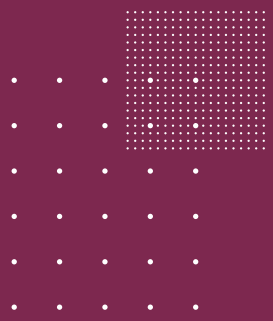
CANONICAL



WHITE PAPER

Configuring Keystone in OpenStack (Essex)

Joshua Tobin | **April 2012**

Executive introduction

Joshua Tobin
is a member of the Professional Engineering Services team at Canonical. As a Services Engineer he is thrilled to be working on all things OpenStack related. Prior to Canonical, Tobin spent five years as DevOps helping to scale the MMOG 3D virtual world of Second Life.

Keystone is an identity service written in Python that provides a pluggable back end, designed to support various protocols for authentication and authorisation (Basic Auth, OAuth, and OpenID, to give a few examples). Simply put, it allows clients to obtain security tokens to access different cloud services. Keystone was spawned from the OpenStack project and is designed to work with the OpenStack API.

The following tutorial is a brief walk-through the configuration of Keystone, based on the Essex-4 OpenStack release in Ubuntu 12.04. There were major changes made to Keystone during Essex-3 and Essex-4, so be sure you're using the latest Essex-4 code. It is important to familiarise yourself with a few key concepts before continuing with the tutorial.

Throughout the tutorial, it is assumed we have three hosts, each with a separate IP address. The first host will be running the nova-api, the second host will consist of MySQL/Glance/Keystone and the third host will be referenced as a swift endpoint.

Contents

Configuring Keystone

Let's start by setting up a MySQL database. Note that the identity service supports various other stores such as LDAP, however these will not be covered here.

Before installing MySQL, we first pre-seed the root user by configuring the password:

```
$ MYSQL_PASS=ubunturocks
$ cat <<MYSQL_PRESEED | sudo debconf-set-selections
mysql-server-5.1 mysql-server/root_password password $MYSQL_PASS
mysql-server-5.1 mysql-server/root_password_again password $MYSQL_PASS
mysql-server-5.1 mysql-server/start_on_boot boolean true
MYSQL_PRESEED
```

Install from apt sources:

```
$ sudo apt-get install -y mysql-server
```

Configure MySQL to listen on all interfaces:

```
$ sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
```

Restart MySQL:

```
$ sudo service mysql restart
```

Lastly, we need to configure a Keystone user and permissions for MySQL:

```
$ mysql -u root -p
mysql> CREATE DATABASE keystone;
       CREATE USER 'keystone'@'localhost' IDENTIFIED BY 'Secret_pass';
       GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'
       WITH GRANT OPTION;
       CREATE USER 'keystone'@'%' IDENTIFIED BY 'Secret_pass';
       GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%'
       IDENTIFIED BY 'Secret_pass';
       FLUSH PRIVILEGES;
```

Now we're ready to start the installation and configuration of Keystone. First we'll install via apt sources.

```
$ sudo apt-get install -y keystone curl python-mysqldb python-dateutil
```

Keystone's configuration resides in /etc/keystone/keystone.conf. We'll need to update the MySQL connection as well as configure the admin token, as shown below.

Using your favourite editor, open /etc/keystone/keystone.conf.

```
$ sudo vi /etc/keystone/keystone.conf
```

Comment out the following default sql connection parameter.

```
#connection = sqlite:////var/lib/keystone/keystone.db
```

Insert a new connection parameter. Be sure to insert your password to match the MySQL permissions.

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

Next we need to update the ADMIN_TOKEN variable. Note that the admin_token is essentially the keys to the kingdom. It should be kept secure and should be changed from the default. Add the following:

```
admin_token = 999888777666
```

Change file permissions to make sure only the Keystone user/group can read the file

```
$  sudo chown keystone:root /etc/keystone/keystone.conf
$  sudo chmod 0640 /etc/keystone/keystone.conf
```

In order to properly communicate with the Keystone service, we'll need to export the following variables. For KEYSTONE_IP be sure the IP address is the IP address of the host that is running the Keystone API, in our case, 10.230.7.2.

```
$ export KEYSTONE_IP=10.230.7.2 # IP of your keystone API server
$ export SERVICE_ENDPOINT=http://$KEYSTONE_IP:35357/v2.0/
$ export SERVICE_TOKEN=999888777666
```

Now we need to seed the database with the keystone-manage utility.

```
$ sudo keystone-manage db_sync
```

Restart the Keystone service and check to see that the API is functional.

```
$ sudo service keystone restart
```

Running Keystone user-list should return an empty set of users. This will verify that the API is functional. If any errors occur, ensure that you've properly exported the Keystone variables listed above.

```
$ keystone user-list
```

Next we need to tell Keystone which endpoints are available. Endpoints are a combination of URLs and ports where a service may be accessed. In this example we'll be defining services for the Compute (Nova-API), Identity (Keystone), Image (Glance-API), and Storage (Swift).

