SSE 692

Engineering Cloud Applications

Project #3

by

Jason Payne

April 29, 2016

TABLE OF CONTENTS

Topics Covered	Topic Examples
Cloud Architecture	• Configuration • Characteristics
Cloud Application Design/Development	• Modularization • Orchestration

1. Overview

This project will demonstrate some of the common concepts utilized by cloud applications to solve specific problems. To accomplish this, the Fractals application that was discussed in the previous project will also be used as the case study for this project as well. This application serves as a prime candidate for some of the topics utilized by cloud applications today thus providing a good mechanism for learning and discussing cloud development techniques and practices.

1.1 System Configuration

In order to maximize the learning experience for this project, the goal is to include a system offering more physical resources in order to run more instances that will be needed by some of the techniques discussed in this demonstration. Another goal is to setup the DevStack host in a KVM-based Nested Virtualization configuration in order to maximize the efficiency of the instances created by the cloud host. The targeted configuration is illustrated below in Figure 1:

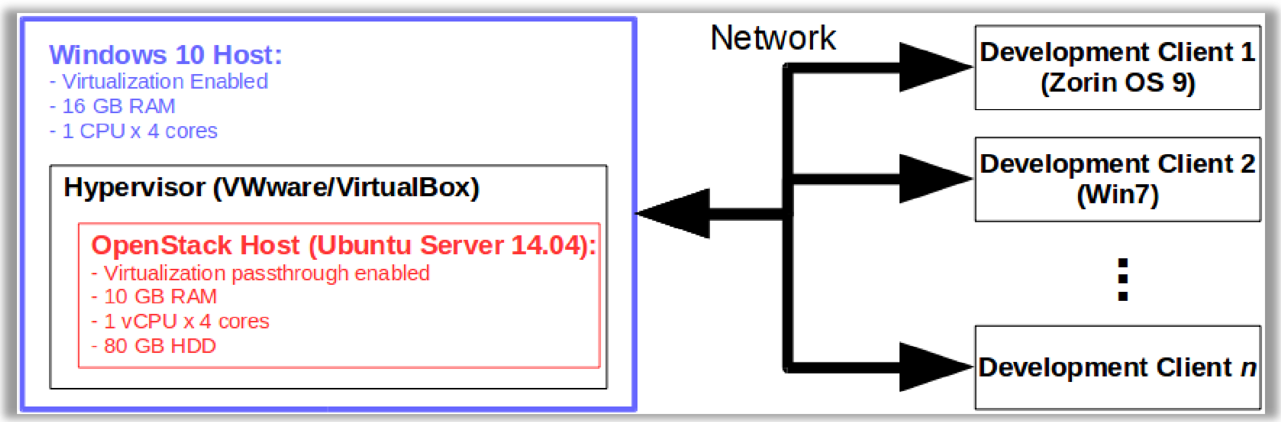


Figure 1: Targeted System Configuration

Configuration Problem #1: Virtualization Pass-Through with VirtualBox

For project 2, the cloud host was successfully setup with a KVM-based Nested Virtualization configuration. This was achieved by using VMware Workstation 9 on a Win7 host and by enabling virtualization pass-through in the VM's settings (i.e. enabling the "Virtualize Intel VT-x/EPT or AMD-V/RVI" option under the 'Processors' settings).

For this project, the same configuration was desired, but with slight changes in hosts and clients due to the desire to increase the physical resources of the OpenStack host machine. Initially, VirtualBox 5.0 was setup on a Ubuntu-based (Linux) host with the intent to serve as the hypervisor and hypervisor host, respectively, for the project configuration. However, during attempts to set this up, it was discovered that VirtualBox is not capable (as of this writing) of supporting the nested virtualization configuration because VirtualBox does not support nested virtualization. This was verified by setting up the OpenStack host's VM to support virtualization pass-through and then checking the status of the nested virtualization capability on the OpenStack host VM, which confirmed that nested (KVM-based) virtualization was not enabled (see Figure 2).

```
jap:~$ cat /sys/module/kvm_intel/parameters/nested
cat: /sys/module/kvm_intel/parameters/nested: No such file or directory
jap:~$
```

Figure 2: KVM-based virtualization showing as not enabled on VirtualBox VM hosting OpenStack

Configuration Problem #2: Removal of rejoin-stack.sh by DevStack Team

When learning about cloud development, a common occurrence is to perform some actions that require the developer to reboot or shutdown their OpenStack host. When this happens, the OpenStack services are also shut down and do not automatically start back up when the host is booted back up again. The problem with this is that, obviously, the OpenStack instance is not accessible after a reboot and the only way to get the OpenStack instance running again is to run the stack.sh script which is a lengthy process. It should also be noted that running the stack.sh script resets the OpenStack instance to its default state which erases any data/instances that were stored within OpenStack. To combat this, the DevStack team provided a shell script that would allow users to start up and rejoin their OpenStack instance where it left off, thereby avoiding the need and hassle of running the stack.sh script.

Unfortunately, within the timeframe of this project, the DevStack team removed the rejoin-stack.sh script due to philosophical concerns related to the intent of DevStack, which – according to the development team – is to serve as a quick way to get OpenStack up and running for demonstration and learning purposes and not as an abbreviated mechanism for deploying a production-level instance of OpenStack. While this topic has merits for argument, the impact for this project is that this leads to long wait times and/or duplicated effort after reboots of the OpenStack host. To combat this, the snapshot feature of VMware was utilized to save the state of the OpenStack host at important times such as after install and after resources had been added to OpenStack.

2. Cloud Application Architecture

The usefulness and resourcefulness of cloud computing and development has created new possibilities into how applications and data are deployed and consumed. This is possible due to design patterns that were formulated well before cloud computing was realized. The sections that follow exercise and demonstrate some of these design concepts.

2.1 Characteristics of Cloud Architecture

To help setup goals for the project, it is good to understand some of the common characteristics of cloud architecture. These characteristics are not new or novel notions within the industry, but they do serve as good reminders as to why a well though-out design can save valuable resources from being spent or exhausted. These are not all of the characteristics of cloud architecture, but it is a good starting point. These characteristics are:

- Modularity & Micro-services – micro-services are an important design pattern that helps achieve application modularity. This is beneficial as it allows different segments of applications to be deployed across different instances (i.e. servers, nodes, etc.). This can also be beneficial in case pieces of application can be reused within other cloud applications (common front-end, calc engine, etc.).
- Scalability – scalability is the concept of using multiple smaller instances to achieve the same results. This enables an application to grow past the limit imposed by the maximum size of an instance.
- Fault Tolerance – since cloud architecture is centralized around the notion of virtualization, there is no longer a dependency on large, expensive servers. Instead, if something goes wrong, the virtual server can be shut down and/or discarded and a new one created in its place in a fraction of the time. Often times this new virtualized server is a clone or replica of the server that had to be shut down or discarded so the server is basically the same server. It is still important to design for environment failures because designing with a high degree of fault tolerance, can make applications and systems resilient and more adaptable to change. Fault tolerance is essential for cloud-based applications.
- Automation – automation decreases recovery time and increases fault tolerance and resilience by automating processes that scale resources up and down to meet demand for an application.

3. Modularity & Micro-Services

The previous project focused on setting up a cloud server with OpenStack and deploying an application in a single all-in-one instance (see Figure 3). For this project, more progress will be made to deploy the same application, but with different design considerations. The first design consideration is a "separation of concerns" using micro-services. The focus here will be on breaking away from the single instance deployment in favor of deploying across multiple instances that have specific design purposes: API controllers for client interactions and workers for calculations.

