Lab 1: Create a VM and launch an Application

Section 1 Accessing the System

As you are now aware, there are two primary interfaces for managing OpenStack resources. You have access to both OpenStack Dashboard (Horizon) and the text-based command line interface (cli) in the TWC Portal.

Exercise 1.1 Accessing the Portal through Horizon

For this class, you are already on the internal network and will simply need to log in using Horizon at https://horizon.os.cloud.twc.net/horizon with your user credentials. Note that for off site use, you will need to log into the TWC network via VPN to access Horizon (or the CLI interfaces for that matter).

If you have not already, log in to the Portal to verify that you have access.

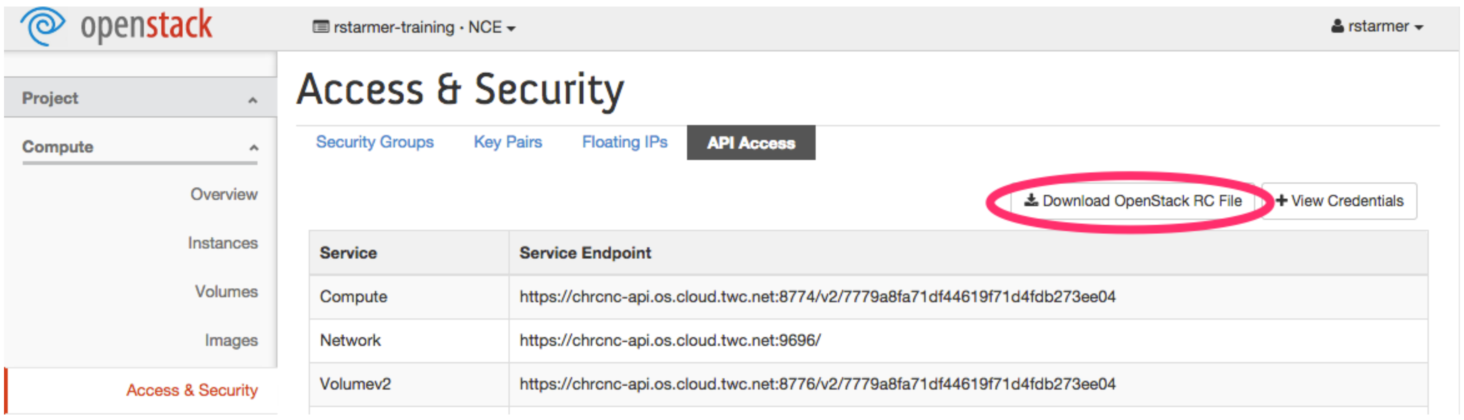
Exercise 1.2 Access the Portal through a Terminal session and set up cli clients

To use the cli you will first need to install the appropriate OpenStack command line clients, and you will normally do this on your development machine (e.g. your laptop, or a development VM depending on your needs). In general, the installation is as simple as running the “pip” python installer. For details on specific OS installations visit:
http://docs.openstack.org/user-guide/content/install_clients.html

As an example, on an Ubuntu/Debian Linux machine:

```
$ sudo apt-get install python-pip python-dev
$ sudo pip install python-{nova,cinder,keystone,neutron,swift,heat,glance,ceilometer}client
```

In order to be able to use OpenStack cli clients you have just installed, you need to set up the necessary environment variables. While you could manually create a script named openrc.sh from scratch, we can get this file from the our Horizon Portal session. If you have logged out or not yet logged in, log into Horizon and then navigate to Project > Compute> Access and Security> API access. Select the ‘Download OpenStack rc file’ button and copy this file into your local dev directory. (You can create one now if needed.)



