# SSE 692

# Engineering Cloud Applications

# Project #2

by

Jason Payne

March 14, 2016

# TABLE OF CONTENTS

| Topics Covered | Topic Examples |
|---|---|
| Cloud-based Development | • OpenStack / DevStack<br>• libcloud SDK<br>• Horizon Dashboard |

# 1. Cloud-based Development

## 1.1 OpenStack

OpenStack is an open-source cloud infrastructure that allows for many different types of configurations based on custom needs.

## 1.2 DevStack

DevStack is a simple, pre-configured OpenStack setup that is ideal for development use because it allows an OpenStack service "cloud" to quickly be setup for testing code and applications.

## 1.3 System Configuration

In production environments, a typical on-site cloud-based system configuration would be composed of one or more servers (physical and/or virtual) offering memory, hard drive space, or some combination of these and other resources. The cloud software (or operating system) would be installed on the "host" server and the other cloud services (such as Horizon, Neutron, Cinder, etc. for OpenStack) could be installed on the same host server or on separate servers. There is no "one-size-fits-all" configuration with the cloud, but that is by design since it is intended to be extremely flexible and configurable to meet individual needs.

The system configuration that will be used for this project is as follows:

- OpenStack/DevStack host – VMware Workstation virtual server loaded with Ubuntu Server 14.04.3 LTS (x64), configured with 3 GB of RAM, 1 x 4-core CPU, and 60 GB HDD space (Figure 1). ***NOTE:*** *The initial configuration specified 2 GB of RAM and 20 GB of HDD space, but it was quickly discovered that this was not enough resources for the cloud virtual images to run successfully or efficiently.*

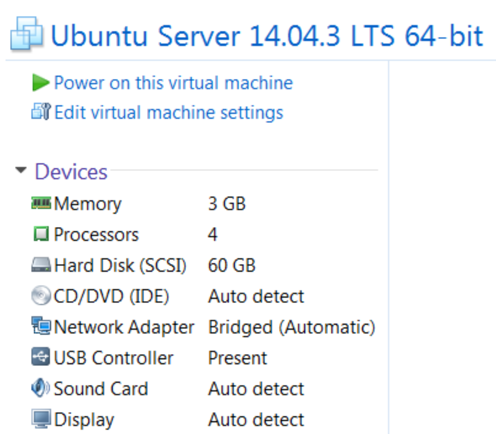



Figure 1: OpenStack/DevStack virtual machine

- Developer Client #1 – virtual image loaded with Windows 7 Professional (x64) and development/production tools. This is primarily used for testing purposes only.

- Developer Client #2 – physical system loaded with the Ubuntu-based Zorin OS 9 and development/production tools. All of the actual development work will be done on this system as a proof-of-concept measure and to avoid exhausting the resources of the physical host machine hosting the OpenStack server.

## 1.4 Installation & Setup

Once the Ubuntu Server host is successfully setup, the next step is to install and setup OpenStack/DevStack on the virtual server. Ideally, the VM instances created in the cloud (Nova guests) would utilize KVM-based virtualization rather than QEMU-based virtualization. KVM virtualization allows nested virtualization (the ability to run KVM on KVM) which is more efficient and responsive than QEMU virtualization which uses less efficient emulation techniques. By default, Linux kernels are not configured for KVM virtualization (Figure 2) so it must be enabled. For this project, KVM virtualization will be configured and utilized as shown in the following sections.

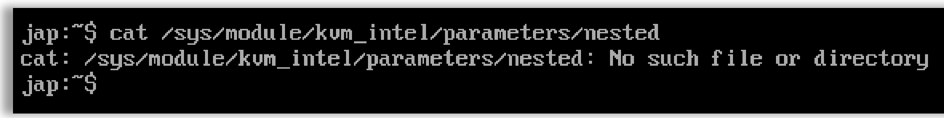



**Figure 2: KVM-based virtualization showing as not enabled**

### 1.4.1 Configuring DevStack for Nested Virtualization

The easiest way to configure the DevStack VM for nested virtualization is to enable it from the VM's configuration settings (Figure 3).

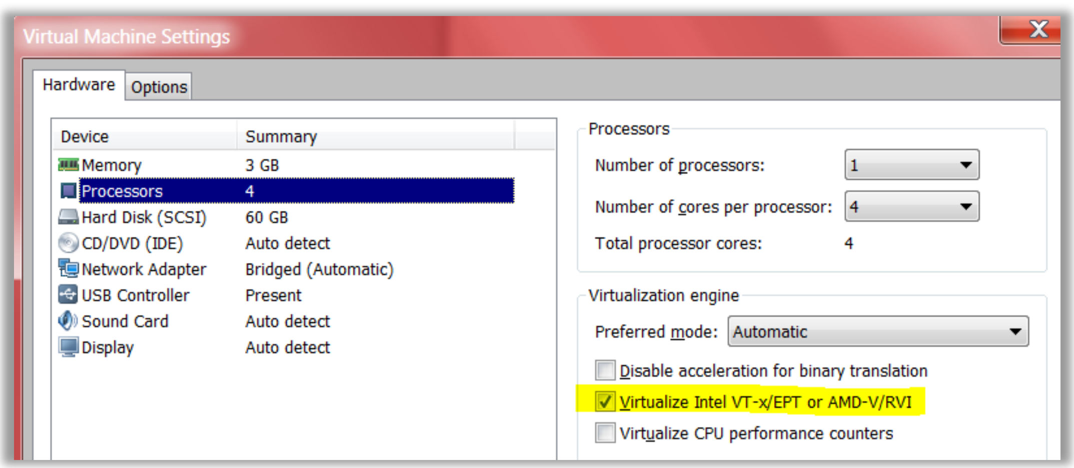



**Figure 3: Enabling the DevStack VM for KVM virtualization**

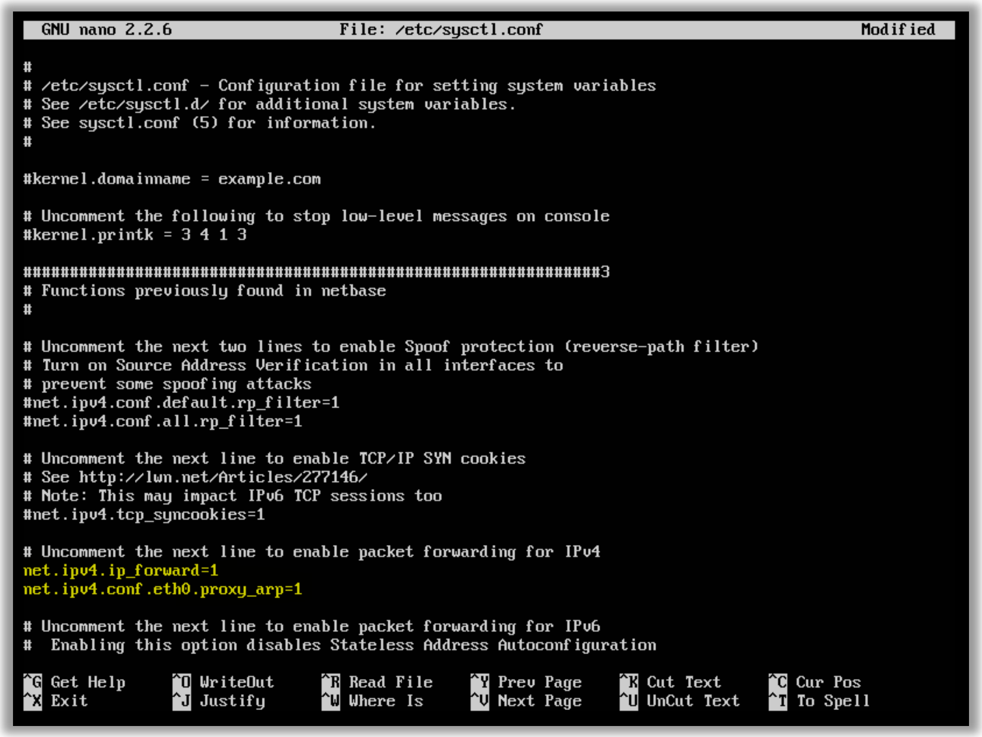
After booting up the server and running the previous command, KVM virtualization should be enabled (Figure 4).

```
jap:~$ cat /sys/module/kvm_intel/parameters/nested
Y
jap:~$ _
```

Figure 4: KVM-based virtualization showing as enabled

### 1.4.2 Setup DevStack Host Networking Environment

Enabling the highlighted properties in Figure 5 will ensure that network traffic is correctly routed to and from the DevStack VMs. Adding these commands to the `/etc/sysctl.conf` file will ensure that the settings persist between reboots of the host.



```
  GNU nano 2.2.6                  File: /etc/sysctl.conf                        Modified

#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
net.ipv4.conf.eth0.proxy_arp=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Figure 5: /etc/sysctl.conf file configured for proper networking

After the properties are enabled, they should be enforced by the IPv4 table administrator tool provided by Linux kernels, iptables, then have the system reboot:

```
:~$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
:~$ sudo reboot
```

### 1.4.3 Download DevStack

The next step is to download the latest reliable version of the DevStack code from git:

```
:~$ sudo apt-get install git -y
:~$ git clone https://github.com/openstack-dev/devstack.git
:~$ cd devstack
```

```
jap:~$ git clone https://github.com/openstack-dev/devstack.git
Cloning into 'devstack'...
remote: Counting objects: 32874, done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 32874 (delta 16), reused 0 (delta 0), pack-reused 32844
Receiving objects: 100% (32874/32874), 11.91 MiB | 4.59 MiB/s, done.
Resolving deltas: 100% (22852/22852), done.
Checking connectivity... done.
jap:~$ ls
devstack  docs
jap:~$ cd devstack/
jap:~/devstack$ _
```

**Figure 6: Cloning latest DevStack code to host**

### 1.4.4 Configure the Installation Script

The `devstack` repository contains an installation script, `stack.sh`, that automates the task of setting up and configuring a local OpenStack configuration on your host. The script allows for certain customizations through the `local.conf` file. For this project, the default file is used along with the following customizations:

```
ADMIN_PASSWORD=password
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=192.168.1.100
FLOATING_RANGE=192.168.1.224/27
FIXED_RANGE=10.0.0.0/24
FIXED_NETWORK_SIZE=256
FLAT_INTERFACE=eth0
```

### 1.4.5 Execute `stack.sh` for Installation

With the DevStack network setup and the local customizations specified, the installation script can be executed for installing and configuring a local instance of the OpenStack/DevStack cloud. This is a long-running script, but a successful run on a decent internet connection should take no more than 10-20 minutes. If there are problems, the log file may provide some clues as to what went wrong. If the installation is successful, the connection settings for accessing DevStack are presented. This should match the settings specified in the local.conf file (Figure 7).

