual machines, so the more RAM you have, the better.

Getting ready

To begin with, we must download VirtualBox from `http://www.virtualbox.org/` and then follow the installation procedure once this has been downloaded.

We also need to download and install Vagrant, which will be covered in the later part of the recipe.

The steps throughout the book assume the underlying operating system that will be used to install OpenStack on will be Ubuntu 12.04 LTS release. We don't need to download a Ubuntu 12.04 ISO as we use our Vagrant environment to do this for us.

How to do it...

To create our sandbox environment within VirtualBox, we will use Vagrant to define a single virtual machine that allows us to run all of the OpenStack Compute services required to run cloud instances. This virtual machine, that we will refer to as the OpenStack Controller, will be configured with at least 2 GB RAM and 20 GB of hard drive space and will have three network interfaces. Vagrant automatically set up an interface on our virtual machine that will **NAT** (**Network Address Translate**) traffic out, allowing our virtual machine to connect to the network outside of VirtualBox to download packages. This NAT interface is not mentioned in `Vagrantfile`, but will be visible on our virtual machine as `eth0`. We configure our first interface for use in our OpenStack environment, which will be the public interface of our OpenStack Compute host; a second interface will be used for our private network that OpenStack Compute uses for internal communication between different OpenStack Compute hosts and a third will be used when we look at Neutron networking in Chapter 8 as an external provider network.

Carry out the following steps to create a virtual machine with Vagrant that will be used to run OpenStack Compute services:

1. Install VirtualBox from `http://www.virtualbox.org/`. You will encounter issues if using the version shipped with Ubuntu 12.04 LTS.

This book was written using VirtualBox Version 4.2.16

2. Install Vagrant from `http://www.vagrantup.com/`. You will encounter issues if using the version shipped with Ubuntu 12.04 LTS.

This book was written using Vagrant Version 1.2.7

3. Once installed, we can define our virtual machine and networking in a file called `Vagrantfile`. To do this, create a working directory (for example, `~/cookbook`) and edit a file in here called `Vagrantfile` as shown in the following command snippet:

```
mkdir ~/cookbook
cd ~/cookbook
vim Vagrantfile
```

Downloading the example code

The code for this book is all on GitHub, so for all the correct steps please go to: `https://github.com/OpenStackCookbook/OpenStackCookbook`

For further queries on code, you can go to: `http://www.openstackcookbook.com/`

4. We can now proceed to configure Vagrant by editing this file with the following code:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

nodes = {
  'controller'  => [1, 200],
}

Vagrant.configure("2") do |config|
  config.vm.box = "precise64"
  config.vm.box_url = "http://files.vagrantup.com/precise64.box"

  # Forescout NAC workaround
  config.vm.usable_port_range = 2800..2900

  nodes.each do |prefix, (count, ip_start)|
    count.times do |i|
      hostname = "%s" % [prefix, (i+1)]

      config.vm.define "#{hostname}" do |box|
        box.vm.hostname = "#{hostname}.book"
        box.vm.network :private_network, ip: "172.16.0.#{ip_
start+i}", :netmask => "255.255.0.0"
        box.vm.network :private_network, ip: "10.10.0.#{ip_
start+i}", :netmask => "255.255.0.0"
        box.vm.network :private_network, ip: "192.168.100.#{ip_
start+i}", :netmask => "255.255.255.0"

      # Otherwise using VirtualBox
        box.vm.provider :virtualbox do |vbox|
        # Defaults
          vbox.customize ["modifyvm", :id, "--memory", 2048]
          vbox.customize ["modifyvm", :id, "--cpus", 1]
        end
      end
    end
  end
end
```

5. We are now ready to power on our `controller` node. We do this by simply running the following command:

```
vagrant up controller
```

Congratulations! We have successfully created a VirtualBox virtual machine running on Ubuntu 12.04 which is able to run OpenStack Controller services.

How it works...

What we have done is created a virtual machine within VirtualBox by defining it in Vagrant. Vagrant then configures this virtual machine based on the settings given in `Vagrantfile` in the directory where we want to store and run our VirtualBox virtual machines from. This file is based on Ruby syntax, but the lines are relatively self-explanatory. We have specified some of the following:

- The hostname is called `controller`
- The VM is based on Precise64, an alias for Ubuntu 12.04 LTS 64-bit
- We have specified 2GB RAM, 1 CPU, and an extra hard disk attached to our VM called `controller-cinder.vdi` that we will utilize later in the book.

We then launch this VirtualBox VM using Vagrant with the help of the following simple command:

```
vagrant up
```

This will launch all VMs listed in `Vagrantfile`. As we have only one, this VM is the only one that is started.

To log in to this new virtual machine, we use the following command:

```
vagrant ssh controller
```

There's more...

You are not limited to Vagrant and VirtualBox for setting up a test environment. There are a number of virtualization products available that are suitable for trying OpenStack, for example, *VMware Server*, *VMware Player*, and *VMware Fusion* are equally suitable.

See also

- *Chapter 10*, *Automating OpenStack Installations*

Configuring the Ubuntu Cloud Archive

Ubuntu 12.04 LTS, the release used throughout this book, provides two repositories for installing OpenStack. The standard repository ships with the Essex release whereas a further supported repository called the Ubuntu Cloud Archive provides access to the latest release (at the time of writing), Grizzly. We will be performing an installation and configuration of OpenStack Identity Service (as well as the rest of the OpenStack services) with packages from the Ubuntu Cloud Archive to provide us with the Grizzly release of software.

Getting ready

Ensure you're logged in to the nominated OpenStack Identity server or OpenStack Controller host where OpenStack Identity Service will be installed that the rest of the OpenStack hosts will have access to.

How to do it...

Carry out the following steps to configure Ubuntu 12.04 LTS to use the Ubuntu Cloud Archive:

1. To access the Ubuntu Cloud Archive repository, we first need to ensure we have the Ubuntu Cloud Archive key. We add this as follows:

   ```
   sudo apt-get update
   sudo apt-get -y install ubuntu-cloud-keyring
   ```

2. We then configure our apt sources by creating a new file (as root) called `/etc/apt/sources.list.d/grizzly.list` with the following line:

   ```
   deb http://ubuntu-cloud.archive.canonical.com/ubuntu precise-updates/grizzly main
   ```

3. Then we update our local package repository to pick up the new resources by issuing the following:

   ```
   sudo apt-get update
   ```

How it works...

What we're doing here is adding an extra repository to our system that provides us with a tested set of OpenStack packages that is fully supported on Ubuntu 12.04 LTS. The packages in here will then be ones that will be used when we perform the installation of OpenStack on our system.

There's more...

More information about the Ubuntu Cloud Archive can be found by visiting the following address: `https://wiki.ubuntu.com/ServerTeam/CloudArchive`. This explains the release process and the ability to use latest releases of OpenStack—where new versions are released every 6 months—on a long term supported release of Ubuntu that gets released every 2 years.

Installing OpenStack Identity Service

We will be performing an installation and configuration of OpenStack Identity Service, known as Keystone, using the Ubuntu Cloud Archive. Once configured, connecting to our OpenStack cloud environment will be performed through our new OpenStack Identity Service.

The backend datastore for our OpenStack Identity Service will be a MySQL database.

Getting ready

To ensure we're running the Ubuntu Cloud Archive, we must first configure our Ubuntu 12.04 installation to use this service.

We will configure Keystone to use MySQL as the database backend, so this needs to be installed prior to installing Keystone. If MySQL is not installed, perform the following steps to install and configure MySQL:

```
MYSQL_ROOT_PASS=openstack
MYSQL_HOST=172.16.0.200
# To enable non-interactive installations of MySQL, set the following
echo "mysql-server-5.5 mysql-server/root_password password $MYSQL_ROOT_
PASS" | sudo debconf-set-selections
echo "mysql-server-5.5 mysql-server/root_password_again password $MYSQL_
ROOT_PASS" | sudo debconf-set-selections
echo "mysql-server-5.5 mysql-server/root_password seen true" \
    | sudo debconf-set-selections
echo "mysql-server-5.5 mysql-server/root_password_again seen true" \
    | sudo debconf-set-selections
```

```
export DEBIAN_FRONTEND=noninteractive
sudo apt-get update
sudo apt-get -q -y install mysql-server python-mysqldb
sudo sed -i "s/^bind\-address.*/bind-address = 0.0.0.0/g" \
    /etc/mysql/my.cnf
sudo service mysql restart

mysqladmin -uroot password ${MYSQL_ROOT_PASS}

mysql -u root --password=${MYSQL_ROOT_PASS} -h localhost \
    -e "GRANT ALL ON *.* to root@\"localhost\" IDENTIFIED BY \"${MYSQL_
ROOT_PASS}\" WITH GRANT OPTION;"

mysql -u root --password=${MYSQL_ROOT_PASS} -h localhost \
    -e "GRANT ALL ON *.* to root@\"${MYSQL_HOST}\" IDENTIFIED BY
\"${MYSQL_ROOT_PASS}\" WITH GRANT OPTION;"

mysql -u root --password=${MYSQL_ROOT_PASS} -h localhost \
    -e "GRANT ALL ON *.* to root@\"%\" IDENTIFIED BY \"${MYSQL_ROOT_
PASS}\" WITH GRANT OPTION;"

mysqladmin -uroot -p${MYSQL_ROOT_PASS} flush-privileges
```

Next, ensure that you're logged in to the nominated OpenStack Identity server or OpenStack Controller host where OpenStack Identity Service will be installed and that the rest of the OpenStack hosts will have access to it.

To log on to our OpenStack Controller host that was created using Vagrant, issue the following command:

```
vagrant ssh controller
```

How to do it...

Carry out the following instructions to install OpenStack Identity Service:

1. Installation of OpenStack Identity Service is done by specifying the `keystone` package in Ubuntu, and we do this as follows:

   ```
   sudo apt-get update
   sudo apt-get -y install keystone python-keyring
   ```

2. Once installed, we need to configure the backend database store, so we first create the `keystone` database in MySQL. We do this as follows (where we have a user in MySQL called `root` with password `openstack` that is able to create databases):

```
MYSQL_ROOT_PASS=openstack
mysql -uroot -p$MYSQL_ROOT_PASS -e \
    "CREATE DATABASE keystone;"
```

3. It is a good practice to create a user that is specific to our OpenStack Identity Service, so we create this as follows:

```
MYSQL_KEYSTONE_PASS=openstack
mysql -uroot -p$MYSQL_ROOT_PASS -e \
  "GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%';"
mysql -uroot -p$MYSQL_ROOT_PASS -e "SET PASSWORD FOR \
  'keystone'@'%' = PASSWORD('$MYSQL_KEYSTONE_PASS');"
```

4. We then need to configure OpenStack Identity Service to use this database by editing the `/etc/keystone/keystone.conf` file and then change the `sql_connection` line to match the database credentials. We do this as follows:

```
MYSQL_HOST=172.16.0.200
sudo sed -i "s#^connection.*#connection = \
  mysql://keystone:openstack@172.16.0.200/keystone#" \
  /etc/keystone/keystone.conf
```

5. A super-user admin token resides in the `/etc/keystone/keystone.conf` file. To configure this, we do the following:

```
sudo sed -i "s/^# admin_token.*/admin_token = ADMIN/" \
  /etc/keystone/keystone.conf
```

6. As of the Grizzly release, Keystone supports PKI infrastructure to cryptographically sign the tokens. To disable this feature for now, we edit the `/etc/keystone/keystone.conf` file to use non-signed tokens as follows:

```
sudo sed -i "s/^#token_format.*/token_format = UUID/" \
  /etc/keystone/keystone.conf
```

7. We can now restart the `keystone` service:

```
sudo stop keystone
sudo start keystone
```

8. With Keystone started, we can now populate the `keystone` database with the required tables by issuing the following command:

```
sudo keystone-manage db_sync
```

Congratulations! We now have OpenStack Identity Service installed and ready for use in our OpenStack environment.

How it works...

A convenient way to install OpenStack Identity Service ready for use in our OpenStack environment is by using the Ubuntu packages. Once installed, we configure our MySQL database server with a `keystone` database and set up the `keystone.conf` configuration file to use this. After starting the Keystone service, running the `keystone-manage db_sync` command populates the `keystone` database with the appropriate tables ready for us to add in the required users, roles, and tenants required in our OpenStack environment.

Creating tenants

A tenant in OpenStack is a project. Users can't be created without having a tenant assigned to them, so these must be created first. For this section, we will create a tenant for our users called `cookbook`.

Getting ready

To begin with, ensure you're logged in to our OpenStack Controller host—where OpenStack Identity Service has been installed—or an appropriate Ubuntu client that has access to where OpenStack Identity Service is installed.

To log on to our OpenStack Controller host that was created using Vagrant, issue the following command:

```
vagrant ssh controller
```

If the `keystoneclient` tool isn't available, this can be installed on an Ubuntu client—to manage our OpenStack Identity Service—by issuing the following command:

```
sudo apt-get update
sudo apt-get -y install python-keystoneclient
```

Ensure that we have our environment set correctly to access our OpenStack environment for administrative purposes:

```
export ENDPOINT=172.16.0.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=http://${ENDPOINT}:35357/v2.0
```

How to do it...

To create a tenant in our OpenStack environment, perform the following steps:

1. The creation of a tenant called `cookbook` is done as follows:

```
keystone tenant-create \
    --name cookbook \
    --description "Default Cookbook Tenant" \
    --enabled true
```

   This will produce output like the following:

```
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     Default Cookbook Tenant      |
|   enabled   |               True               |
|      id     | 8ec8e07a759e46d2abb316ee368d0e5b |
|     name    |             cookbook             |
+-------------+----------------------------------+
```

2. We also need an `admin` tenant, so when we create users in this tenant, they have access to our complete environment. We do this in the same way as in the previous step:

```
keystone tenant-create \
    --name admin \
    --description "Admin Tenant" \
    --enabled true
```

How it works...

Creation of the roles is simply achieved by using the `keystone` client, specifying the `tenant-create` option with the following syntax:

```
keystone tenant-create \
    --name tenant_name \
    --description "A description" \
    --enabled true
```

`tenant_name` is an arbitrary string and must not contain spaces. On creation of the tenant, this returns an ID associated with it that we use when adding users to this tenant. To see a list of tenants and the associated IDs in our environment, we can issue the following command:

```
keystone tenant-list
```

Configuring roles

Roles are the permissions given to users within a tenant. Here we will configure two roles, an *admin* role that allows for administration of our environment and a *Member* role that is given to ordinary users who will be using the cloud environment.

Getting ready

To begin with, ensure that you're logged in to our OpenStack Controller host—where OpenStack Identity Service has been installed—or an appropriate Ubuntu client that has access to where OpenStack Identity Service is installed.

To log on to our OpenStack Controller host that was created using Vagrant, issue the following command:

```
vagrant ssh controller
```

If the `keystoneclient` tool isn't available, this can be installed on any Ubuntu client that has access to manage our OpenStack Identity Service by issuing the following commands:

```
sudo apt-get update
sudo apt-get -y install python-keystoneclient
```

To configure OpenStack Identity Service, we use super-user privileges in the form of a permanently set admin token set in the `/etc/keystone/keystone.conf` file, and we set the correct environment variables for this purpose as follows:

```
export ENDPOINT=172.16.0.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=http://${ENDPOINT}:35357/v2.0
```

How to do it...

To create the required roles in our OpenStack environment, perform the following steps:

1. The creation of the `admin` role is done as follows:

   ```
   # admin role
   keystone role-create --name admin
   ```

   This will show output like the following when successful:

   ```
   +----------+----------------------------------+
   | Property |              Value               |
   +----------+----------------------------------+
   |    id    | e20157f33ae14cfab3ddd193b57ce747 |
   |   name   |              admin               |
   +----------+----------------------------------+
   ```

2. To create the `Member` role, we repeat the step specifying the `Member` role:

```
# Member role
keystone role-create --name Member
```

How it works...

Creation of the roles is simply achieved by using the `keystone` client, specifying the `role-create` option with the following syntax:

```
keystone role-create --name role_name
```

The `role_name` attribute can't be arbitrary. The `admin` role has been set in `/etc/keystone/policy.json` as having administrative rights:

```
{
    "admin_required": [["role:admin"], ["is_admin:1"]]
}
```

And when we configure the OpenStack Dashboard, Horizon, it has the `Member` role configured as default when users are created in that interface.

On creation of the role, this returns an ID associated with it that we use when assigning roles to users. To see a list of roles and the associated IDs in our environment, we can issue the following command:

```
keystone role-list
```

Adding users

Adding users to OpenStack Identity Service requires that the user have a tenant they can exist in, and also have a role defined that can be assigned to them. For this section, we will create two users. The first user will be named `admin` and will have the `admin` role assigned to them in the `cookbook` tenant. The second user will be named `demo` and will have the `Member` role assigned to them in the same `cookbook` tenant.

Getting ready

To begin with, ensure that you're logged in to our OpenStack Controller host—where OpenStack Identity Service has been installed—or an appropriate Ubuntu client that has access to where OpenStack Identity Service is installed.

To log on to our OpenStack Controller host that was created using Vagrant, issue the following command:

```
vagrant ssh controller
```

If the `keystone` client tool isn't available, this can be installed on an Ubuntu client—to manage our OpenStack Identity Service—by issuing the following commands:

```
sudo apt-get update
sudo apt-get -y install python-keystoneclient
```

Ensure that we have our environment set correctly to access our OpenStack environment for administrative purposes:

```
export ENDPOINT=172.16.0.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=http://${ENDPOINT}:35357/v2.0
```

How to do it...

To create the required users in our OpenStack environment, perform the following steps:

1. To create a user in the `cookbook` tenant, we first need to get the `cookbook` tenant ID. To do this, issue the following command, which we conveniently store in a variable named `TENANT_ID` with the `tenant-list` option:

   ```
   TENANT_ID=$(keystone tenant-list \
       | awk '/\ cookbook\ / {print $2}')
   ```

2. Now that we have the tenant ID, the creation of the admin user in the `cookbook` tenant is done as follows, using the `user-create` option and choosing a password for the user:

   ```
   PASSWORD=openstack
   keystone user-create \
       --name admin \
       --tenant_id $TENANT_ID \
       --pass $PASSWORD \
       --email root@localhost \
       --enabled true
   ```

   This will produce the following output:

```
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |          root@localhost          |
| enabled  |               True               |
|    id    | b5f7f18eea8b46e5ba8832b27be771fd |
|   name   |              admin               |
| tenantId | 8ec8e07a759e46d2abb316ee368d0e5b |
+----------+----------------------------------+
```

3. As we are creating the admin user, which we are assigning the `admin` role, we need the `admin` role ID. In a similar way to the discovery of the tenant ID in step 1, we pick out the ID of the `admin` role and conveniently store it in a variable to use it when assigning the role to the user with the `role-list` option:

```
ROLE_ID=$(keystone role-list \
    | awk '/\ admin\ / {print $2}')
```

4. To assign the role to our user, we need to use the user ID that was returned when we created that user. To get this, we can list the users and pick out the ID for that particular user with the following `user-list` option:

```
USER_ID=$(keystone user-list \
    | awk '/\ admin\ / {print $2}')
```

5. Finally, with the tenant ID, user ID, and an appropriate role ID available, we can assign that role to the user with the following `user-role-add` option:

```
keystone user-role-add \
    --user $USER_ID \
    --role $ROLE_ID \
    --tenant_id $TENANT_ID
```

Note that there is no output produced on successfully running this command.

6. The admin user also needs to be in the `admin` tenant for us to be able to administer the complete environment. To do this, we need to get the `admin` tenant ID and then repeat the previous step using this new tenant ID as follows:

```
ADMIN_TENANT_ID=$(keystone tenant-list \
    | awk '/\ admin\ / {print $2}')
keystone user-role-add \
    --user $USER_ID \
    --role $ROLE_ID \
    --tenant_id $ADMIN_TENANT_ID
```

7. To create the demo user in the `cookbook` tenant with the `Member` role assigned, we repeat the process as defined in steps 1 to 5:

```
# Get the cookbook tenant ID
TENANT_ID=$(keystone tenant-list \
    | awk '/\ cookbook\ / {print $2}')

# Create the user
PASSWORD=openstack
keystone user-create \
    --name demo \
    --tenant_id $TENANT_ID \
    --pass $PASSWORD \
    --email demo@localhost \
    --enabled true

# Get the Member role ID
ROLE_ID=$(keystone role-list \
    | awk '/\ Member\ / {print $2}')

# Get the demo user ID
USER_ID=$(keystone user-list \
    | awk '/\ demo\ / {print $2}')

# Assign the Member role to the demo user in cookbook
keystone user-role-add \
    --user $USER_ID \
    --role $ROLE_ID \
    --tenant $TENANT_ID
```

How it works...

Adding users in OpenStack Identity Service requires that the tenant and roles for that user be created first. Once these are available, in order to use the `keystone` command-line client, we need the IDs of the tenants and IDs of the roles that are to be assigned to the user in that tenant. Note that a user can be a member of many tenants and can have different roles assigned in each.

To create a user with the `user-create` option, the syntax is as follows:

```
keystone user-create \
    --name user_name \
    --tenant_id TENANT_ID \
    --pass PASSWORD \
    --email email_address \
    --enabled true
```

The `user_name` attribute is an arbitrary name but cannot contain any spaces. A `password` attribute must be present. In the previous examples, these were set to `openstack`. The `email_address` attribute must also be present.

To assign a role to a user with the `user-role-add` option, the syntax is as follows:

```
keystone user-role-add \
    --user USER_ID \
    --role ROLE_ID \
    --tenant TENANT_ID
```

This means we need to have the ID of the user, the ID of the role, and the ID of the tenant in order to assign roles to users. These IDs can be found using the following commands:

```
keystone tenant-list
keystone role-list
keystone user-list
```

Defining service endpoints

Each of the services in our cloud environment runs on a particular URL and port—these are the endpoint addresses for our services. When a client communicates with our OpenStack environment that runs OpenStack Identity Service, it is this service that returns the endpoint URLs, which the user can then use in an OpenStack environment. To enable this feature, we must define these endpoints. In a cloud environment, though, we can define multiple regions. Regions can be thought of as different datacenters, which would imply that they would have different URLs or IP addresses. Under OpenStack Identity Service, we can define these URL endpoints separately for each region. As we only have a single environment, we will reference this as RegionOne.

Getting ready

To begin with, ensure you're logged in to our OpenStack Controller host—where OpenStack Identity Service has been installed—or an appropriate Ubuntu client that has access to where OpenStack Identity Service is installed.

To log on to our OpenStack Controller host that was created using Vagrant, issue the following command:

```
vagrant ssh controller
```

If the `keystone` client tool isn't available, this can be installed on an Ubuntu client—to manage our OpenStack Identity Service—by issuing the following commands:

```
sudo apt-get update
sudo apt-get -y install python-keystoneclient
```

Ensure that we have our environment set correctly to access our OpenStack environment for administrative purposes:

```
export ENDPOINT=172.16.0.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=http://${ENDPOINT}:35357/v2.0
```

How to do it...

Defining the services and service endpoints in OpenStack Identity Service involves running the `keystone` client command to specify the different services and the URLs that they run from. Although we might not have all services currently running in our environment, we will be configuring them within OpenStack Identity Service for future use. To define endpoints for services in our OpenStack environment, carry out the following steps:

1. We can now define the actual services that OpenStack Identity Service needs to know about in our environment:

```
# OpenStack Compute Nova API Endpoint
keystone service-create \
    --name nova \
    --type compute \
    --description 'OpenStack Compute Service'
```

```
# OpenStack Compute EC2 API Endpoint
keystone service-create \
    --name ec2 \
    --type ec2 \
    --description 'EC2 Service'

# Glance Image Service Endpoint
keystone service-create \
    --name glance \
    --type image \
    --description 'OpenStack Image Service'

# Keystone Identity Service Endpoint
keystone service-create \
    --name keystone \
    --type identity \
    --description 'OpenStack Identity Service'

#Cinder Block Storage Endpoint
keystone service-create \
    --name volume \
    --type volume \
    --description 'Volume Service'
```

2. After we have done this, we can add in the service endpoint URLs that these services run on. To do this, we need the ID that was returned for each of the service endpoints created in the previous step. This is then used as a parameter when specifying the endpoint URLs for that service.

OpenStack Identity Service can be configured to service requests on three URLs: a public facing URL (that the end users use), an administration URL (that users with administrative access can use that might have a different URL), and an internal URL (that is appropriate when presenting the services on either side of a firewall to the public URL).

For the following services, we will configure the public and internal service URLs to be the same, which is appropriate for our environment:

```
# OpenStack Compute Nova API
NOVA_SERVICE_ID=$(keystone service-list \
    | awk '/\ nova\ / {print $2}')

PUBLIC="http://$ENDPOINT:8774/v2/\$(tenant_id)s"
ADMIN=$PUBLIC
INTERNAL=$PUBLIC

keystone endpoint-create \
    --region RegionOne \
    --service_id $NOVA_SERVICE_ID \
    --publicurl $PUBLIC \
    --adminurl $ADMIN \
    --internalurl $INTERNAL
```

This will produce output similar to the following:

```
+-------------+-----------------------------------------+
|   Property  |                  Value                  |
+-------------+-----------------------------------------+
|   adminurl  | http://172.16.0.200:8774/v2/$(tenant_id)s |
|      id     |      e64eca45d255414e984a84877e902423     |
| internalurl | http://172.16.0.200:8774/v2/$(tenant_id)s |
|  publicurl  | http://172.16.0.200:8774/v2/$(tenant_id)s |
|    region   |                RegionOne                |
|  service_id |      0999b3e54a874a95a995e6fa7adc300f     |
+-------------+-----------------------------------------+
```

3. We continue to define the rest of our service endpoints as shown in the following code:

```
# OpenStack Compute EC2 API
EC2_SERVICE_ID=$(keystone service-list \
    | awk '/\ ec2\ / {print $2}')

PUBLIC="http://$ENDPOINT:8773/services/Cloud"
ADMIN="http://$ENDPOINT:8773/services/Admin"
INTERNAL=$PUBLIC
```

```
keystone endpoint-create \
    --region RegionOne \
    --service_id $EC2_SERVICE_ID \
    --publicurl $PUBLIC \
    --adminurl $ADMIN \
    --internalurl $INTERNAL


# Glance Image Service
GLANCE_SERVICE_ID=$(keystone service-list \
    | awk '/\ glance\ / {print $2}')

PUBLIC="http://$ENDPOINT:9292/v1"
ADMIN=$PUBLIC
INTERNAL=$PUBLIC

keystone endpoint-create \
    --region RegionOne \
    --service_id $GLANCE_SERVICE_ID \
    --publicurl $PUBLIC \
    --adminurl $ADMIN \
    --internalurl $INTERNAL


# Keystone OpenStack Identity Service
KEYSTONE_SERVICE_ID=$(keystone service-list \
    | awk '/\ keystone\ / {print $2}')