Service	Service Endpoint
Compute	https://chrcnc-api.os.cloud.twc.net:8774/v2/7779a8fa71df44619f71d4fdb273ee04
Network	https://chrcnc-api.os.cloud.twc.net:9696/
Volumev2	https://chrcnc-api.os.cloud.twc.net:8776/v2/7779a8fa71df44619f71d4fdb273ee04

The file you've just downloaded contains the following local environmental information:

```
export OS_USERNAME="JoeUser"
export OS_PASSWORD="joes_password"
export OS_TENANT_NAME="demo"
export OS_AUTH_URL="https://chrcnc-api.os.cloud.twc.net:5000/v2.0"
export OS_REGION_NAME="NCE"
```

In the above, OS_USERNAME/OS_PASSWORD are OpenStack user name and its password. OS_TENANT_NAME is the name of the project created. OS_AUTH_URL is the URL of the Keystone endpoint. in your deployment, and OS_REGION_NAME is the region (NCE or NCW) for your application deployment

Now, set the environment variables by running openrc.sh as follows (note the download name will be {project}-openrc.sh).

```
$ source openrc.sh
```

You will be prompted to provide your TWC Portal password. Once that is accepted, you are ready to interact with OpenStack through OpenStack command line clients.

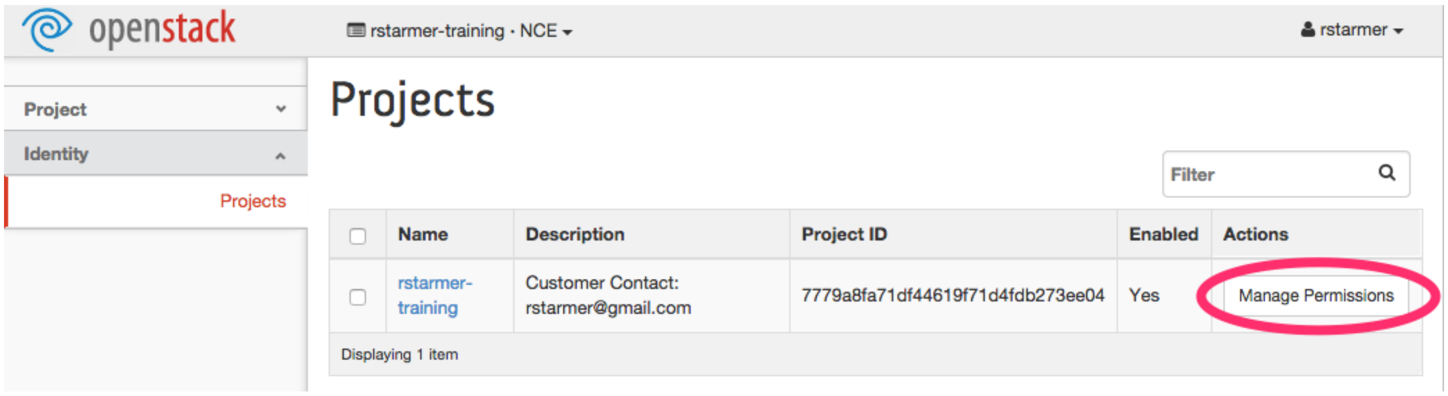
Just to check that all is working run the command:

```
$ nova list
```

This should show an empty result (as below), as you have not yet created any VMs, but this indicates that you now have cli access to OpenStack.

```
+----+------+--------+------------+-------------+----------+
| ID | Name | Status | Task State | Power State | Networks |
+----+------+--------+------------+-------------+----------+
+----+------+--------+------------+-------------+----------+
```

One capability that only exists in the Horizon portal is the ability to manage project users if you have the “ProjectOwner” role. With this capability, you can add users to your project, and grant them access based on the currently available set of roles, at a minimum other users should have the _member_ role.