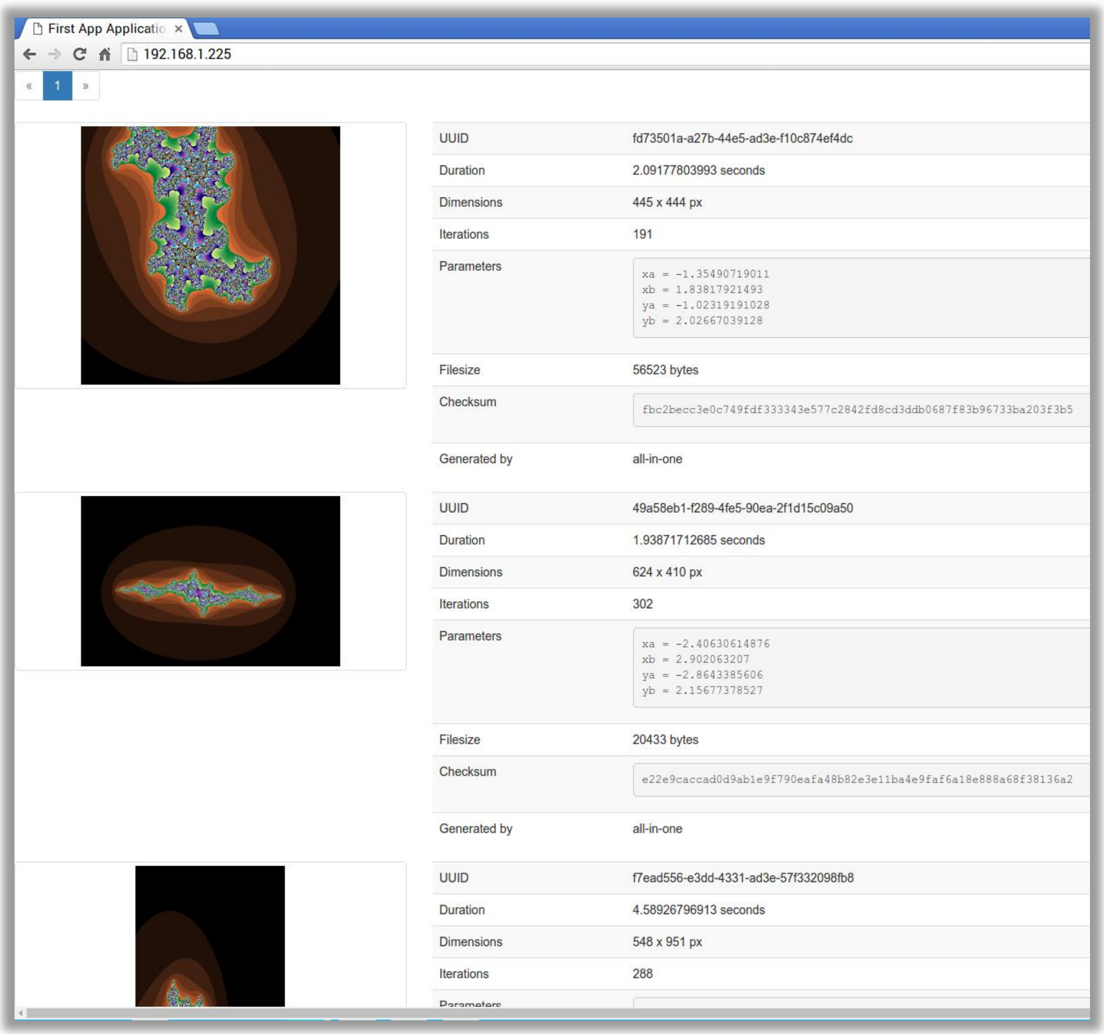


Figure 3: All-In-One Fractals App Instance

3.1 Deployment

The Python script below is similar to the previous deployment script used to deploy the Fractals app as an all-in-one instance. The main difference, however, is that the highlighted portions illustrate how multiple instances and security groups/rules are created. An API controller instance is created with the RabbitMQ and faafo services installed. A separate worker instance is also created with the faafo service installed and its userdata script includes the address of the API controller instance and the message queue so that the worker knows where to “listen” for requests.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 22 00:20:40 2016
@author: jap
"""

# APIs for interacting 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'
keypair_name = 'jap_sse692_key'
image_name = 'Ubuntu QCOW2'

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)

# Setup image and flavor
images = conn.list_images()

# Get the image from its name rather than its complex id.
image = [i for i in images if image_name in i.name][0]
flavor_id = '2'
flavor = conn.ex_get_size(flavor_id)

# Setup access key
print('Checking for existing SSH key pair...')
pub_key_file = '~/.ssh/{}.pub'.format(keypair_name)
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)

worker_group = conn.ex_create_security_group('worker', 'for services that run on a worker node \
                                                       (instance)')
conn.ex_create_security_group_rule(worker_group, 'TCP', 22, 22)

controller_group = conn.ex_create_security_group('control', 'for services that run on a control \
                                                             node')
conn.ex_create_security_group_rule(controller_group, 'TCP', 22, 22)
conn.ex_create_security_group_rule(controller_group, 'TCP', 80, 80)

# Create a rule that applies to only worker group instances.
conn.ex_create_security_group_rule(controller_group, 'TCP', 5672, 5672, \
                                   source_security_group=worker_group)

# For the application instance, have the install script:
# - install the RabbitMQ messaging service (-i messaging)
# - install the Faafo (-i faafo) service
# - enable the API service (-r api)
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
'''

# Create controller instance to host the API, database, and messaging services.
instance_controller_1 = conn.create_node(name='app-controller',
                                         image=image,
                                         size=flavor,
                                         ex_keyname=keypair_name,
                                         ex_userdata=userdata,
                                         ex_security_groups=[controller_group])

conn.wait_until_running([instance_controller_1])

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:
    pool = conn.ex_list_floating_ip_pools()[0]
    print('Allocating new Floating IP from pool: {}'.format(pool))
    unused_floating_ip = pool.create_floating_ip()

conn.ex_attach_floating_ip_to_node(instance_controller_1, unused_floating_ip)
```

```
print('Application will be deployed to http://%s' % unused_floating_ip.ip_address)

# Create a second instance that will be the worker instance.
instance_controller_1 = conn.ex_get_node_details(instance_controller_1.id)
if instance_controller_1.public_ips:
    ip_controller = instance_controller_1.private_ips[0]
else:
    ip_controller = instance_controller_1.public_ips[0]

# For the worker instance, have the install script:
# - install the Faafo (-i faafo) services
# - enable and start the worker service (-r worker)
# - pass the address of the API instance (-e) and message queue (-m) so the
# worker can pick up requests
# - (optional) use the -d option to specify a database connection URL
userdata = '''#!/usr/bin/env bash
curl -L -s https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh | bash -s -- \
-i faafo -r worker -e 'http://%(ip_controller)s' \
-m 'amqp://guest:guest@%(ip_controller)s:5672/'
''' % {'ip_controller': ip_controller}

instance_worker_1 = conn.create_node(name='app-worker-1',
                                     image=image,
                                     size=flavor,
                                     ex_keyname=keypair_name,
                                     ex_userdata=userdata,
                                     ex_security_groups=[worker_group])

conn.wait_until_running([instance_worker_1])
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:
    pool = conn.ex_list_floating_ip_pools()[0]
    print('Allocating new Floating IP from pool: {}'.format(pool))
    unused_floating_ip = pool.create_floating_ip()

conn.ex_attach_floating_ip_to_node(instance_worker_1, unused_floating_ip)
print('The worker will be available for SSH at %s' % unused_floating_ip.ip_address)
```

Executing the script, reveals the assigned floating IPs for the respective instances as can be seen below.

Output:

```
Checking for existing SSH key pair...
```

```
adding keypair...
<KeyPair name=jap_sse692_key
fingerprint=a5:8e:3d:52:f6:27:5c:e6:6e:14:b6:0e:0c:29:63:33 driver=OpenStack>
Checking for unused Floating IP...
Allocating new Floating IP from pool: <OpenStack_1_1_FloatingIpPool: name=public>
Application will be deployed to http://192.168.1.225
Checking for unused Floating IP...
Allocating new Floating IP from pool: <OpenStack_1_1_FloatingIpPool: name=public>
The worker will be available for SSH at 192.168.1.226
```

Looking at the Overview tab from the Horizon dashboard reveals the two instances along with remaining resources available from the host (see Figure 4). The Instances tab also shows the instances along with the floating IPs that have been assigned to them (see Figure 5).

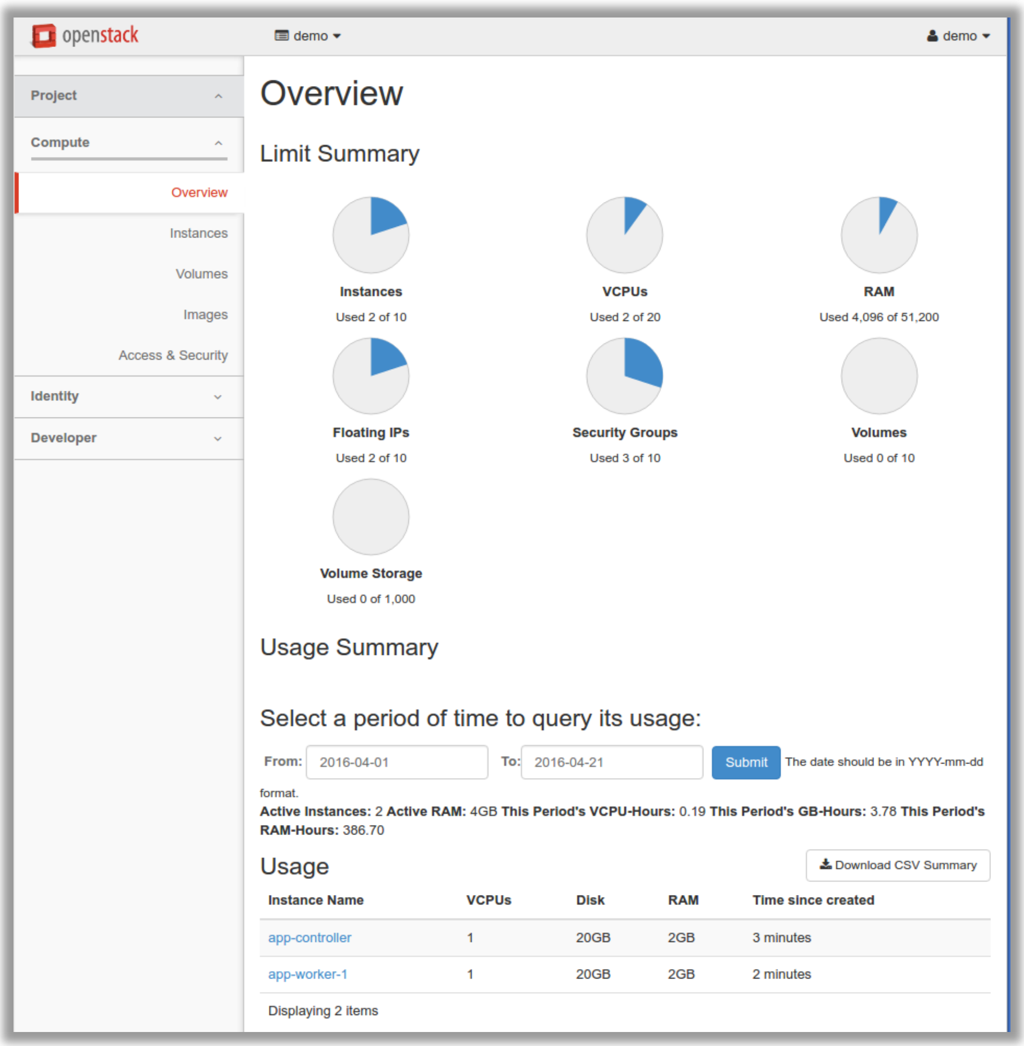


Figure 4: Overview of resources utilized while running Fractals app with separate instances

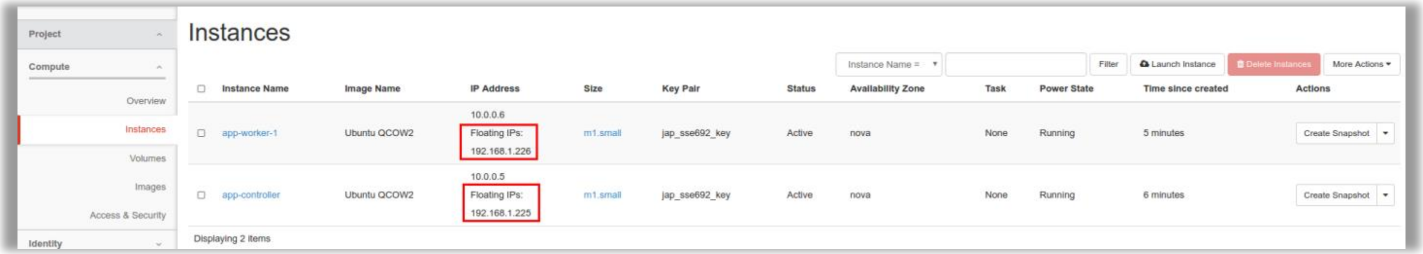


Figure 5: Different floating IPs assigned to the Fractal instances

At this point, the controller and worker instances can be accessed via SSH from a remote terminal (see Figure 6). Once logged in, the faafo services can be seen running in processes on each instance (see Figure 7 – Figure 10).

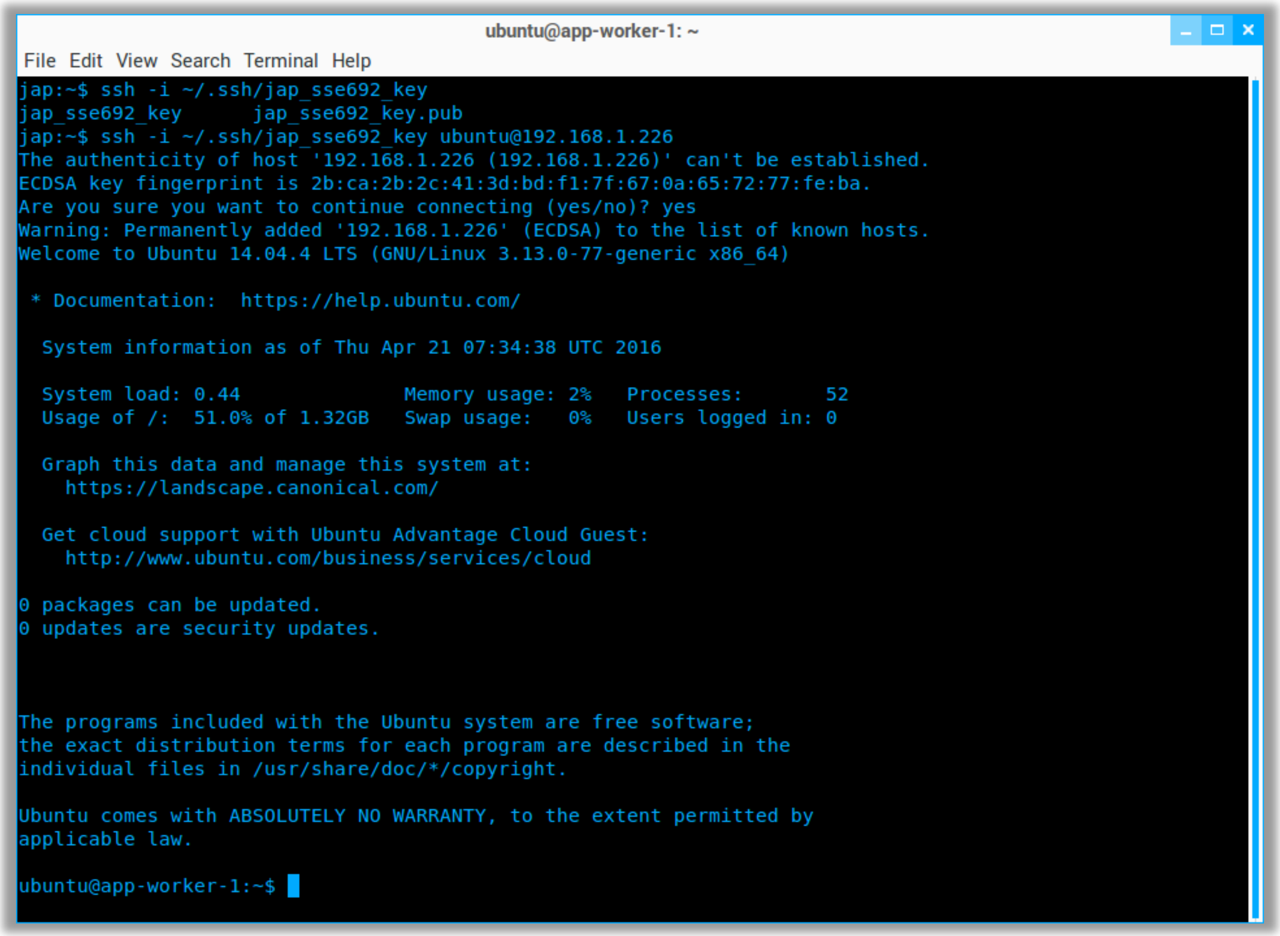


Figure 6: Logged into the Fractal's worker instance

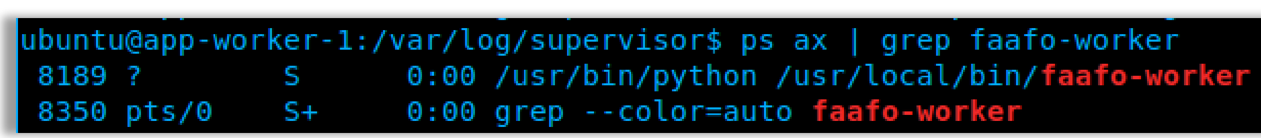


Figure 7: Worker instance showing the existence of the Fractal's worker service