PUBLIC="http://$ENDPOINT:5000/v2.0"
ADMIN="http://$ENDPOINT:35357/v2.0"
INTERNAL=$PUBLIC

keystone endpoint-create \
    --region RegionOne \
    --service_id $KEYSTONE_SERVICE_ID \
    --publicurl $PUBLIC \
    --adminurl $ADMIN \
    --internalurl $INTERNAL
```

```
#Cinder Block Storage ServiceService
CINDER_SERVICE_ID=$(keystone service-list \
    | awk '/\ volume\ / {print $2}')

PUBLIC="http://$ENDPOINT:8776/v1/%(tenant_id)s"
ADMIN=$PUBLIC
INTERNAL=$PUBLIC

keystone endpoint-create \
    --region RegionOne \
    --service_id $CINDER_SERVICE_ID  \
    --publicurl $PUBLIC \
    --adminurl $ADMIN \
    --internalurl $INTERNAL
```

How it works...

Configuring the services and endpoints within OpenStack Identity Service is done with the `keystone` client command.

We first add the service definitions by using the `keystone` client and the `service-create` option with the following syntax:

```
keystone service-create \
    --name service_name \
    --type service_type \
    --description 'description'
```

`service_name` is an arbitrary name or label defining our service of a particular type. We refer to the name when defining the endpoint to fetch the ID of the service.

The `type` option can be one of the following: compute, object-store, image-service, and identity-service. Note that we haven't configured the OpenStack Object Storage service (type object-store) or Cinder at this stage as these are covered in later recipes in the book.

The `description` field is again an arbitrary field describing the service.

Once we have added in our service definitions, we can tell OpenStack Identity Service where those services run from by defining the endpoints using the `keystone` client and the `endpoint-create` option with the following syntax:

```
keystone endpoint-create \
    --region region_name \
    --service_id service_id \
    --publicurl public_url \
    --adminurl admin_url \
    --internalurl internal_url
```

Here, `service_id` is the ID of the service when we created the service definitions in the first step. The list of our services and IDs can be obtained by running the following command:

```
keystone service-list
```

As OpenStack is designed for global deployments, a region defines a physical datacenter or a geographical area that comprises of multiple connected datacenters. For our purpose, we define just a single region—*RegionOne*. This is an arbitrary name that we can reference when specifying what runs in what datacenter/area, and we carry this through to when we configure our client for use with these regions.

All of our services can be configured to run on three different URLs as follows, depending on how we want to configure our OpenStack cloud environment:

- The `public_url` parameter is the URL that end users would connect on. In a public cloud environment, this would be a public URL that resolves to a public IP address.
- The `admin_url` parameter is a restricted address for conducting administration. In a public deployment, you would keep this separate from `public_URL` by presenting the service you are configuring on a different, restricted URL. Some services have a different URI for the admin service, so this is configured using this attribute.
- The `internal_url` parameter would be the IP or URL that existed only within the private local area network. The reason for this is that you are able to connect to services from your cloud environment internally without connecting over a public IP address space, which could incur data charges for traversing the Internet. It is also potentially more secure and less complex to do so.

Once the initial `keystone` database has been set up, after running the initial `keystone-manage db_sync` command on the OpenStack Identity Service server, administration can be done remotely using the `keystone` client.

Creating the service tenant and service users

With the service endpoints created, we can now configure them so that our OpenStack services can utilize them. To do this, each service is configured with a username and password within a special `service` tenant. Configuring each service to have their own username and password allows for greater security, troubleshooting, and auditing within our environment. For each service that uses OpenStack Identity Service for authentication and authorization, we then specify these details in their relevant configuration file when setting up that service. Each service itself has to authenticate with `keystone` in order for it to be available within OpenStack. Configuration of that service is then done using these credentials. For example, for `glance`, we specify the following in `/etc/glance/glance-registry-api.ini` when used with OpenStack Identity Service, which matches what we created previously:

```
[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 172.16.0.200
service_port = 5000
auth_host = 172.16.0.200
auth_port = 35357
auth_protocol = http
auth_uri = http://172.16.0.200:5000/
admin_tenant_name = service
admin_user = glance
admin_password = glance
```

Getting ready

To begin with, ensure you're logged in to our OpenStack Controller host—where OpenStack Identity Service has been installed—or an appropriate Ubuntu client that has access to where OpenStack Identity Service is installed.

To log on to our OpenStack Controller host that was created using Vagrant, issue the following command:

```
vagrant ssh controller
```

If the `keystone` client tool isn't available, this can be installed on an Ubuntu client to manage our OpenStack Identity Service by issuing the following command:

```
sudo apt-get update
sudo apt-get -y install python-keystoneclient
```

Ensure that we have our environment set correctly to access our OpenStack environment:

```
export ENDPOINT=172.16.0.200
export SERVICE_TOKEN=ADMIN
export SERVICE_ENDPOINT=http://${ENDPOINT}:35357/v2.0
```

How to do it...

To configure an appropriate `service` tenant, carry out the following steps:

1. Create the `service` tenant as follows:

   ```
   keystone tenant-create \
       --name service \
       --description "Service Tenant" \
       --enabled true
   ```

   This produces output similar to what is shown as follows:

   ```
   +-------------+----------------------------------+
   |   Property  |              Value               |
   +-------------+----------------------------------+
   | description |          Service Tenant          |
   |   enabled   |               True               |
   |      id     | ffb9576f8fe847f883fb73784ca6ab48 |
   |     name    |             service              |
   +-------------+----------------------------------+
   ```

2. Record the ID of the `service` tenant so that we can assign service users to this ID, as follows:

   ```
   SERVICE_TENANT_ID=$(keystone tenant-list \
       | awk '/\ service\ / {print $2}')
   ```

3. For each of the services in this section, we will create the user accounts to be named the same as the services and set the password to be the same as the service name too. For example, we will add a user called `nova` with a password `nova` in the `service` tenant, using the `user-create` option as follows:

```
keystone user-create \
    --name nova \
    --pass nova \
    --tenant_id $SERVICE_TENANT_ID \
    --email nova@localhost \
    --enabled true
```

This will produce output similar to what is shown as follows:

```
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |          nova@localhost          |
| enabled  |               True               |
|    id    | 44021f9a74fc4c98bb0a0e2a50b401fe |
|   name   |               nova               |
| tenantId | ffb9576f8fe847f883fb73784ca6ab48 |
+----------+----------------------------------+
```

4. We then repeat this for each of our other services that will use OpenStack Identity Service:

```
keystone user-create \
    --name glance \
    --pass glance \
    --tenant_id $SERVICE_TENANT_ID \
    --email glance@localhost \
    --enabled true


keystone user-create \
    --name keystone \
    --pass keystone \
    --tenant_id $SERVICE_TENANT_ID \
    --email keystone@localhost \
    --enabled true
```

```
keystone user-create \
    --name cinder \
    --pass cinder \
    --tenant_id $SERVICE_TENANT_ID \
    --email cinder@localhost \
    --enabled true
```

5. We can now assign these users the `admin` role in the `service` tenant. To do this, we use the `user-role-add` option after retrieving the user ID of the `nova` user. For example, to add the `admin` role to the `nova` user in the `service` tenant, we do the following:

```
# Get the nova user id
NOVA_USER_ID=$(keystone user-list \
    | awk '/\ nova\ / {print $2}')

# Get the admin role id
ADMIN_ROLE_ID=$(keystone role-list \
    | awk '/\ admin\ / {print $2}')

# Assign the nova user the admin role in service tenant
keystone user-role-add \
    --user $NOVA_USER_ID \
    --role $ADMIN_ROLE_ID \
    --tenant_id $SERVICE_TENANT_ID
```

6. We then repeat this for our other service users: `glance`, `keystone`, and `cinder`:

```
# Get the glance user id
GLANCE_USER_ID=$(keystone user-list \
    | awk '/\ glance\ / {print $2}')

# Assign the glance user the admin role in service tenant
keystone user-role-add \
    --user $GLANCE_USER_ID \
    --role $ADMIN_ROLE_ID \
    --tenant_id $SERVICE_TENANT_ID
# Get the keystone user id
KEYSTONE_USER_ID=$(keystone user-list \
    | awk '/\ keystone\ / {print $2}')
```

```
# Assign the keystone user the admin role in service tenant
keystone user-role-add \
    --user $KEYSTONE_USER_ID \
    --role $ADMIN_ROLE_ID \
    --tenant_id $SERVICE_TENANT_ID

# Get the cinder user id
CINDER_USER_ID=$(keystone user-list \
    | awk '/\ cinder \ / {print $2}')

# Assign the cinder user the admin role in service tenant
keystone user-role-add \
    --user $CINDER_USER_ID \
    --role $ADMIN_ROLE_ID \
    --tenant_id $SERVICE_TENANT_ID
```

How it works...

Creation of the `service` tenant populated with the services required to run OpenStack is no different from creating any other users on our system that require the `admin` role. We create the usernames and passwords and ensure they exist in the `service` tenant with the `admin` role assigned to each user. We then use these credentials when configuring the services to authenticate with OpenStack Identity Service.