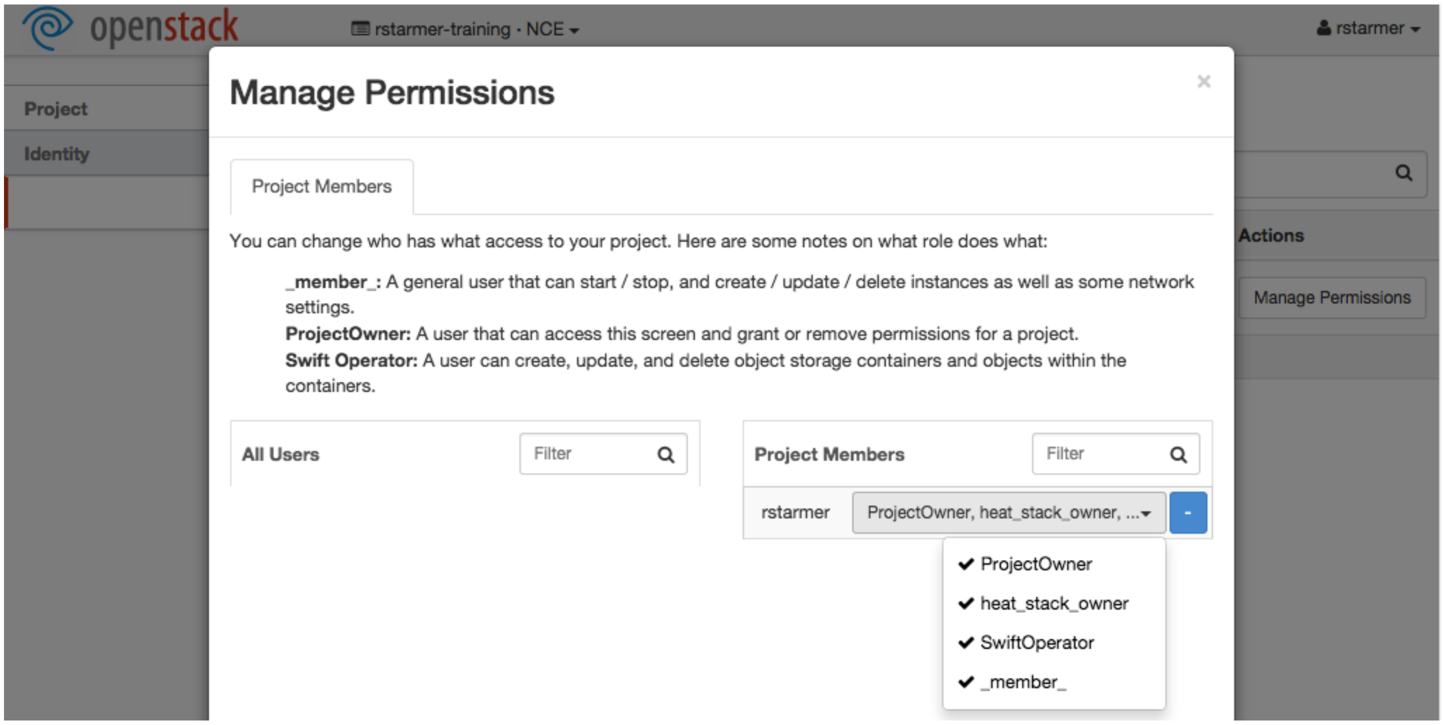


Projects

Name	Description	Project ID	Enabled	Actions
rstarmer-training	Customer Contact: rstarmer@gmail.com	7779a8fa71df44619f71d4fdb273ee04	Yes	Manage Permissions

Displaying 1 item

sss



Manage Permissions

Project Members

You can change who has what access to your project. Here are some notes on what role does what:

member: A general user that can start / stop, and create / update / delete instances as well as some network settings.
ProjectOwner: A user that can access this screen and grant or remove permissions for a project.
Swift Operator: A user can create, update, and delete object storage containers and objects within the containers.

Section 2 Planning your deployment model

Now you have access, its time to think about planning the environment for the application you are going to develop.

Consider the following requirements for your application, as this may simplify some of the initial configuration, though the flexible nature of OpenStack makes it fairly simple to re-architect your application as development progresses.