First, let's make a few bash variables to simplify the configuration. Be sure to substitute the IP addresses below to match your environment.

```
$ NOVA_IP=10.230.7.1
$ GLANCE_IP=10.230.7.2
$ SWIFT_IP=10.230.7.3
```

Each endpoint requires a Public, Admin, and Internal URL. Note that '%(tenant_id)s' is a template variable that will be used by Keystone to substitute a tenant's id and form a proper URL to gain access to specific tenants. Again we're going to make a few bash variables for each endpoint to make configuration a bit easier.

```
NOVA_PUBLIC_URL="http://$NOVA_IP:8774/v1.1/%(tenant_id)s"
NOVA_ADMIN_URL=$NOVA_PUBLIC_URL
NOVA_INTERNAL_URL=$NOVA_PUBLIC_URL

GLANCE_PUBLIC_URL="http://$GLANCE_IP:9292/v1"
GLANCE_ADMIN_URL=$GLANCE_PUBLIC_URL
GLANCE_INTERNAL_URL=$GLANCE_PUBLIC_URL

KEYSTONE_PUBLIC_URL="http://$KEYSTONE_IP:5000/v2.0"
KEYSTONE_ADMIN_URL="http://$KEYSTONE_IP:35357/v2.0"
KEYSTONE_INTERNAL_URL=$KEYSTONE_PUBLIC_URL

SWIFT_PUBLIC_URL="https://$SWIFT_IP:443/v1/AUTH_%(tenant_id)s"
SWIFT_ADMIN_URL="https://$SWIFT_IP:443/v1"
SWIFT_INTERNAL_URL=$SWIFT_PUBLIC_URL
```

Use the service-create argument, which will add entries in the database for each service endpoint.

```
$ keystone service-create --name nova --type compute --description
'OpenStack Compute Service'

$ keystone service-create --name swift --type object-store
--description 'OpenStack Storage Service'

$ keystone service-create --name glance --type image --description
'OpenStack Image Service'

$ keystone service-create --name keystone --type identity
--description 'OpenStack Identity Service'
```

Now for the tricky part. In order for Keystone to properly map each service endpoint to a URL we need to associate each service endpoint ID, which is a UUID created by Keystone, to a URL and a region. For this we'll again use bash variables to create each endpoint.

Starting with the compute service, we'll need to use the service-list command to get the service ID.

```
$ ID=$(keystone service-list | grep -i compute | awk '{print $2}')
```

Then let's echo the ID to make sure we've got what we're expecting.

```
$  echo $ID

e97d2e58d0d34054b34ce1994701a136
```

Using the endpoint-create argument, map the service ID to a region, let's call it RegionOne, and each Public, Admin and Internal URL variable we've created above.

```
$ keystone endpoint-create --region RegionOne --service_id $ID
--publicurl $NOVA_PUBLIC_URL --adminurl $NOVA_ADMIN_URL --internalurl
$NOVA_INTERNAL_URL
```

We've now got an endpoint created in the database. Let's verify that the endpoint looks as expected by using the endpoint-list argument. This will return an ASCII table of each of the endpoints available to Keystone.

```
$ keystone endpoint-list
```

Next create the remaining three endpoints:

```
$ ID=$(keystone service-list | grep -i object-store | awk ‘{print
$2}’)

$ keystone endpoint-create --region RegionOne --service_id $ID
--publicurl $SWIFT_PUBLIC_URL --adminurl $SWIFT_ADMIN_URL
--internalurl $SWIFT_INTERNAL_URL

$ ID=$(keystone service-list | grep -i identity | awk ‘{print $2}’)

$ keystone endpoint-create --region RegionOne --service_id $ID
--publicurl $KEYSTONE_PUBLIC_URL --adminurl $KEYSTONE_ADMIN_URL
--internalurl $KEYSTONE_INTERNAL_URL

$ ID=$(keystone service-list | grep -i  image | awk ‘{print $2}’)

$ keystone endpoint-create --region RegionOne --service_id $ID
--publicurl $GLANCE_PUBLIC_URL --adminurl $GLANCE_ADMIN_URL
--internalurl $GLANCE_INTERNAL_URL
```