```
top - 06:07:42 up 27 min,  1 user,  load average: 0.00, 0.01, 0.12
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2050044 total,   741776 used,  1308268 free,    30264 buffers
KiB Swap:        0 total,        0 used,        0 free.   593868 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 8189 root      20   0   82164  21032   4036 S  0.0  1.0   0:00.80 faafo-worker
```

Figure 8: Worker instance showing the Fractal's worker service running

```
ubuntu@app-controller: ~
File Edit View Search Terminal Help
jap:~$ ssh -i ~/.ssh/
jap_sse692_key      jap_sse692_key.pub  known_hosts
jap:~$ ssh -i ~/.ssh/jap_sse692_key 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 90:b6:75:6d:83:50:54:60:b4:76:8f:82:ea:34:84:43.
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 Thu Apr 21 07:32:50 UTC 2016

  System load: 0.47              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@app-controller:~$
```

Figure 9: Logged into the Fractal's controller instance

```
ubuntu@app-controller:/var/log/supervisor$ ps ax | grep faafo-api
 9144 ?        S      0:01 /usr/bin/python /usr/local/bin/faafo-api
 9583 pts/0    S+     0:00 grep --color=auto faafo-api
```

Figure 10: Controller instance showing the existence of the Fractal's API service

3.2 Demonstration

With the instances up and the faafo processes successfully running, the Fractal app can be used to create fractal images using the Command Line Interface, or CLI, from the controller instance (see Figure 11).

```
ubuntu@app-controller:~$ faafo --endpoint-url http://localhost --verbose create
Option "verbose" from group "DEFAULT" is deprecated for removal.  Its value may be silently ignored in th
e future.
2016-04-23 06:14:01.321 9624 INFO faafo.client [-] generating 9 task(s)
```

Figure 11: Creating fractals from the API controller instance

Once the request has been made, the request is posted in the message queue service (RabbitMQ) which is “heard” by the worker instance. At this point, the worker retrieves the request and starts producing the images. Depending on the specifics of the request, it could cause most or all of the worker’s resources to be utilized during processing (see Figure 12).

```
ubuntu@app-worker-1: /var/log/supervisor
File  Edit  View  Search  Terminal  Help
top - 06:14:43 up 34 min,  1 user,  load average: 0.49, 0.14, 0.12
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 98.0 us,  0.0 sy,  0.0 ni,  2.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2050044 total,   745744 used,  1304300 free,    30296 buffers
KiB Swap:        0 total,        0 used,        0 free.   593868 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 8189 root      20   0   85820  24680   4220 R 97.5  1.2   0:41.57 faafo-worker
```

Figure 12: Increased CPU load on the worker instance as it creates fractals

Once the fractals have been generated, their details can displayed from the controller instance’s faafo CLI (see Figure 13). This information is stored on the worker instance, so the message queue service is once again utilized behind the scenes to retrieve the info from the worker instance.