```
This is your host IP address: 192.168.1.100
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.1.100/dashboard
Keystone is serving at http://192.168.1.100:5000/
The default users are: admin and demo
The password: password
```

**Figure 7: DevStack connection settings**

With DevStack successfully installed, the Horizon dashboard (Figure 8) can be accessed from any network machine's web browser.

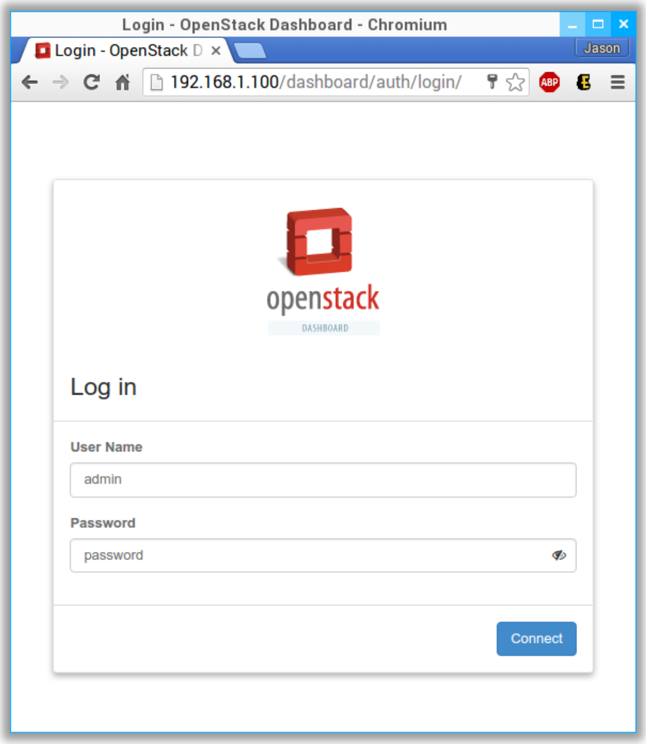
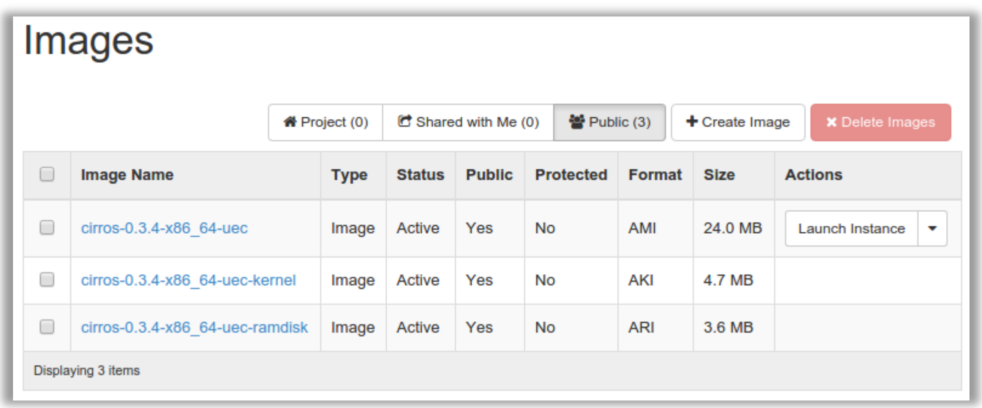



Figure 8: OpenStack's Horizon dashboard UI

After logging into Horizon, the dashboard provides convenient features for viewing available resources (Figure 9, Figure 10), adding images (Figure 11), managing existing instances (Figure 12) and many other common tasks.



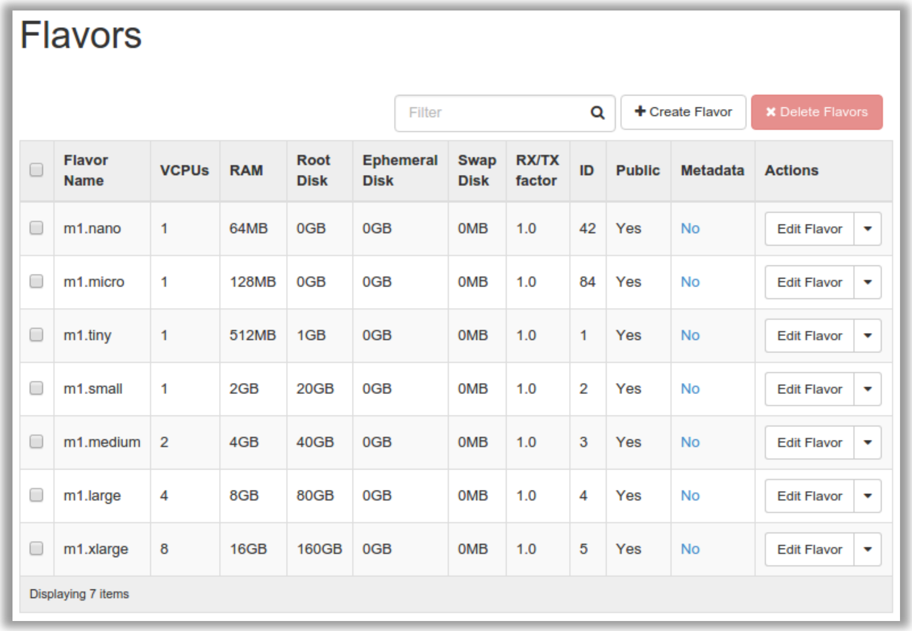
Images

Project (0) | Shared with Me (0) | Public (3) | Create Image | Delete Images

| | Image Name | Type | Status | Public | Protected | Format | Size | Actions |
|---|---|---|---|---|---|---|---|---|
| | cirros-0.3.4-x86_64-uec | Image | Active | Yes | No | AMI | 24.0 MB | Launch Instance |
| | cirros-0.3.4-x86_64-uec-kernel | Image | Active | Yes | No | AKI | 4.7 MB | |
| | cirros-0.3.4-x86_64-uec-ramdisk | Image | Active | Yes | No | ARI | 3.6 MB | |

Displaying 3 items

Figure 9: Test Cirros Public images provided by DevStack



Flavors

| | Flavor Name | VCPUs | RAM | Root Disk | Ephemeral Disk | Swap Disk | RX/TX factor | ID | Public | Metadata | Actions |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | m1.nano | 1 | 64MB | 0GB | 0GB | 0MB | 1.0 | 42 | Yes | No | Edit Flavor |
| | m1.micro | 1 | 128MB | 0GB | 0GB | 0MB | 1.0 | 84 | Yes | No | Edit Flavor |
| | m1.tiny | 1 | 512MB | 1GB | 0GB | 0MB | 1.0 | 1 | Yes | No | Edit Flavor |
| | m1.small | 1 | 2GB | 20GB | 0GB | 0MB | 1.0 | 2 | Yes | No | Edit Flavor |
| | m1.medium | 2 | 4GB | 40GB | 0GB | 0MB | 1.0 | 3 | Yes | No | Edit Flavor |
| | m1.large | 4 | 8GB | 80GB | 0GB | 0MB | 1.0 | 4 | Yes | No | Edit Flavor |
| | m1.xlarge | 8 | 16GB | 160GB | 0GB | 0MB | 1.0 | 5 | Yes | No | Edit Flavor |

Displaying 7 items

Figure 10: Flavors provided by DevStack

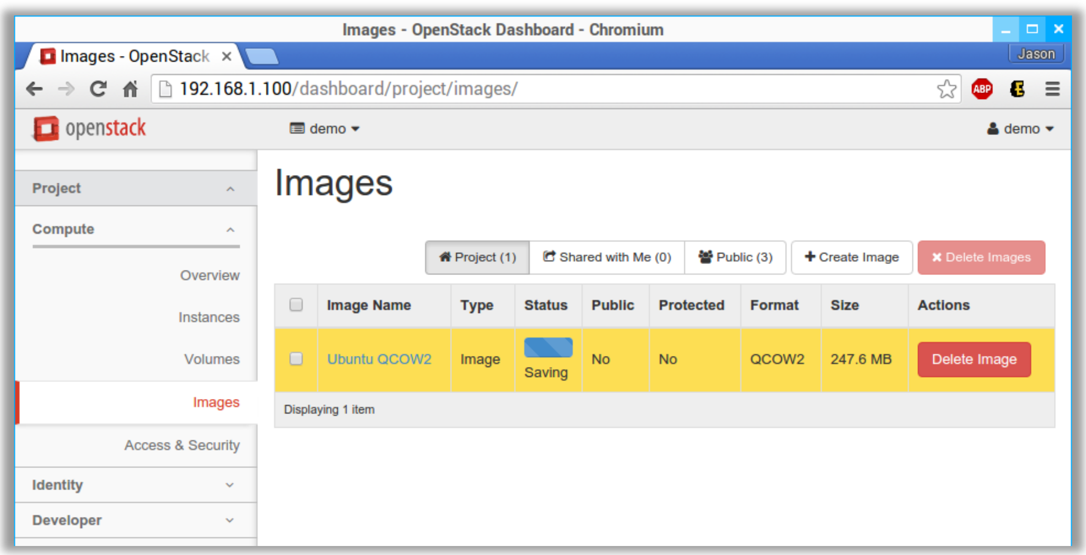



Figure 11: Adding an Official Ubuntu cloud image from Canonical (14.04.4 LTS Trusty Tahr)

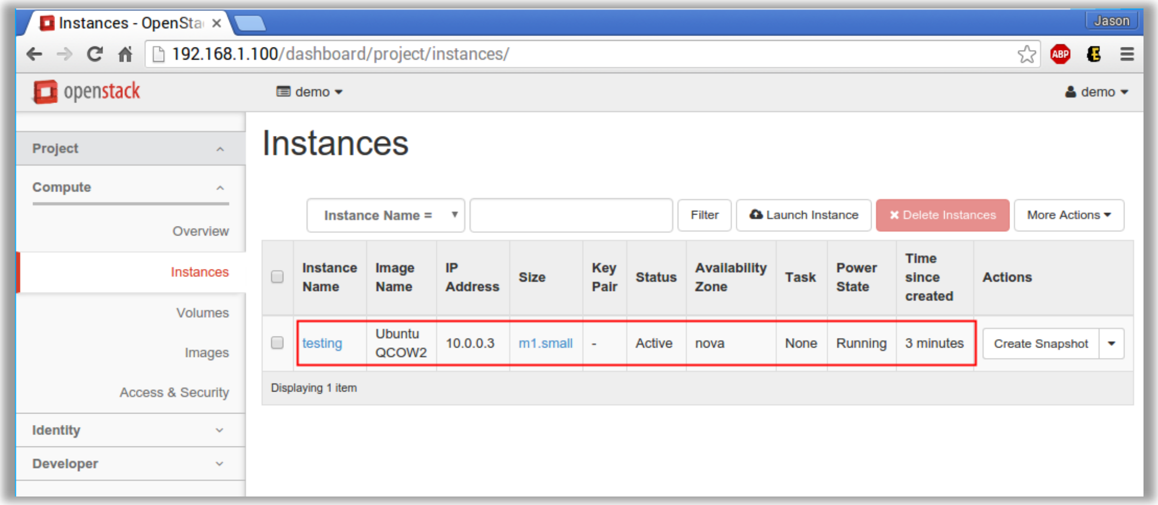



Figure 12: Managing instances within Horizon

# 2. Case Study: Fractal Generator Example

To help learn the intricacies of OpenStack (i.e. cloud-based) development, the OpenStack developers provide a simple fractal generator application that utilizes and demonstrates the usefulness and typical use-cases of the system. In the following sections, this application is exercised and demonstrated through the use of the libcloud API (Python) and then demonstrated by mimicking the same procedures through the Horizon dashboard. This is a really useful way to learn some of the basic capabilities and features of both the API and the OpenStack system.

## 2.1 libcloud API Demonstration

It is worth noting that several APIs exist for different programming languages allowing them to access the OpenStack cloud in code. The libcloud API is a Python-based library that the Apache Foundation manages and works with several other cloud systems other than OpenStack. For this reason, libcloud is widely supported and used in cloud development today.

However, before the script can be run, the first thing to do is to generate the public SSH key that will allow remote access to any created cloud VM instances. Figure 13 illustrates how this is accomplished from the command line interface (CLI).

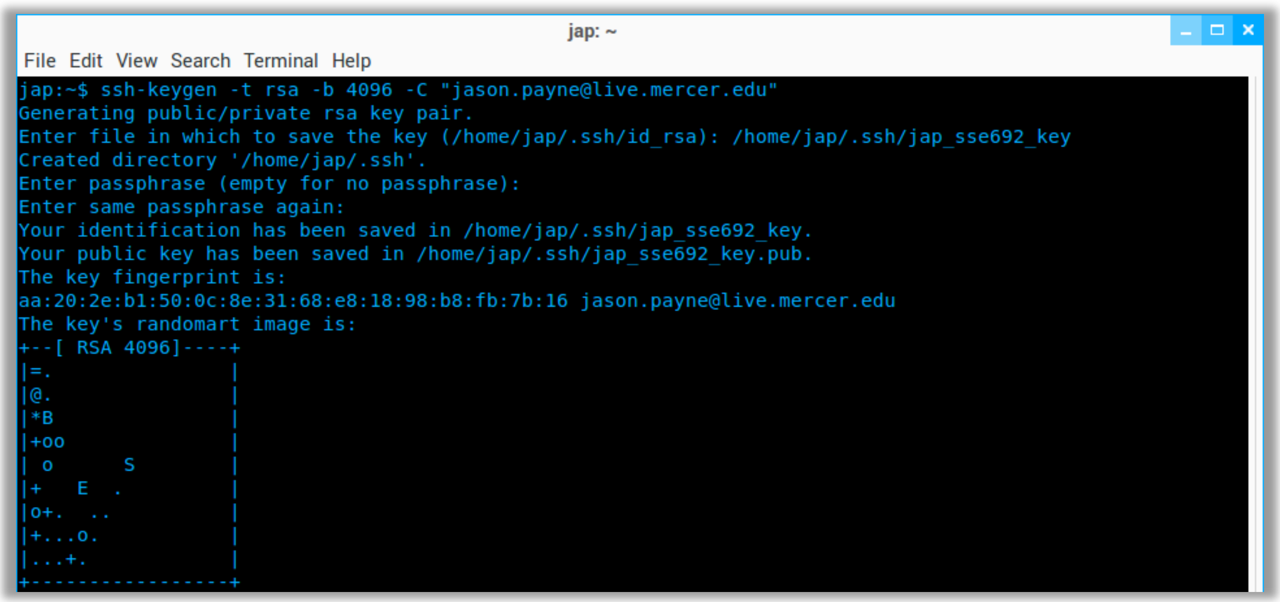



**Figure 13: Generating the SSH key for cloud instance access**

After the SSH key has been created, the latest stable version of the libcloud library can be installed from the Python Installer Package, pip, as follows:

```
pip install apache-libcloud
```

Once this has been installed, the most common scenario is to setup the reference to the cloud provider. From there, a connection is created and that connection is the main access point that the code will use to perform the cloud-based tasks. The full code listing for the application (below) illustrates this point.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 12 17:55:07 2016
@author: jap
"""

# How you interact with OpenStack
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver

auth_username = 'demo'
auth_password = 'password'
auth_url = 'http://192.168.1.100:5000'
project_name = 'demo'
region_name = 'RegionOne'

provider = get_driver(Provider.OPENSTACK)

conn = provider(auth_username, auth_password,
                ex_force_auth_url=auth_url,
                ex_force_auth_version='2.0_password',
                ex_tenant_name=project_name,
                ex_force_service_region=region_name)

# Flavors and images
images = conn.list_images()
for image in images:
    print(image)

flavors = conn.list_sizes()
for flavor in flavors:
    print(flavor)

image_id = '3350a9d0-d655-4de8-9017-30b7bfb40114'
image = conn.get_image(image_id)
print(image)

flavor_id = '2'
flavor = conn.ex_get_size(flavor_id)
print(flavor)

# Launch an instance
instance_name = 'testing'
testing_instance = conn.create_node(name=instance_name, image=image, size=flavor)
print(testing_instance)
```

```
instances = conn.list_nodes()
for instance in instances:
    print(instance)

# Missing line that caused the app to execute successfully!!
conn.wait_until_running([testing_instance])

# Destroy an instance
conn.destroy_node(testing_instance)

# Deploy the application to a new instance
print('Checking for existing SSH key pair...')
keypair_name = 'jap_sse692_key'
pub_key_file = '~/.ssh/jap_sse692_key.pub'
keypair_exists = False
for keypair in conn.list_key_pairs():
    if keypair.name == keypair_name:
        keypair_exists = True
if keypair_exists:
    print('Keypair ' + keypair_name + ' already exists. Skipping import.')
else:
    print('adding keypair...')
    conn.import_key_pair_from_file(keypair_name, pub_key_file)
for keypair in conn.list_key_pairs():
    print(keypair)

print('Checking for existing security group...')
security_group_name = 'all-in-one'
security_group_exists = False
for security_group in conn.ex_list_security_groups():
    if security_group.name == security_group_name:
        all_in_one_security_group = security_group
        security_group_exists = True
if security_group_exists:
    print('Security Group ' + all_in_one_security_group.name + ' already exists. Skipping
          creation.')
else:
    all_in_one_security_group = conn.ex_create_security_group(security_group_name, 'network
                                                        access for all-in-one application.')
    conn.ex_create_security_group_rule(all_in_one_security_group, 'TCP', 80, 80)
    conn.ex_create_security_group_rule(all_in_one_security_group, 'TCP', 22, 22)

for security_group in conn.ex_list_security_groups():
    print(security_group)

userdata = '''#!/usr/bin/env bash
curl -L -s https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh | bash -s -- \
-i faafo -i messaging -r api -r worker -r demo
'''
```

```python
# Boot and configure an instance
print('Checking for existing instance...')
instance_name = 'all-in-one'
instance_exists = False
for instance in conn.list_nodes():
    if instance.name == instance_name:
        testing_instance = instance
        instance_exists = True
if instance_exists:
    print('Instance ' + testing_instance.name + ' already exists. Skipping creation.')
else:
    testing_instance = conn.create_node(name=instance_name, image=image, size=flavor,
                                        ex_keyname=keypair_name, ex_userdata=userdata,
                                        ex_security_groups=[all_in_one_security_group])
    conn.wait_until_running([testing_instance])

for instance in conn.list_nodes():
    print(instance)

# Associate a floating IP for external connectivity
private_ip = None # Test for private IP
if len(testing_instance.private_ips):
    private_ip = testing_instance.private_ips[0]
    print('Private IP found: {}'.format(private_ip))

public_ip = None # Test for public IP
if len(testing_instance.public_ips):
    public_ip = testing_instance.public_ips[0]
    print('Public IP found: {}'.format(public_ip))

print('Checking for unused Floating IP...')
unused_floating_ip = None
for floating_ip in conn.ex_list_floating_ips():
    if not floating_ip.node_id:
    unused_floating_ip = floating_ip
    break

if not unused_floating_ip and len(conn.ex_list_floating_ip_pools()):
    pool = conn.ex_list_floating_ip_pools()[0]
    print('Allocating new Floating IP from pool: {}'.format(pool))
    unused_floating_ip = pool.create_floating_ip()

if public_ip:
    print('Instance ' + testing_instance.name + ' already has a public ip. Skipping
          attachment.')
elif unused_floating_ip:
    conn.ex_attach_floating_ip_to_node(testing_instance, unused_floating_ip)
```

```
# Access the application
actual_ip_address = None
if public_ip:
    actual_ip_address = public_ip
elif unused_floating_ip:
    actual_ip_address = unused_floating_ip.ip_address
elif private_ip:
    actual_ip_address = private_ip

print('The Fractals app will be deployed to http://{}'.format(actual_ip_address))
```