1) Compute:
   a) How many VMs
   b) Generic image or custom upload
   c) Access to VM via ssh/security keys
2) Storage:
   a)Single ephemeral root?
   b)Custom flavor?
   c)Persistent root?
   d)Extra disks?
   e)Backup strategy?
3) Network:
   a)Single or multi-segment
   b)Security between instances on a segment (or from “other” segments)

As an example, throughout this session, we will build an application that has the following characteristics:

Compute:
Start with 1 VM, and eventually move to 2+ (scaleable app servers with a single database node)
Use a generic image
VM access via individual user ssh keys (and a group shared ‘admin’ ssh key)
Storage:
Initially everything on a single ephemeral root disk
DB node will be deployed on a persistent root disk with a separate storage volume for data
Backup will be done via snapshot to swift
Network:
Single network
Restrictions on DB access, limited to the app VM(s) and ssh from anywhere that it is allowed (the TWC newtork)

Section 3 Deploy your first instance and load an application

In this section you will manually create a VM instance and then deploy an application on that VM. You will have the opportunity to first run through this process using the OpenStack cli and then replicate this with Horizon.

Exercise 3.1 Manual VM creation with the cli

If you did not do so earlier, after installing the cli clients, start by seeing if there are already any VM instances running in your environment. Run the following:

```
$ nova list
```

This should show an empty result (as below), as you have not yet created any VMs.

```
+----+------+--------+------------+-------------+----------+
| ID | Name | Status | Task State | Power State | Networks |
+----+------+--------+------------+-------------+----------+
+----+------+--------+------------+-------------+----------+
```

Now verify that you have a network to connect your VM to by listing available networks:

```
$ neutron net –list
+--------------------------------------+-----------------+-----------------------------------------------------+
| id                                   | name            | subnets                                             |
+--------------------------------------+-----------------+-----------------------------------------------------+
| 0e54e9a0-01f8-44f9-8414-3608a9c9eb21 | default-network | be07929a-0bea-4ea7-ade3-89dd252a467f 192.168.0.0/24 |
| 83eabd71-839c-4ae4-bc51-830436b632ee | Ext-Net         | b445135f-fd86-4361-80e9-7925415c1bb5                |
+--------------------------------------+-----------------+-----------------------------------------------------+
$ neutron subnet-list
+--------------------------------------+----------------+----------------+--------------------------------------------------+
| id                                   | name           | cidr           | allocation_pools                                 |
+--------------------------------------+----------------+----------------+--------------------------------------------------+
| be07929a-0bea-4ea7-ade3-89dd252a467f | default-subnet | 192.168.0.0/24 | {"start": "192.168.0.2", "end": "192.168.0.254"} |
+--------------------------------------+----------------+----------------+--------------------------------------------------+
```

You should see the information for a ‘default-network’ and ‘default-subnet’ listed. You could create a new network and subnet at this point if needed with the commands:

```
$ neutron net-create ‘network-name’
$ neutron subnet-create ‘network-name’ 10.0.0.0/24 --name ‘network-name’_subnet
```

In the output, make a note of the "id" of the default-network. These long strings of variable characters are found throughout the OpenStack system to identify a wide variety of resources. In this case, the network id you have identified will be used connect your VM to when you run the command to create the new instance.

You now need to determine type of VM that you want to create. Run the following command to see what ‘flavors’ you have access to:

```
$ nova flavor-list
```

```
+----+-----------------+-----------+------+-----------+------+-------+-------------+-----------+
| ID | Name            | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------------+-----------+------+-----------+------+-------+-------------+-----------+
| 16 | standard.large  | 16384     | 60   | 0         |      | 8     | 1.0         | True      |
| 32 | standard.xlarge | 32768     | 80   | 0         |      | 8     | 1.0         | True      |
| 4  | standard.small  | 4096      | 20   | 0         |      | 2     | 1.0         | True      |
| 8  | standard.medium | 8192      | 40   | 0         |      | 4     | 1.0         | True      |
+----+-----------------+-----------+------+-----------+------+-------+-------------+-----------+
```

For this lab exercise ‘standard.small’ is more than adequate in size, so make a note of this flavor name.