```
ubuntu@app-controller:~$ faafo list
2016-04-23 06:35:22.124 10019 INFO faafo.client [-] listing all fractals
+--------------------------------------+------------------+-------------+
|                 UUID                 |    Dimensions    |   Filesize  |
+--------------------------------------+------------------+-------------+
| 0922acaf-8c9a-4d60-ade8-d0750c9603a3 | 352 x 937 pixels | 43933 bytes |
| 2a1b7a2c-540f-497d-9509-756da17039e5 | 450 x 760 pixels | 30492 bytes |
| 74af2f7d-eb06-428d-ad94-80ea107ef57d | 945 x 760 pixels | 69707 bytes |
| 9d7c22be-30a6-4242-8af9-53bd56e1f610 | 420 x 580 pixels | 28300 bytes |
| 9f75ab56-dbd1-4ed2-8a01-f1e9f87dbd0d | 383 x 631 pixels | 38187 bytes |
| e8db5225-8e37-4dd4-b1d4-0e7043689456 | 717 x 577 pixels | 58147 bytes |
| e8fe9cfe-edcc-46a6-8f06-a8158a39e30a | 858 x 351 pixels | 17762 bytes |
| f1dca409-cf2b-471f-91d9-369f054b646c | 973 x 967 pixels | 52754 bytes |
| f7582795-8373-464e-b8ea-d6e44cd69244 | 721 x 794 pixels | 91110 bytes |
+--------------------------------------+------------------+-------------+
```

Figure 13: API controller showing details of the generated fractals

Pulling up the API controller's floating IP in a web browser displays the generated fractals. The interesting thing to note is how the information reveals that the images were not created by the controller instance, but by the worker instance (see Figure 14).

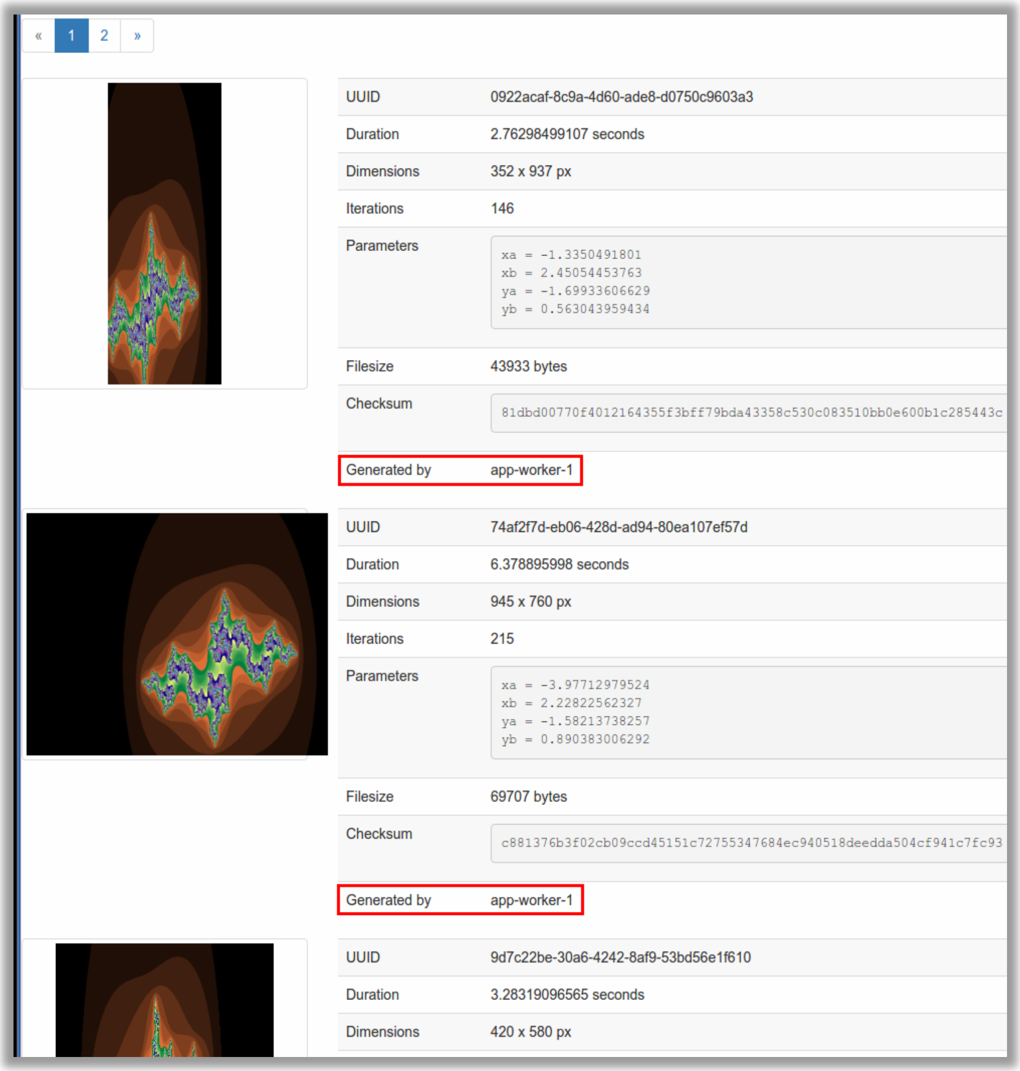


Figure 14: Fractals App running with separate instances

4. Load Balancing

The previous section illustrated how the all-in-one instance deployment of the Fractals app could be split into multiple instances based on a “separation of concerns” principle where a specific instance is only responsible for a given task. This is a good step in the right direction for the design of the application, but it is still limited by single instances with limited resources. Problems are encountered if/when the creation of high resolution images is requested from the single worker (see Figure 15 and Figure 16) or when simultaneous fractal creation requests are initiated on the API controller (see Figure 17).

```
ubuntu@app-controller:~$ uptime
 02:42:54 up 7 min,  1 user,  load average: 0.20, 0.54, 0.33
ubuntu@app-controller:~$ faafo create --height 9999 --width 9999 --tasks 5
2016-04-24 02:44:11.794 9355 INFO faafo.client [-] generating 5 task(s)
ubuntu@app-controller:~$ uptime
 02:50:32 up 14 min,  1 user,  load average: 0.00, 0.12, 0.20
```

Figure 15: Fractal’s API controller load averages after max size fractal create command