Console Output:

```
<NodeImage: id=3350a9d0-d655-4de8-9017-30b7bfb40114, name=Ubuntu QCOW2, driver=OpenStack ...>
<NodeImage: id=852d803d-2721-41ad-ad38-51dfa85ae3ab, name=cirros-0.3.4-x86_64-uec,
driver=OpenStack ...>
<NodeImage: id=0b877ff2-f342-4e5a-afbd-bcb0751cb5a7, name=cirros-0.3.4-x86_64-uecramdisk,
driver=OpenStack ...>
<NodeImage: id=046e546e-9901-40f7-a365-002a1ce0f559, name=cirros-0.3.4-x86_64-ueckernel,
driver=OpenStack ...>
<OpenStackNodeSize: id=1, name=m1.tiny, ram=512, disk=1, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=1, ...>
<OpenStackNodeSize: id=2, name=m1.small, ram=2048, disk=20, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=1, ...>
<OpenStackNodeSize: id=3, name=m1.medium, ram=4096, disk=40, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=2, ...>
<OpenStackNodeSize: id=4, name=m1.large, ram=8192, disk=80, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=4, ...>
<OpenStackNodeSize: id=42, name=m1.nano, ram=64, disk=0, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=1, ...>
<OpenStackNodeSize: id=5, name=m1.xlarge, ram=16384, disk=160, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=8, ...>
<OpenStackNodeSize: id=84, name=m1.micro, ram=128, disk=0, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=1, ...>
<NodeImage: id=3350a9d0-d655-4de8-9017-30b7bfb40114, name=Ubuntu QCOW2, driver=OpenStack ...>
<OpenStackNodeSize: id=2, name=m1.small, ram=2048, disk=20, bandwidth=None, price=0.0,
driver=OpenStack, vcpus=1, ...>
<Node: uuid=340fd62e2b429bd085183b71dbc2ef519285f56d, name=testing, state=PENDING, public_ips=[],
private_ips=[], provider=OpenStack ...>
<Node: uuid=340fd62e2b429bd085183b71dbc2ef519285f56d, name=testing, state=PENDING, public_ips=[],
private_ips=[], provider=OpenStack ...>
Checking for existing SSH key pair...
adding keypair...
<KeyPair name=jap_sse692_key fingerprint=aa:20:2e:b1:50:0c:8e:31:68:e8:18:98:b8:fb:7b:16
driver=OpenStack>
Checking for existing security group...
<OpenStackSecurityGroup id=6 tenant_id=defc64b1efd14a2eb45a8fdb90323c42 name=all-inone
description=network access for all-in-one application.>
<OpenStackSecurityGroup id=1 tenant_id=defc64b1efd14a2eb45a8fdb90323c42 name=default
description=default>
Checking for existing instance...
<Node: uuid=3cdda06130048ccfc2fa75edfb6fb01067b446ad, name=all-in-one, state=RUNNING,
public_ips=[], private_ips=['10.0.0.18'], provider=OpenStack ...>
Checking for unused Floating IP...
Allocating new Floating IP from pool: <OpenStack_1_1_FloatingIpPool: name=public>
The Fractals app will be deployed to http://192.168.1.225
```

As pointed out in the code listing, the highlighted line that was missing from the example code causes the app to never run successfully when executed from the script. However, since that part of the code is not specifically applicable to the execution of the app itself, commenting out that section or using the highlighted line will allow the app to run successfully.

Once the app is running successfully, the created cloud VM instance can be seen from the Horizon dashboard (Figure 14) and/or direct access into the instance can be achieved through the SSH CLI (Figure 15). The app itself can be accessed from any web browser on the network (Figure 16).

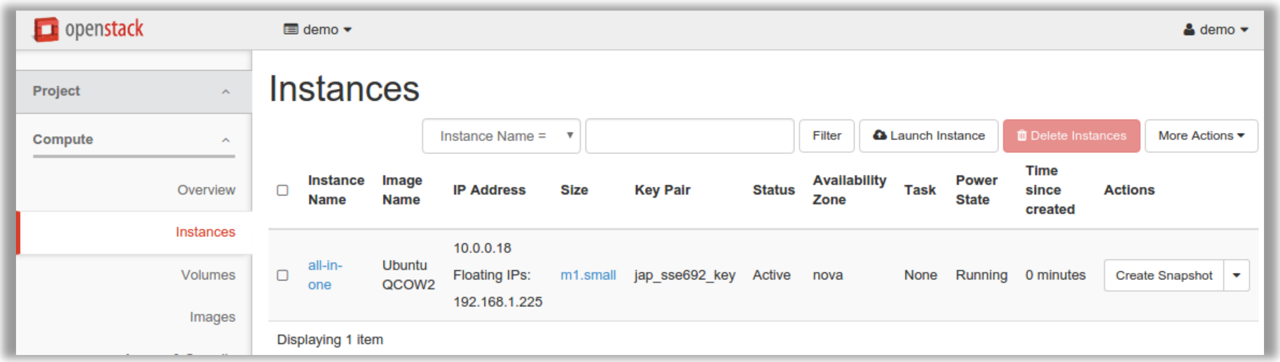
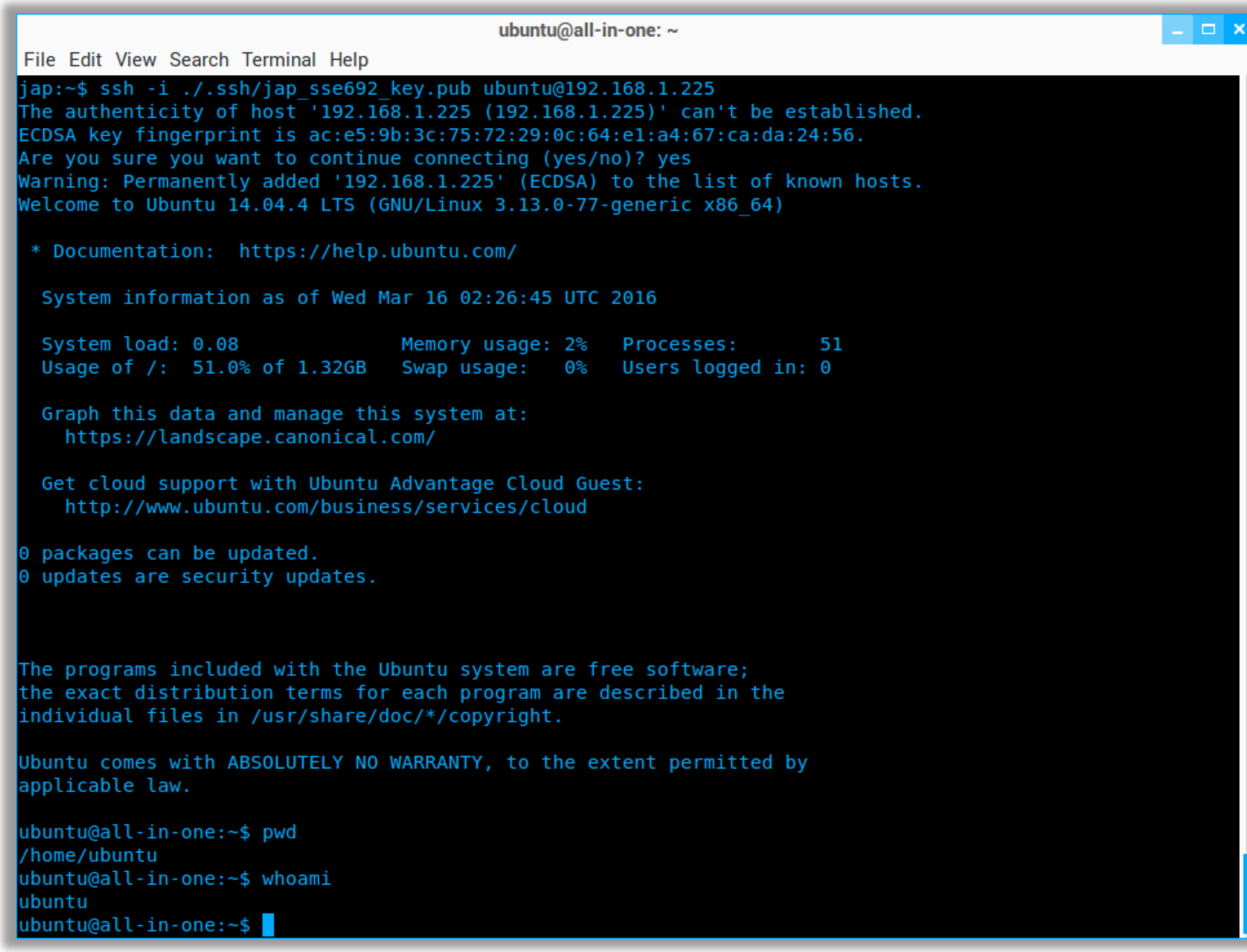