Tenants within Keystone represent a high level of grouping of users. A tenant can have zero or more users and users can also be in more than one tenant. Below, we’re creating a tenant using the tenant-create argument with a name of ubuntu. We’ll need to refer to the tenant ID in a moment so we’ll create yet another bash variable.

```
$ TENANT_ID=$(keystone tenant-create --name ubuntu | grep id | awk
‘{print $4}’)
```

A role in Keystone can be thought of as metadata that is assigned to user and tenant pairs. As you will again see, we’ll need to capture the ID of each role.

```
$ ADMIN_ROLE=$(keystone role-create --name Admin|grep id| awk
‘{print $4}’)

$ KEYSTONE_ADMIN_ROLE=$(keystone role-create --name
KeystoneServiceAdmin|grep id| awk ‘{print $4}’)

$ MEMBER_ROLE=$(keystone role-create --name Member|grep id| awk
‘{print $4}’)
```

Users have account credentials and can be associated with one or more tenants. Let’s create two users, admin and ubuntu, using the user-create argument. Note that we’re associating both the admin user and the ubuntu user with the ubuntu tenant.

```
$ keystone user-create --name admin --tenant_id $TENANT_ID --pass
openstack \

--email root@localhost --enabled true

$ keystone user-create --name ubuntu --tenant_id $TENANT_ID --pass
openstack \

--email ubuntu@localhost --enabled true
```

We're almost there! Next, we need to associate a user with a role, using the user-role-add argument of the Keystone tool. This requires a user id, role id, and a tenant id. As you may have noticed above we created three different roles. An ADMIN_ROLE, KEYSTONE_ADMIN_ROLE and a MEMBER_ROLE, each with different id's. Since a user can belong to multiple roles, we're going to script a for loop to add the admin user to all available roles, and the ubuntu user to all roles except the KEYSTONE_ADMIN_ROLE.

 # Get the user id for the admin user

```
$ ADMIN_USER=$(keystone user-list | grep admin | awk '{print $2}')
```

 # Add the Admin user to Admin, KeystoneServiceAdmin, and Member Roles

```
$ for ROLE in Admin KeystoneServiceAdmin Member
do

ROLE_ID=$(keystone role-list | grep "\ $ROLE\ " | awk '{print $2}')

keystone user-role-add --user $ADMIN_USER --role $ROLE_ID --tenant_id
$TENANT_ID

done
```

 # Get the user id for the ubuntu user

```
$ UBUNTU_USER=$(keystone user-list | grep ubuntu | awk '{print $2}')
```

 # Add the ubuntu User to Admin and Member Roles

```
$ for ROLE in Admin Member
do

ROLE_ID=$(keystone role-list | grep "\ $ROLE\ " | awk '{print $2}')

keystone user-role-add --user $UBUNTU_USER --role $ROLE_ID --tenant_
id $TENANT_ID

done
```

Now let's ask Keystone about our new configuration and verify that our tokens have access to the tenant!

Log into the host that has the Keystone API and run the following command using curl. This will verify the API is up and listening. Hint: piping to python -mjson.tool makes the output much easier to read.

```
curl http://localhost:5000/v2.0/ | python -mjson.tool
```

Next, let's ask Keystone for the token ID for the admin user. You should have received json output printed. Note the ID of the token.

```
$ curl -d '{"auth":{"passwordCredentials":{"username": "admin",
"password": "openstack"}}}' -H "Content-type: application/json"
http://localhost:35357/v2.0/tokens|python -mjson.tool
```

We can then ask Keystone what tenants a token has access to. Be sure to substitute the token ID you received from the last curl.

```
$ curl -H "X-Auth-Token:887665443383838" http://localhost:5000/v2.0/
tenants
```

Thats it! You should now see the ubuntu tenant associated with a user's token!

Professional services from Canonical

Canonical provides commercial support for Ubuntu in the form of Ubuntu Advantage, a range of professional service packages, delivered by Ubuntu experts. To learn more about how we could help you implement your cloud strategy, visit:

ubuntu.com/business/services/overview

Credit for the endpoint creation code goes to Kevin Jackson (kevin@linuxservices.co.uk).

Canonical Limited, Registered in England and Wales, Company number 110334C
Registered Office: One Circular Road, Douglas, Isle of Man IM1 1SB VAT Registration: GB 003 2322 47