```
ubuntu@app-worker-1: ~
File  Edit  View  Search  Terminal  Help
top - 03:24:30 up 48 min,  1 user,  load average: 1.00, 0.99, 0.95
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2050044 total,  1133844 used,   916200 free,    30096 buffers
KiB Swap:        0 total,        0 used,        0 free.   593316 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 8272 root      20   0  473976 412840   4220 R 99.9 20.1  40:14.03 faafo-worker
```

Figure 16: Fractal’s worker instance at max load after max size fractal create command

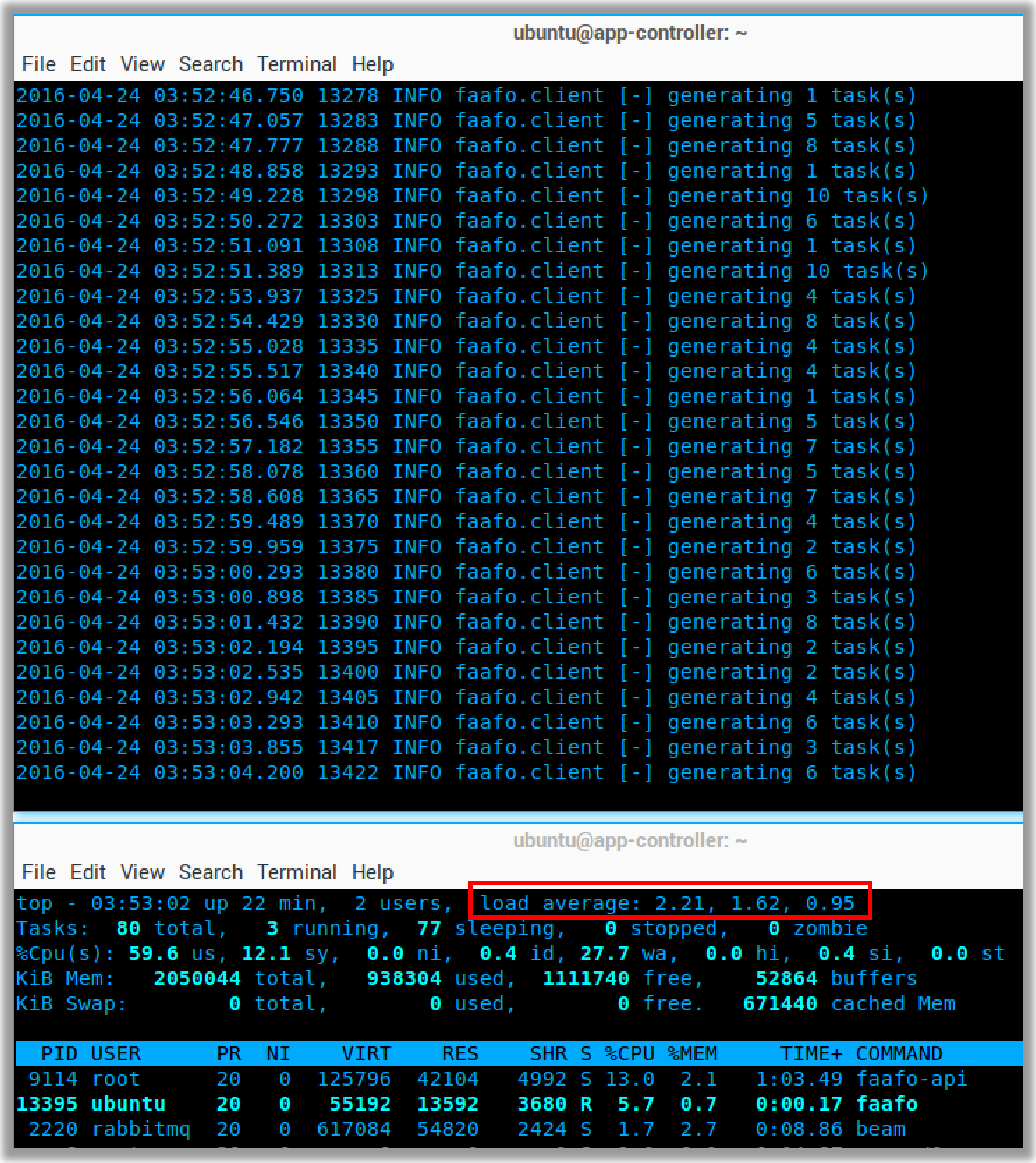


Figure 17: Fractal's API controller at max load during multiple creation commands

4.1 Deployment

When dealing with loading and scaling issues, a useful cloud design approach is to deploy multiple service instances for the purposes of load balancing. In the previous section, the all-in-one instance was decomposed into separate instances that served unique purposes within the scope of the system. This modularity can be advanced such that multiple controller and worker instances can be created and utilized to handle increasing task loads. The deployment script below does exactly that. In concept it is essentially the same as the previous deployment script with the main differences being the number of API controller and worker instances that are created and the introduction of a service instance that will be used to store the database and messaging service.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 23 22:56:51 2016
@author: jap
"""

# APIs for interacting 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'
keypair_name = 'jap_sse692_key'
image_name = 'Ubuntu QCOW2'
flavor_id = 'd2'

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)

# Delete any previously created instances and security groups.
for instance in conn.list_nodes():
    if instance.name in ['all-in-one', 'app-worker-1', 'app-worker-2', 'app-controller', \
    'app-services', 'app-api-1', 'app-api-2', 'app-worker-1', 'app-worker-2', 'app-worker-3',]:
        print('Destroying Instance %s' % instance.name)
        conn.destroy_node(instance)

for group in conn.ex_list_security_groups():
    if group.name in ['control', 'worker', 'api', 'services']:
        print('Deleting security group: %s' % group.name)
        conn.ex_delete_security_group(group)

# Setup image and flavor
images = conn.list_images()
# Get the image from its name rather than its complex id.
image = [i for i in images if image_name in i.name][0]
flavor = conn.ex_get_size(flavor_id)

# Setup access key
```

```
print('Checking for existing SSH key pair...')
pub_key_file = '~/.ssh/{}.pub'.format(keypair_name)
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)

api_group = conn.ex_create_security_group('api', 'for API services only')
conn.ex_create_security_group_rule(api_group, 'TCP', 80, 80)
conn.ex_create_security_group_rule(api_group, 'TCP', 22, 22)

worker_group = conn.ex_create_security_group('worker', 'for services that run on a worker node \
                                                        (instance)')
conn.ex_create_security_group_rule(worker_group, 'TCP', 22, 22)
controller_group = conn.ex_create_security_group('control', 'for services that run on a control \
                                                            node')
conn.ex_create_security_group_rule(controller_group, 'TCP', 22, 22)
conn.ex_create_security_group_rule(controller_group, 'TCP', 80, 80)
conn.ex_create_security_group_rule(controller_group, 'TCP', 5672, 5672, \
                                   source_security_group=worker_group)

services_group = conn.ex_create_security_group('services', 'for DB and AMQP services only')
conn.ex_create_security_group_rule(services_group, 'TCP', 22, 22)
conn.ex_create_security_group_rule(services_group, 'TCP', 3306, 3306, \
                                   source_security_group=api_group)
conn.ex_create_security_group_rule(services_group, 'TCP', 5672, 5672, \
                                   source_security_group=worker_group)
conn.ex_create_security_group_rule(services_group, 'TCP', 5672, 5672, \
                                   source_security_group=api_group)

# Floating IP Helper Function
def get_floating_ip(conn):
    '''A helper function to re-use available Floating IPs'''
    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:
        pool = conn.ex_list_floating_ip_pools()[0]
        unused_floating_ip = pool.create_floating_ip()
    return unused_floating_ip
```

```
# Add a central database and a messaging instance. These will be used to track the state of \
# the fractals and to coordinate the communication between the services:
# - install the RabbitMQ messaging service (-i messaging)
# - install the central database service (-i database)
userdata = '''#!/usr/bin/env bash
curl -L -s https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh | bash -s -- \
-i database -i messaging
'''

instance_services = conn.create_node(name='app-services',
                                     image=image,
                                     size=flavor,
                                     ex_keyname=keypair_name,
                                     ex_userdata=userdata,
                                     ex_security_groups=[services_group])
instance_services = conn.wait_until_running([instance_services])[0][0]
services_ip = instance_services.private_ips[0]

# Create multiple API instances to handle more fractal requests.
userdata = '''#!/usr/bin/env bash
curl -L -s https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh | bash -s -- \
-i faafo -r api -m 'amqp://guest:guest@%(services_ip)s:5672/' \
-d 'mysql://faafo:password@%(services_ip)s:3306/faafo'
''' % {'services_ip': services_ip}
instance_api_1 = conn.create_node(name='app-api-1',
                                  image=image,
                                  size=flavor,
                                  ex_keyname=keypair_name,
                                  ex_userdata=userdata,
                                  ex_security_groups=[api_group])
instance_api_2 = conn.create_node(name='app-api-2',
                                  image=image,
                                  size=flavor,
                                  ex_keyname=keypair_name,
                                  ex_userdata=userdata,
                                  ex_security_groups=[api_group])
instance_api_1 = conn.wait_until_running([instance_api_1])[0][0]
api_1_ip = instance_api_1.private_ips[0]
instance_api_2 = conn.wait_until_running([instance_api_2])[0][0]
api_2_ip = instance_api_2.private_ips[0]

for instance in [instance_api_1, instance_api_2]:
    floating_ip = get_floating_ip(conn)
    conn.ex_attach_floating_ip_to_node(instance, floating_ip)
    print('allocated %(ip)s to %(host)s' % {'ip': floating_ip.ip_address, 'host': instance.name})

# Create extra workers to create more fractals.
userdata = '''#!/usr/bin/env bash
curl -L -s https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh | bash -s -- \
```

```
-i faafo -r worker -e 'http://%(api_1_ip)s' -m 'amqp://guest:guest@%(services_ip)s:5672/'
''' % {'api_1_ip': api_1_ip, 'services_ip': services_ip}
instance_worker_1 = conn.create_node(name='app-worker-1',
                                     image=image,
                                     size=flavor,
                                     ex_keyname=keypair_name,
                                     ex_userdata=userdata,
                                     ex_security_groups=[worker_group])
instance_worker_2 = conn.create_node(name='app-worker-2',
                                     image=image,
                                     size=flavor,
                                     ex_keyname=keypair_name,
                                     ex_userdata=userdata,
                                     ex_security_groups=[worker_group])
instance_worker_3 = conn.create_node(name='app-worker-3',
                                     image=image,
                                     size=flavor,
                                     ex_keyname=keypair_name,
                                     ex_userdata=userdata,
                                     ex_security_groups=[worker_group])
```

As can be seen in the script, two API controller instances are created along with three worker instances and a services instance. These extra instances will help offset any load/task imbalances that could occur from multiple fractal requests or process-intensive calculations. Only the API controller instances can be accessed via a web browser since they are the only instances with assigned floating IPs, but this is by design because public access to the worker instances is not desirable in this scenario (see Figure 18).