**Figure 14: Ubuntu cloud instance running the fractal application**



```
ubuntu@all-in-one: ~
File Edit View Search Terminal Help
jap:~$ ssh -i ./.ssh/jap_sse692_key.pub ubuntu@192.168.1.225
The authenticity of host '192.168.1.225 (192.168.1.225)' can't be established.
ECDSA key fingerprint is ac:e5:9b:3c:75:72:29:0c:64:e1:a4:67:ca:da:24:56.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.225' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-77-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Wed Mar 16 02:26:45 UTC 2016

  System load: 0.08              Memory usage: 2%   Processes:       51
  Usage of /:  51.0% of 1.32GB   Swap usage:   0%   Users logged in: 0

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ubuntu@all-in-one:~$ pwd
/home/ubuntu
ubuntu@all-in-one:~$ whoami
ubuntu
ubuntu@all-in-one:~$
```

**Figure 15: Accessing the cloud instance through the SSH CLI**

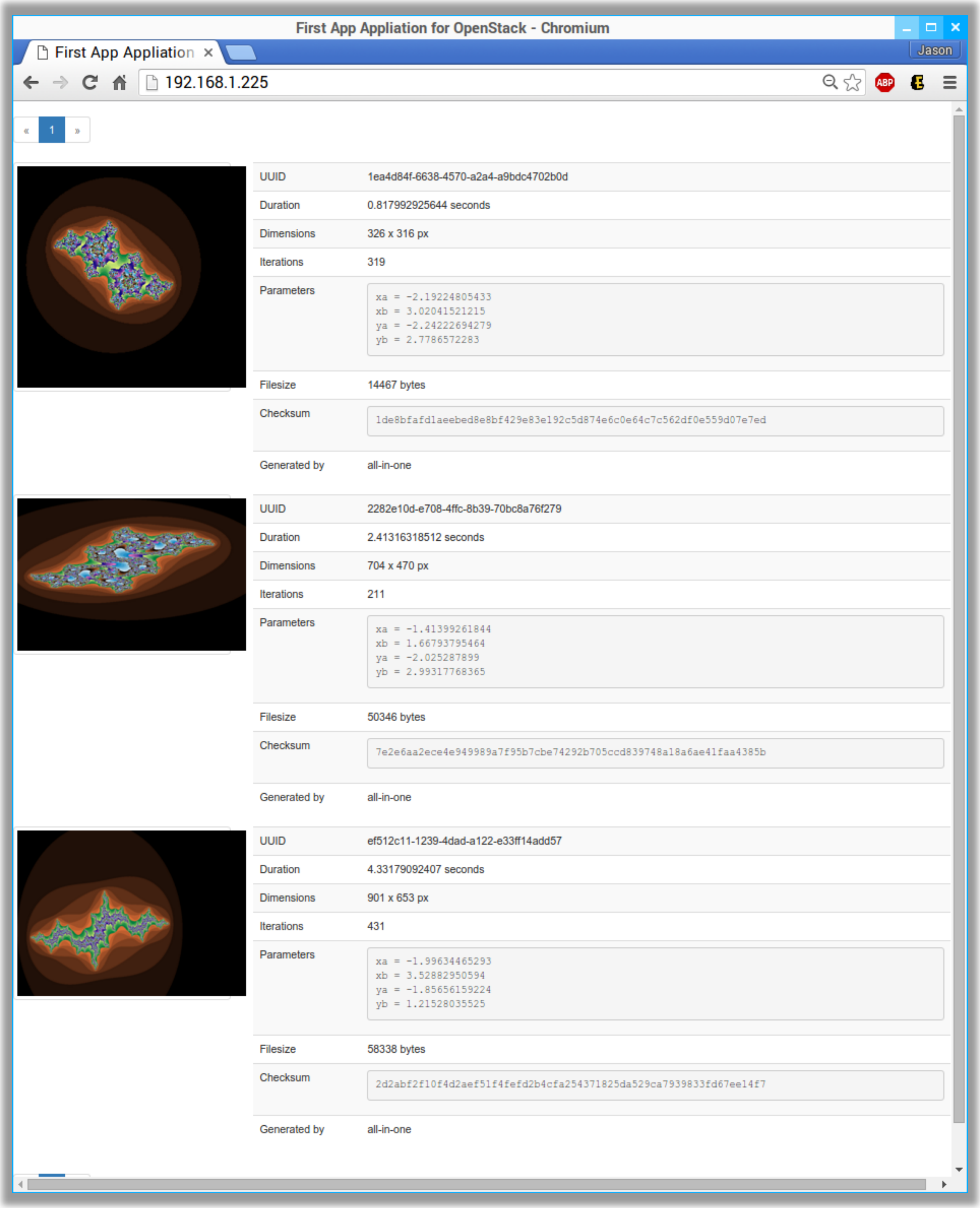



**Figure 16: Fractal Generator app running in a web browser**

## 2.2 Horizon Dashboard Demonstration

The next part of the demonstration is to duplicate the results in the previous section by performing the same tasks through the Horizon dashboard and getting the app to run in a web browser the same way that the script did. With that said, this demonstration assumes that Horizon is being accessed from a default state where no security groups, key pairs, etc. have been setup.

The first step is to navigate to the 'Access & Security' section and create a security group (Figure 17) and add rules that allow SSH communications (Figure 18).

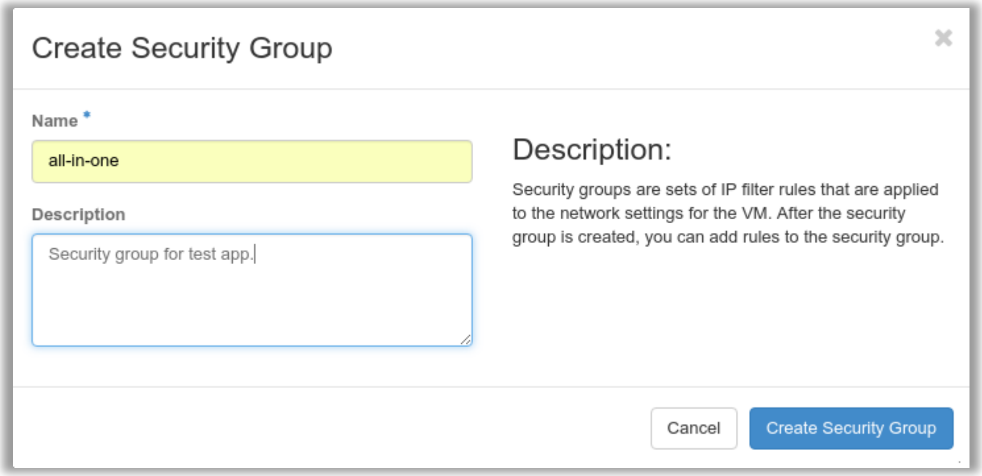



Figure 17: Creating the security group from Horizon

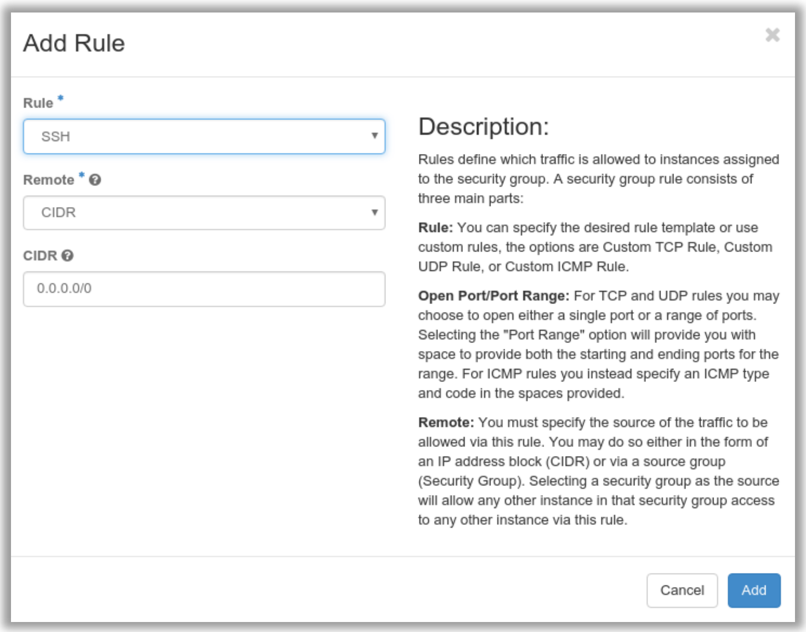



Figure 18: Adding the SSH rule to the security group

The next thing to do is to import the public SSH key that was created earlier into the project (Figure 19).

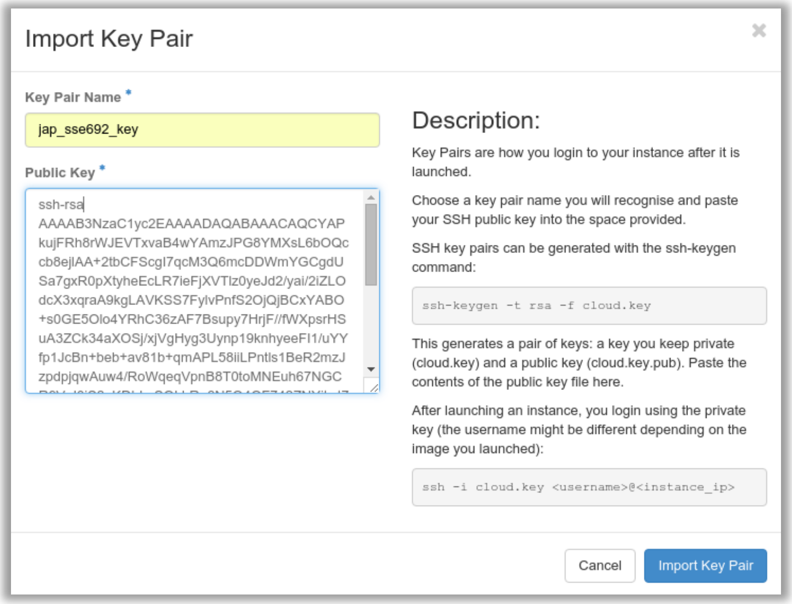



**Figure 19: Importing the SSH public key**

Once the key pair has been created and the rules setup in the security group, the instance can be launched. Navigate to the 'Instances' section and select the 'Launch Instance' button to initiate the 'Launch Instance' wizard that steps through the process of setting up and instantiating a cloud VM instance.