Next you need to select a VM disk image to use for your VM. To get a list of all available VM images, run:

```
$ nova image-list
```

```
+--------------------------------------+--------------------------------------+--------+---------------------+
| ID                                   | Name                                 | Status | Server              |
+--------------------------------------+--------------------------------------+--------+---------------------+
| 3acd2573-4c15-4ec7-9678-dd045657627d | CentOS-6.6-i386                      | ACTIVE |                     |
| 030b2b42-137a-4b22-a145-5a9a88d24b90 | CentOS-Server-6-x86_64               | ACTIVE |                     |
| 84547b8c-dd0d-45a1-9abe-277af95ed2cd | CentOS.6.5.2014.05.001[DEPRECATED]   | ACTIVE |                     |
| 3d63ff80-0049-40e1-b892-608b719f47ab | CentOS.6.5.2014.08.002[DEPRECATED]   | ACTIVE |                     |
| 8f518f87-b669-4048-85aa-e77e68a75ad2 | CentOS.6.5.2014.10.003[DEPRECATED]   | ACTIVE | 197b6-...90a989412  |
| 4f71ca14-7305-48d1-9182-92c0f2c3ba2e | RedHat-Server-6-x86_64               | ACTIVE |                     |
| 9bfc58e6-4300-43be-96d9-462a510a828e | RedHat-Server-7-x86_64               | ACTIVE |                     |
| 125cf2cc-42a7-4bea-9e9c-766361a692b6 | Ubuntu-Server-12.04                  | ACTIVE |                     |
| 183ed8b0-586a-436b-ae8d-b7769bf2e441 | Ubuntu-Server-14.04                  | ACTIVE |                     |
| e697c97c-d9d4-490e-9270-f3eeff540e31 | Ubuntu.12.04.2014.05.001[DEPRECATED] | ACTIVE |                     |
| 2f4c5bec-421d-4126-957a-cd006d9cbdb3 | Ubuntu.12.04.2014.10.002[DEPRECATED] | ACTIVE |                     |
| 1d8e1185-0e59-44b4-aabd-2d0102fd6731 | Ubuntu.14.04.2014.06.001[DEPRECATED] | ACTIVE |                     |
| c8d242c1-394a-452b-9327-91505b27fe2c | Ubuntu.14.04.2014.10.002[DEPRECATED] | ACTIVE |                     |
| b4955b1e-1207-4890-a1d2-079c48ecaf7b | rhel.6.5.2014.07.001[DEPRECATED]     | ACTIVE |                     |
| 1ab3b6a9-3c11-4a35-9a04-34aaa421d745 | rhel.6.5.2014.10.002[DEPRECATED]     | ACTIVE | c654-...739f6fd181  |
| 7a84e8ab-562f-4729-ba70-af1e6d4e006b | rhel.7.0.2014.07.001[DEPRECATED]     | ACTIVE |                     |
| b59c9ea0-a4ce-4a01-89c8-377979ba97fc | rhel.7.0.2014.10.002[DEPRECATED]     | ACTIVE | fc1fe-...7e8615d5d  |
+--------------------------------------+--------------------------------------+--------+---------------------+
```

You will note a number of images are tagged as [Deprecated]. Although these are still functional (listed as active), it is recommended that you use the most recent, current versions available. For the purpose of this lab exercise, we suggest you use Ubuntu-Server-14.04. Make a note of the ID of the image that you want to use for your VM.

Next, choose the type of security group to use for your VM. A security group determines inbound access rules for your VM. To find out available security groups:

```
$ nova secgroup-list
```

```
+--------------------------------------+---------+-------------+
| Id                                   | Name    | Description |
+--------------------------------------+---------+-------------+
| bcfca464-aab6-47ec-83a4-54318db94b7a | default | default     |
+--------------------------------------+---------+-------------+
```

You can check the access rules of the ‘default’ security group, run:

```
$ nova secgroup-list-rules default
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | -1      | 0.0.0.0/0 |              |
| tcp         | 3389      | 3389    | 0.0.0.0/0 |              |
| tcp         | 22        | 22      | 0.0.0.0/0 |              |
| tcp         | 80        | 80      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

Initially we only have the ‘default’ security group defined, but it allows the initial system inputs we need, so we’ll just use that.

Now lets check that we have a keypair available:

```
$ nova keypair-list
```

```
+---------+-------------------------------------------------+
| Name    | Fingerprint                                     |
+---------+-------------------------------------------------+
+---------+-------------------------------------------------+
```

We will need to upload one (or have the system generate it for us): In this case, we will upload a public key from our system:

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub default
```

Now we see that there is a keypair available for mapping to our VM. This is important as otherwise, we would have to reduce the security of the deployed VM by forcing a password onto the system default user, which is not a good idea. Again run:

```
$ nova keypair-list
```

```
+---------+-------------------------------------------------+
| Name    | Fingerprint                                     |
+---------+-------------------------------------------------+
| default | bc:7c:26:df:19:7e:68:95:b7:02:b7:9a:3a:62:3e:e9 |
+---------+-------------------------------------------------+
```

We are now ready to create a VM. You will specify the ID or name of the VM type (--flavor), ID or name of VM image (--image), ID of a network (net-id=) as we have multiple tenant accessible networks, and keypair name (--keyname) that you have determined. Replace [my-first-vm-name] with the name you would like to associate with your VM. It will make your life easier if you use a unique name for your VM (and will simplify DNS configuration when the DNSaaS solution becomes available later in 2015)

```
$ nova boot [my-first-vm-name] --flavor standard.small --image
d2b830be-37df-4fa9-90b2-91c472d19aaa --security-groups default --keyname  default --nic
net-id=1cbcddcf-3a7d-481f-b6f2-a97c6447c925
```

To verify that the VM has been successfully created and launched, again run:

```
$ nova list
```

You should now see [my-first-vm-name] listed. The name trusty is shown in the example below:

```
+--------------------------+--------+--------+------------+-------------+-------------------------------------------+
| ID                       | Name   | Status | Task State | Power State | Networks                                  |
+--------------------------+--------+--------+------------+-------------+-------------------------------------------+
| 9934f91e-...-9ebf8e783d24 | trusty | ACTIVE | None       | Running     | default-network=192.168.0.23, 71.74.180.214 |
+--------------------------+--------+--------+-----------+------------+------------------------------------------+
```

Although you initially just want to verify that the VM's status is 'active' and power state is 'running,' you can run the following command to get more detail about your new VM.

```
$ nova show [my-first-vm-name]
```

Your VM power state is running after it is created (assuming that all went well). When you stop a running VM, it gets shut down completely but still persists on disk (whether deployed with an ephemeral disk or booting off of a cinder attached volume). When you suspend a VM, it is temporarily frozen, and can be restarted from the suspended state any time. In both cases, the VM instance still exists in OpenStack. When your run nova-list after any one of these commands, the status of the VM should reflect the new state.

To stop a VM:

```
$ nova stop [vm-name]
```

To suspend a VM:

```
$ nova suspend [vm-name]
```

If you want to delete the instance of a running, stopped or suspended VM, use the command:

```
$ nova delete [vm-name]
```

In order to access the VM, we'll need to map a Floating IP to the VM. This assumes that you've selected the default-network as the target for your VM deployment, otherwise, there is additional configuration needed for the network in order to allow Floating IPs to be mapped.

Start by running:

```
$ nova list
+--------------------------------+--------+--------+------------+-------------+--------------------------------------------+
| ID                             | Name   | Status | Task State | Power State | Networks                                   |
+--------------------------------+--------+--------+------------+-------------+--------------------------------------------+
| 9934f91e-e63a-...-9ebf8e783d24 | trusty | ACTIVE | None       | Running     | default-network=192.168.0.23, 71.74.180.214 |
+--------------------------------+--------+--------+------------+-------------+--------------------------------------------+
```

You will then need to run the following command using the default-network IP address:

```
$ neutron port-list | grep 192.168.0.24
```

You should get a result similar to the following:

```
| 58fd70e4-8e63-4b46-b880-2dc5fe00e10c |      | fa:16:3e:5b:fa:ca | {"subnet_id":
"be07929a-0bea-4ea7-ade3-89dd252a467f", "ip_address": "192.168.0.24"} |
```

This will give you the port ID you will need to use. Now run:

```
$ neutron floatingip-list
```