It is also worth noting that a point of potential trouble is with the deployment of the controller instances. For this deployment, a MySQL database is deployed with the service instance, and the database type specified in the libcloud userdata field (highlighted blue above) of the controller instances must match what is installed on the service instance. If they do not match, then the controllers will be unable to communicate with the database on the service instance and the image data cannot be stored properly and no images will be displayed from the browser. Everything may appear fine, but if the file sizes are showing 0x0, then that is an indicator that communication with the database is not setup properly.

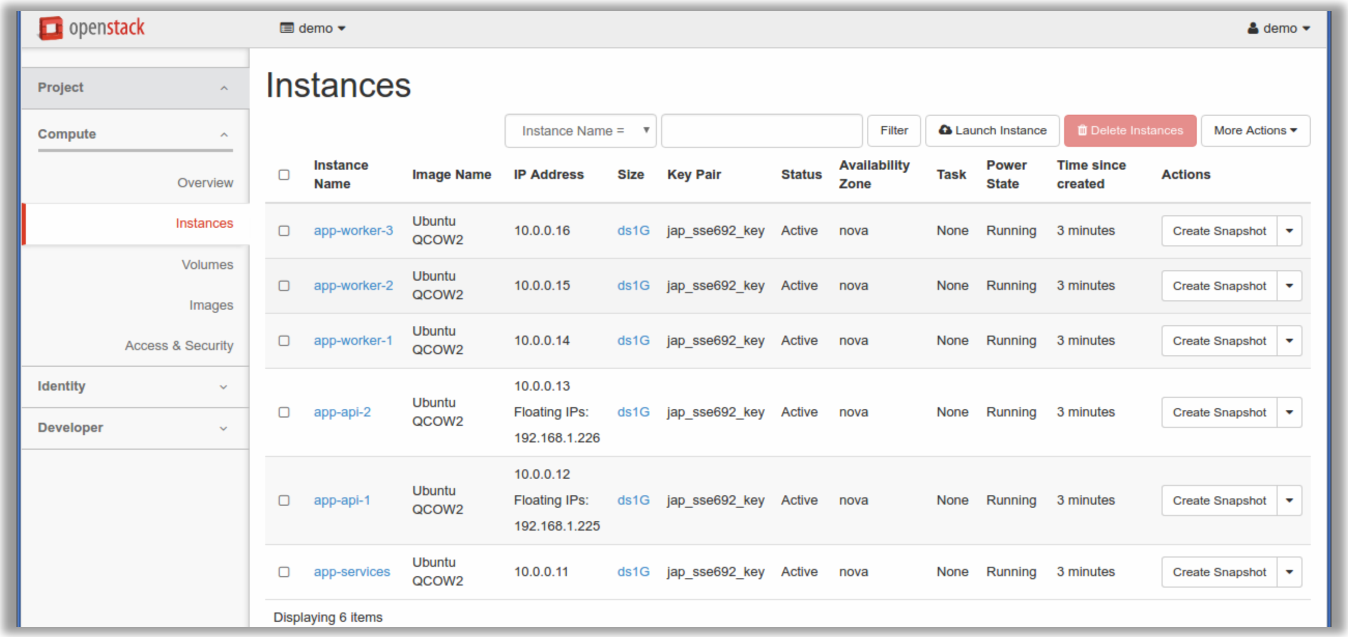


Figure 18: Modularized instances of the Fractal app

4.2 Demonstration

Now that the application has been successfully deployed to the cloud, the Fractals app can be accessed from the CLI of either one of the API controller instances. The `faafo create` command is used to create fractal images (see Figure 19). Here, the `create` command initiates the creation of 7 fractal image generation tasks. Similar to the previous deployment, the requests are placed in a message queue that is being listened to by all of the worker instances. If a worker instance is not busy or in a failed state, the worker will take the request and start executing its tasks. The main advantage over the previous architecture is that instead of a single worker being responsible for all requests creating a potential bottleneck point, other similar worker instances exist so that multiple task requests can be executed simultaneously. This also has the nice side effect of providing fail-safes in the architecture in that if a worker instance goes down, there are other worker instances in the system that can still handle task requests.

Figure 19: Fractal's API-1 controller requesting fractals

The `faafo list` command lists the fractal images that have been generated to this point (see Figure 20). It is worth noting that the same information is available from either controller instance since the controller is only responsible for serving as the front-end of the application. The actual data is stored on the service instance which can be accessed from any number of API controllers.

```
ubuntu@app-api-1:~$ faafo list
2016-04-25 06:58:00.961 8372 INFO faafo.client [-] listing all fractals
+--------------------------------------+------------------+-------------+
|                 UUID                 |    Dimensions    |   Filesize  |
+--------------------------------------+------------------+-------------+
| 0f599654-ea1c-477a-a6da-4042f51e1b80 | 748 x 844 pixels | 21625 bytes  |
| 4b0bf2e2-bde1-4798-a487-c8143b59b1f9 | 783 x 320 pixels | 18248 bytes  |
| 5c39acc5-4e6f-48d3-9873-529210f11b2a | 707 x 443 pixels | 106975 bytes |
| 6dfca0de-c043-4737-a22e-7ca2ecbb7a0e | 847 x 747 pixels | 60552 bytes  |
| 6ee050d5-f085-4404-b71d-6661c212caf8 | 786 x 923 pixels | 62611 bytes  |
| 88b7544e-5fc2-49e9-86ed-e79768549725 | 678 x 731 pixels | 38392 bytes  |
| f2b7ae0e-de90-4872-b63f-5d8ae6044b20 | 498 x 262 pixels | 22268 bytes  |
+--------------------------------------+------------------+-------------+
```