The 'Details' page sets the name of the instance (required) and the number of instances to be created (Figure 20).

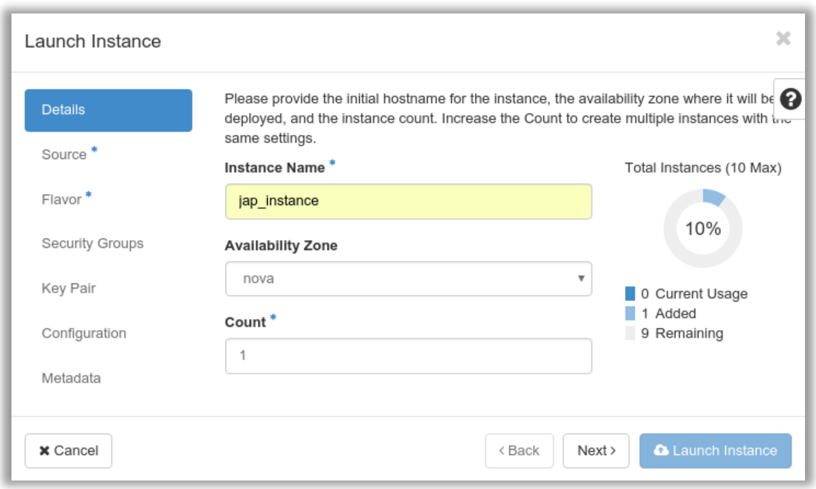



**Figure 20: Details page of the 'Launch Instance' wizard**

The 'Source' page determines the image source (required) to be used for the OS of this instance. For this instance, the official Ubuntu image is used as the OS for the instance (Figure 21).

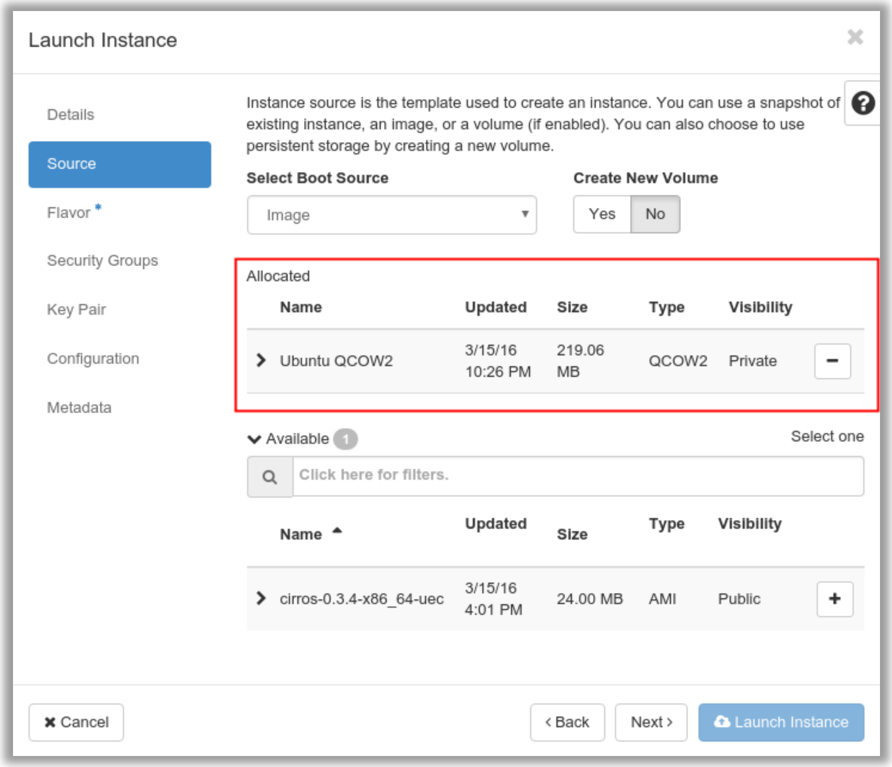



**Figure 21: Source page of the 'Launch Instance' wizard**

The 'Flavor' page sets the resources (RAM, HDD, CPUs) that will be available to this instance. This is typically dependent on the image selected as certain images have minimum requirements. For this instance, since the Ubuntu image is being used, a minimum of 2 GB of RAM and 4 GB of HDD is recommended, so the m1.small flavor is used (Figure 22).

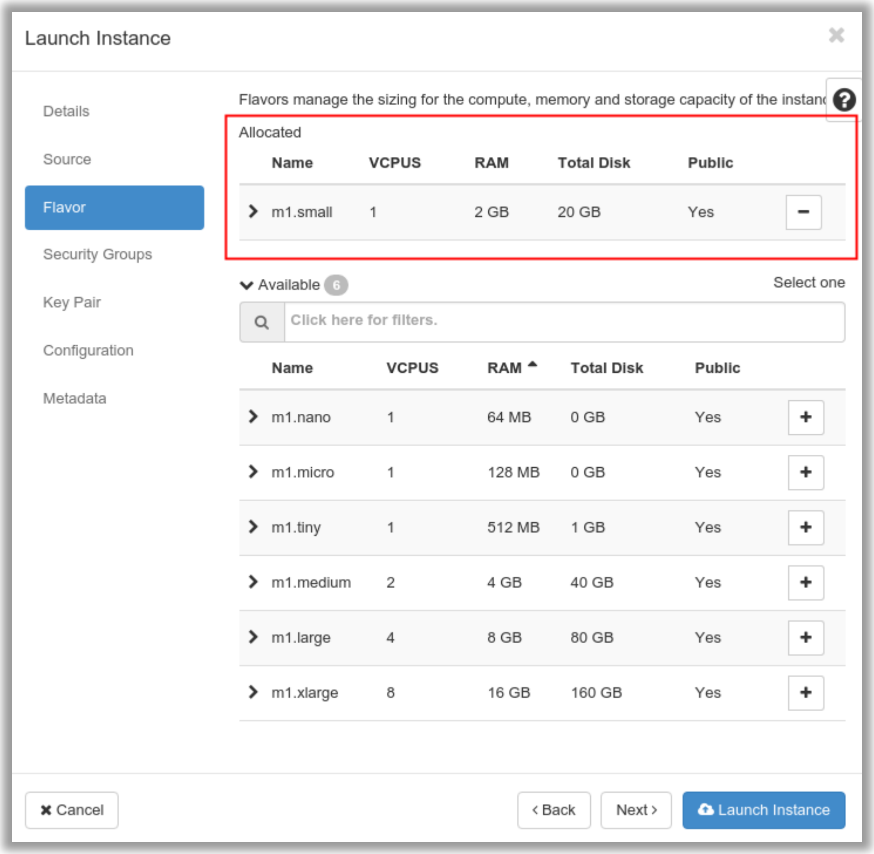



Figure 22: Flavor page of the 'Launch Instance' wizard

The 'Security Groups' page allows a security group to be specified from any available project security groups. The security group, all-in-one, created earlier is used here (Figure 23). This property is optional.

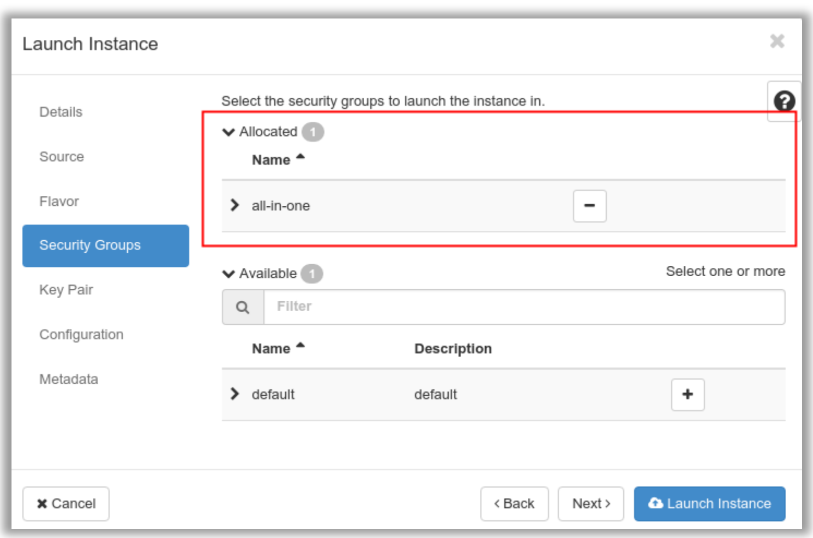



Figure 23: Security Groups page of the 'Launch Instance' wizard

The ‘Key Pair’ page allows the SSH key to be “inserted” into the instance when it is created so that remote access is possible from the SSH CLI. The key generated earlier is used for this instance (Figure 24).