This list is initially empty as you have not allocated floating IPs

```
$ neutron floatingip-create Ext-Net

Created a new floatingip:
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 71.74.179.226                        |
| floating_network_id | 83eabd71-839c-4ae4-bc51-830436b632ee |
| id                  | c17e9b0c-85a8-4c50-add2-05144dc9d96b |
| port_id             |                                      |
| router_id           |                                      |
| status              | DOWN                                 |
| tenant_id           | 7779a8fa71df44619f71d4fdb273ee04     |
+---------------------+--------------------------------------+

$ neutron floatingip-associate [insert floating IP ID] [insert nova host port ID]
```

Now see if anything has changed when you again run:

```
$ neutron floatingip-list
```

Now that the VM has a floating IP, it should be possible to ssh directly to the instance:

```
$ ssh root@{FLOATING_IP}
```

Please login as the user "ubuntu" rather than the user "root".

This should actually not allow you to log in, but will tell you the name of the default user, so we can use that to log in now:

```
$ ssh ubuntu@71.74.180.214
```

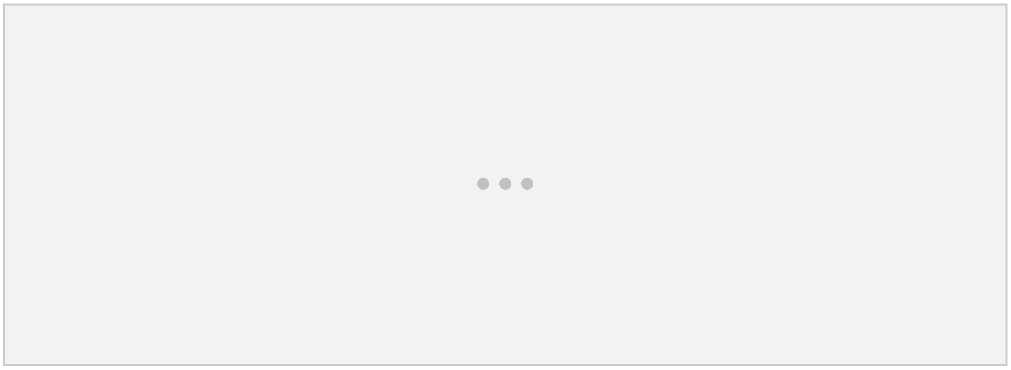
You've now brought your first VM online, and have successfully logged in!

Exercise 3.2 Deploy a VM using Horizon

Now try to recreate the creation of a VM using the GUI provided by Horizon. Horizon provides automation of some of the steps needed to create a VM with the cli. Can you figure out which ones?

Exercise 3.3 Now for something duplicate…

You have two regions available to deploy applications into, NCE and NCW. By default, VMs will be created in the region that is either set in Horizon (if you are using the UI), or in your openrc.sh script (or equivalent) from the command line. If you want to point at NCW instead of NCE (which appears to be the default), in Horizon, you toggle the location in the pulldown at the top of the screen:



In the openrc.sh script, the differences are both in the AUTH_URL and in the REGION:

```
$ diff nce.sh ncw.sh

33c33
< export OS_REGION_NAME="NCE"
---
> export OS_REGION_NAME="NCW"
```

If you configure two scripts, one for each region, you can then easily switch back and forth between them as needed. Since your password doesn't change, and it's never a good idea to store your password in a file, you might actually create three scripts:

1) A script to configure the base parameters, and to ask you for your password (to store it as an environment variable.
2) A script to set NCE region and Auth
3) A script to set NCW region and Auth

The easiest approach to this is to copy the Horizon provided openrc.sh to something like default-openrc.sh (perhaps with NCE credentials), and then create the following two files:

```
cat > nce.sh <<EOF
#!/bin/bash
export OS_REGION_NAME="NCE"
EOF

cat > ncw.sh <<EOF
#!/bin/bash
export OS_REGION_NAME="NCW"
EOF
```

Now, you can be working in NCE, and create new VM in NCW:

Try it!

Exercise 3.4 Deploy an application on your VM with the cli

Implementing this tiny but hugely useful HTTP server is very simple, its just a single line command.

Assume that I would like to share the directory `~/class_dir` and my IP address is the floating IP that you assigned.

Paste the following into the Terminal on the VM:

```
cat >> ~/class_dir/index.html <<EOF
<html>
<body>
Hello world
</body>
</html>
EOF
```

Then type:

```
$ cd ~/class_dir
$ python -m SimpleHTTPServer 80
```

Question: Why is it important that we used port 80 here?
That's it! Now your http server will start in port 80. You will get the message:

```
Serving HTTP on 0.0.0.0 port 80 ...
```

Now open a browser and type the following address:

```
http://{floating_ip}
```

If the directory has a file named `index.html`, that file will be served as the initial file. If there is no `index.html`, then the files in the directory will be listed.

If you wish to change the port that's used start the program via:

```
$ python -m SimpleHTTPServer 8080
```

Exercise 3.5 Deploy an application on your VM with Horizon

Horizon has access to the virtual machine console, from which you can see the current logs that are sent to the console (if any). How would you log into the VM from this console (it has a login capability for most cloud images)?

Hint: There is no password for the default users on the system.

Exercise 3.6 Now add cloud-init and use a simple script to deploy the same app

Cloud-Init is a powerful tool that allows a script (or yaml declarataion, or both, or more than one) to be passed to the VM on boot to accelerate initialization of configuration. OpenStack uses Cloud-init whether you do or not, in order to pass in the ssh 'key-pair' public key and register it as an authorized user for both the “default” user and for the root user (along with that curt message to those who would attempt to log in as root!).

We can extend the login process of our VM to run the same simple Web Application as we did manually (and in future labs we will use this to build much more complicated environments). At it’s simplest we can create a file as follows:

```
cat > user-data.txt <<EOF
#!/bin/bash

mkdir ~/class_dir
cd ~/class_dir
cat >> ~/class_dir/index.html <<CIEOF
<html>
<body>
Hello world
</body>
</html>
CIEOF

nohup python -m SimpleHTTPServer 80

EOF
```

This file can then be used as a part of the nova boot process to pass this “user data” to the VM on boot, and while we still need to manually associate a floating IP as we did before, we should be able to the directly point our web browser at the floating IP to get our simple web page!

Section 4 Debugging VM Startup (from an end user perspective)

Currently debugging VM startup issues is a challenging proposition as there is essentially no documentation on the subject and there is little in the way of logging that will provide insight into where the process failed. The first step is always to double that check the parameters you specified when creating the VM in the first place were correct. If you feel certain that all is in order with the request, it is time to dig a bit deeper into the environment that you were trying to place the VM.

- SSH key mapping
  - Did you select the ssh key that maps to the private key on your current machine?
  - Did you pass a security key name on booting the VM?

- Security group configuration
  - Does your application need ports other than 22, 80, 443 and 3398 (and did you get an except from InfoSec for your additional port in the upstream firewall)?
  - Did you add 443 to the default security group, or create another with that port available (it’s not part of the default group)
- floating IP mapping
  - Did you map your floating IP?

What region did you spin the instance up in?

- Is your openrc file pointing to that region (echo $OS_REGION_NAME can help), or is horizon pointing to the correct region endpoint?

- Network interface mapping
  - The order of networks passed to the VM is important. Did you map the newtorks in the appropriate order (especially important in determining which network to map a floating IP to)
  - Did you configure your second interface to be "up" in the VM operating system (they are usually not configured even if you map a port to them via OpenStack)

Section 5 Extra Credit

Exercise 5.1 Creating an Instance

a) instance with volume to boot
b) instance with an extra volume (what do you need to do to make it accessible to the vm)
c) instance with 2 networks
d) what do you need to do to the instance to communicate
e) can you set up a second instance and have the two systems talk
f) what security group configuration is needed for the second instance?

Exercise 5.2 Create a new security group

Create a security group with an extra port available. What restrictions can you place on this "interface"

Exercise 5.3 Networks

a) with two interfaces, how do you specify the second interface
b) how is interface ordering specified?