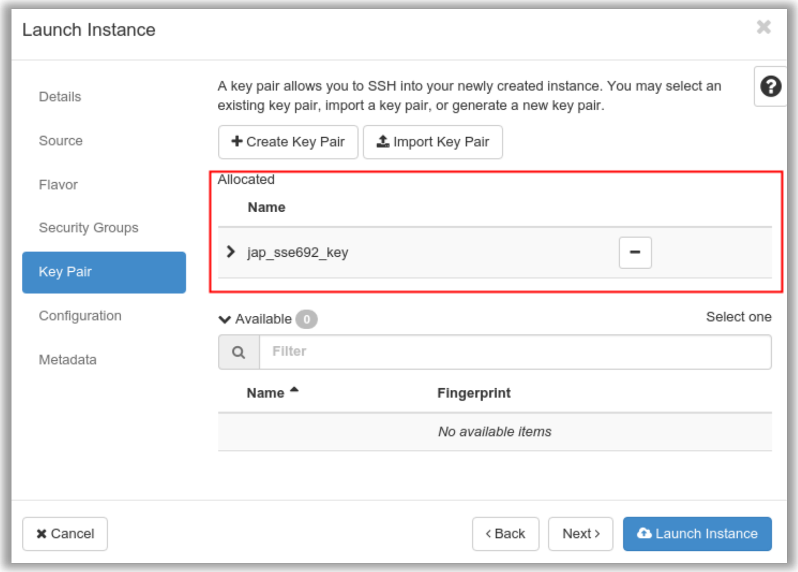



Figure 24: Key Pair page of the ‘Launch Instance’ wizard

The ‘Configuration’ page is critical to the execution of the fractal generator app. The script specified here (Figure 25) will automatically be executed during instance boot by the cloud-init service that is pre-installed on the Ubuntu image. If this is not correct, the application will not run!

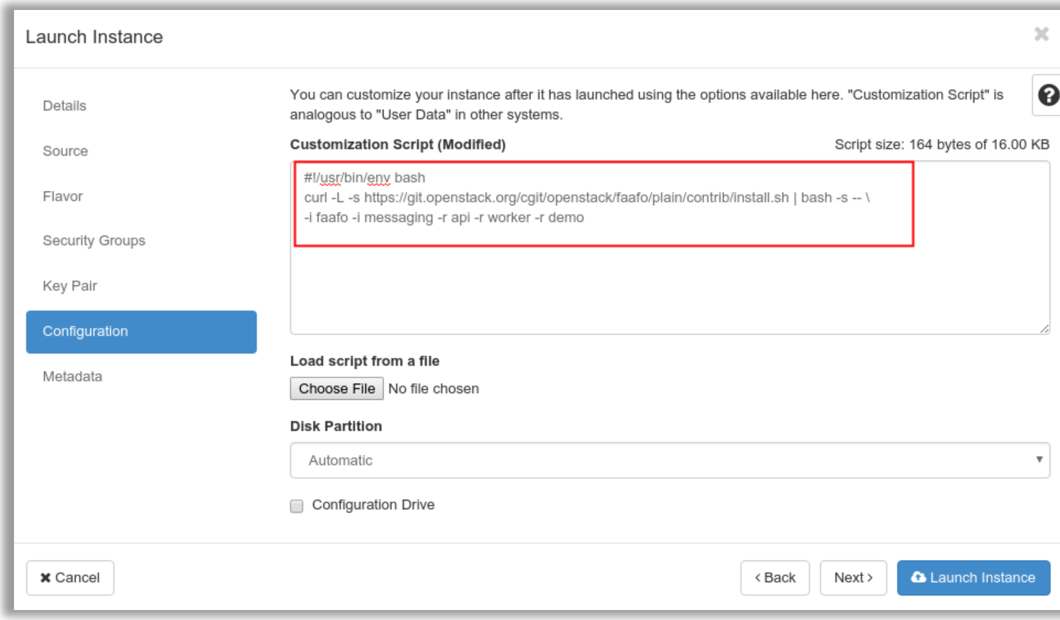



Figure 25: Configuration page of the ‘Launch Instance’ wizard

At this point, all critical properties have been setup for the instance and the ‘Launch Instance’ button can be selected. Once the instance is up and running, a floating IP can be assigned to associated with the instance (Figure 26) which is what allows the app to be accessed from a web browser.

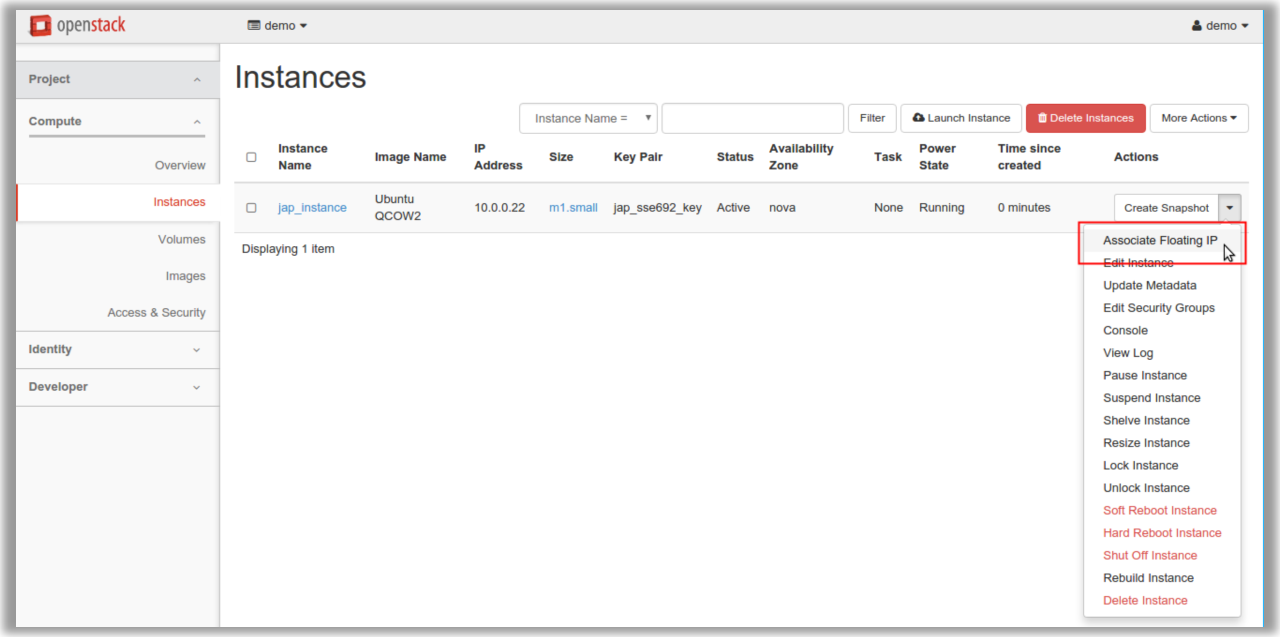



Figure 26: Associating a floating IP to the instance

As with the script-generated instance, the cloud instance can be accessed remotely through the SSH CLI (Figure 27).

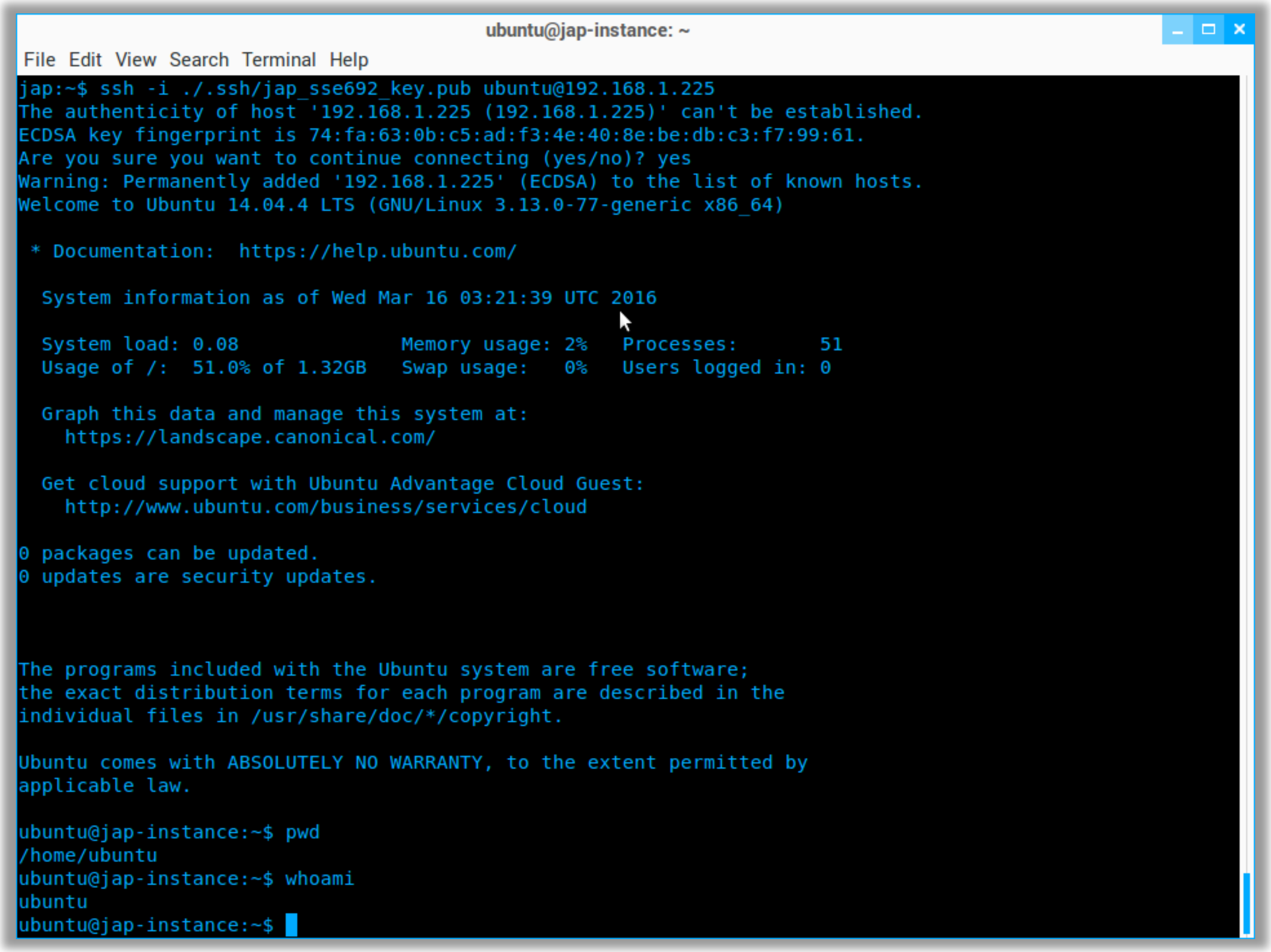



Figure 27: Accessing the Horizon-created cloud instance through the SSH CLI

The running cloud instance can be monitored and managed from the Horizon dashboard (Figure 28). Snapshots of the running instance can be created and other VM-common features can be accessed from the dashboard.

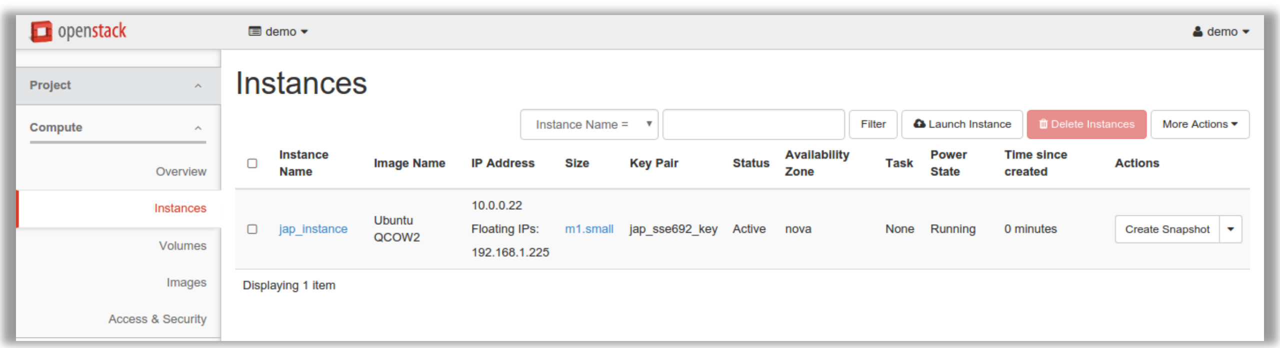



Figure 28: Horizon-created Ubuntu cloud instance running the fractal application

Based on the status of the instance, everything appears to be as it should be. However, attempting to access the app from a browser is still not possible! The problem lies in the configuration set up in the security group for this instance. A rule is needed to allow HTTP communications through port 80 of the instance (Figure 29). Once that rule is created and applied to the instance, the fractal generator app will be accessible from the web browser (Figure 30). The great thing about this is that the rule can be added to the security group and applied to the instance without needing to shut down the instance!

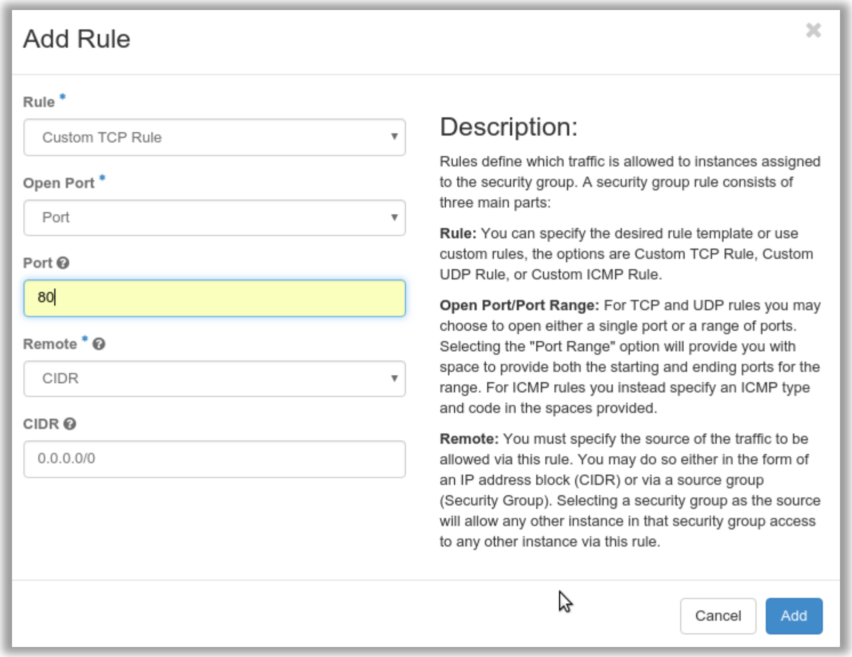



**Figure 29: Adding the HTTP rule to the security group**

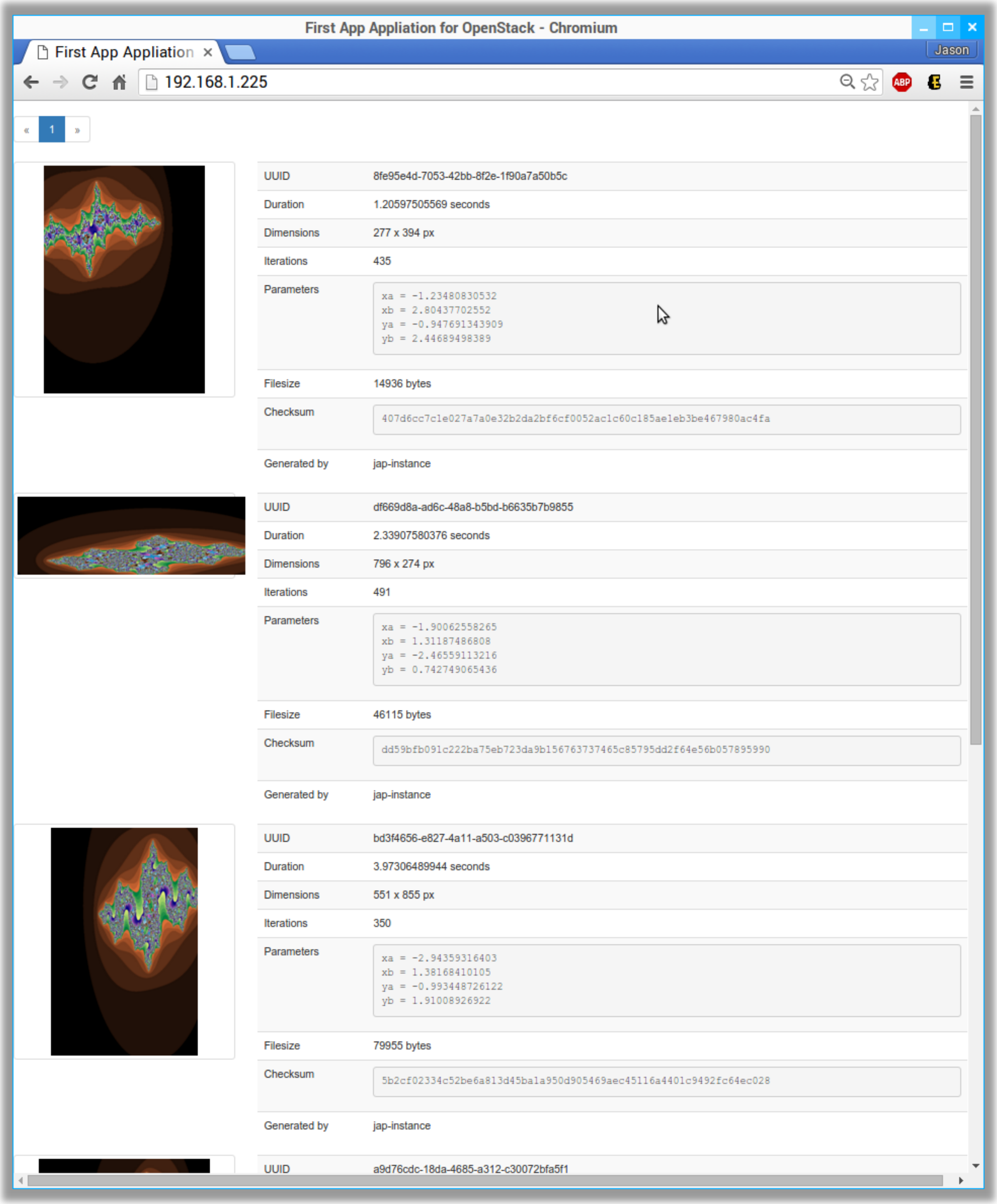



**Figure 30: Fractal Generator app running in a web browser**

# Non-Direct Activity Report

| Date | Duration (minutes) | Specific Task / Activity |
|---|---|---|
| 13-Feb-2016 | 118 | Work on project #2 |
| 14-Feb-2016 | 322 | Work on project #2 |
| 18-Feb-2016 | 489 | Work on project #2 |
| 5-Mar-2016 | 60 | Work on project #2 |
| 6-Mar-2016 | 420 | Work on project #2 |
| 10-Mar-2016 | 120 | Work on project #2 |
| 11-Mar-2016 | 210 | Work on project #2 |
| 12-Mar-2016 | 330 | Work on project #2 |
| 13-Mar-2016 | 600 | Work on project #2 |
| 14-Mar-2016 | 240 | Work on project #2 |
| 15-Mar-2016 | 360 | Work on project #2 |
| 16-Mar-2016 | 600 | Work on project #2 |
| **Sum for Report #1** | **1511** | **/ 1200 (5 weeks @ 300/wk)** |
| **Sum for Report #2** | **3869** | **/ 1500 (5 weeks @ 300/wk)** |
| **Sum for Report #3** | | **/ 1800 (5 weeks @ 300/wk)** |
| **Sum for Class** | **5380** | **/ 4500 (15 weeks @ 300/wk)** |