Figure 20: API-1 controller listing the generated fractals

The `faafo show` command displays the general properties of a specific fractal image. In Figure 21 below, the properties of three unique fractal images are shown. The interesting thing to note here is how each fractal image was generated with a different worker instance which confirms that the tasks are being split amongst the different workers.

```
ubuntu@app-api-1:~$ faafo show 0f599654-ea1c-477a-a6da-4042f51e1b80
2016-04-25 07:05:27.268 8417 INFO faafo.client [-] showing fractal 0f599654-ea1c-477a-a6da-4042f51e1b80
+--------------+------------------------------------------------------------------+
| Parameter    | Value                                                            |
+--------------+------------------------------------------------------------------+
| uuid         | 0f599654-ea1c-477a-a6da-4042f51e1b80                             |
| duration     | 5.371580 seconds                                                 |
| dimensions   | 748 x 844 pixels                                                 |
| iterations   | 253                                                              |
| xa           | -3.06992                                                         |
| xb           | 3.87433                                                          |
| ya           | -0.583997                                                        |
| yb           | 0.643574                                                         |
| size         | 21625 bytes                                                      |
| checksum     | 2c28aceb1dee4eb9e76464bbf4cfaccba7f4fc44a9d37603335430bf859a0d88 |
| generated_by | app-worker-3                                                     |
+--------------+------------------------------------------------------------------+
ubuntu@app-api-1:~$ faafo show 4b0bf2e2-bde1-4798-a487-c8143b59b1f9
2016-04-25 07:05:38.426 8422 INFO faafo.client [-] showing fractal 4b0bf2e2-bde1-4798-a487-c8143b59b1f9
+--------------+------------------------------------------------------------------+
| Parameter    | Value                                                            |
+--------------+------------------------------------------------------------------+
| uuid         | 4b0bf2e2-bde1-4798-a487-c8143b59b1f9                             |
| duration     | 3.468610 seconds                                                 |
| dimensions   | 783 x 320 pixels                                                 |
| iterations   | 444                                                              |
| xa           | -3.72236                                                         |
| xb           | 1.20942                                                          |
| ya           | -0.790315                                                        |
| yb           | 2.72242                                                          |
| size         | 18248 bytes                                                      |
| checksum     | 79f56d398acd663bbea9184059a3f7f9b169a16a4a61a6aa6cd2293775a4a442 |
| generated_by | app-worker-1                                                     |
+--------------+------------------------------------------------------------------+
ubuntu@app-api-1:~$ faafo show 6ee050d5-f085-4404-b71d-6661c212caf8
2016-04-25 07:05:50.873 8427 INFO faafo.client [-] showing fractal 6ee050d5-f085-4404-b71d-6661c212caf8
+--------------+------------------------------------------------------------------+
| Parameter    | Value                                                            |
+--------------+------------------------------------------------------------------+
| uuid         | 6ee050d5-f085-4404-b71d-6661c212caf8                             |
| duration     | 6.950060 seconds                                                 |
| dimensions   | 786 x 923 pixels                                                 |
| iterations   | 325                                                              |
| xa           | -2.50087                                                         |
| xb           | 2.08538                                                          |
| ya           | -2.99304                                                         |
| yb           | 2.26681                                                          |
| size         | 62611 bytes                                                      |
| checksum     | 6e8d65236cf7d7e8e28de38afd578b2dd59d52e5f873abbc05e46777bc88af4b |
| generated_by | app-worker-2                                                     |
+--------------+------------------------------------------------------------------+
```

Figure 21: API-1 controller listing details of certain fractals

With the architecture validated and confirmed, the final step is to view the images from the web browser. As noted earlier, there are two API controller instances that have been assigned floating IPs. These IPs are publicly accessible and provide a convenient mechanism for displaying the fractal images from any web browser (see Figure 22). It is also worth noting that with two controller instances – like the worker instances – if one of the controller instances goes down, the other controller instance can still be utilized as the front-end of the Fractal app.

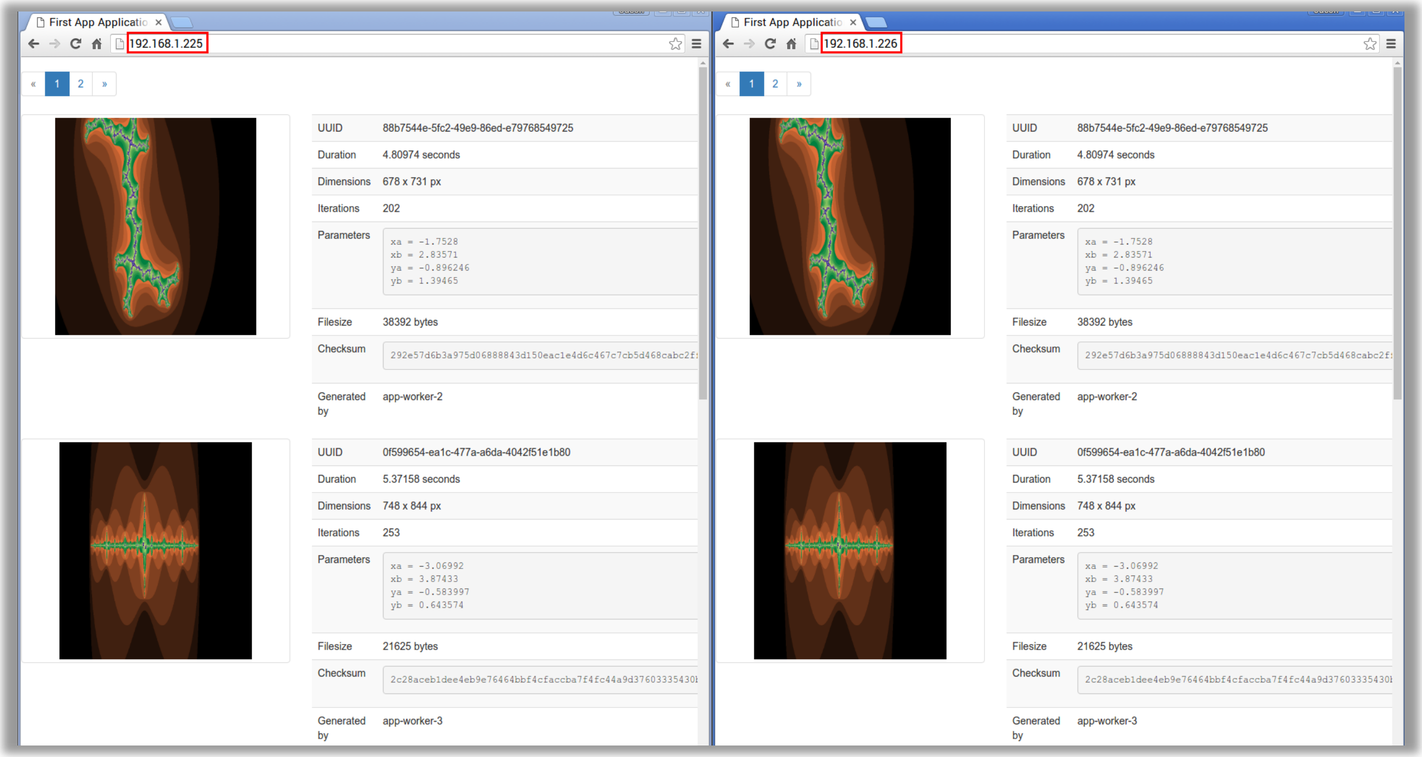


Figure 22: Fractal app running with redundant API controllers

5. Orchestrating with Heat

While the previous architecture takes great strides of achieving several of the architecture goals discussed earlier, there are still limitations with the system. Namely, manual processes for scaling the application up or down. If any of the instances goes down, a new instance must be manually spun up. This requires constant monitoring of the system and becomes a very fragile characteristic of the architecture. To combat this, OpenStack provides the Heat service to serve as the "orchestrator" of the system. This service allows the system to add automated monitoring and scaling and removes the rigidness of the architecture. For the context of this section Orchestration and the Heat service will be referenced interchangeably.

5.1 Setup & Configuration

By default, Orchestration is not installed or configured by the DevStack script, so this must be installed and configured before the DevStack stack.sh script is executed. The snippet below provides the extra settings that are needed for the Heat service to be setup properly from the DevStack script.

Extra settings in local.conf to configure Heat services (plus extra images and plugins)

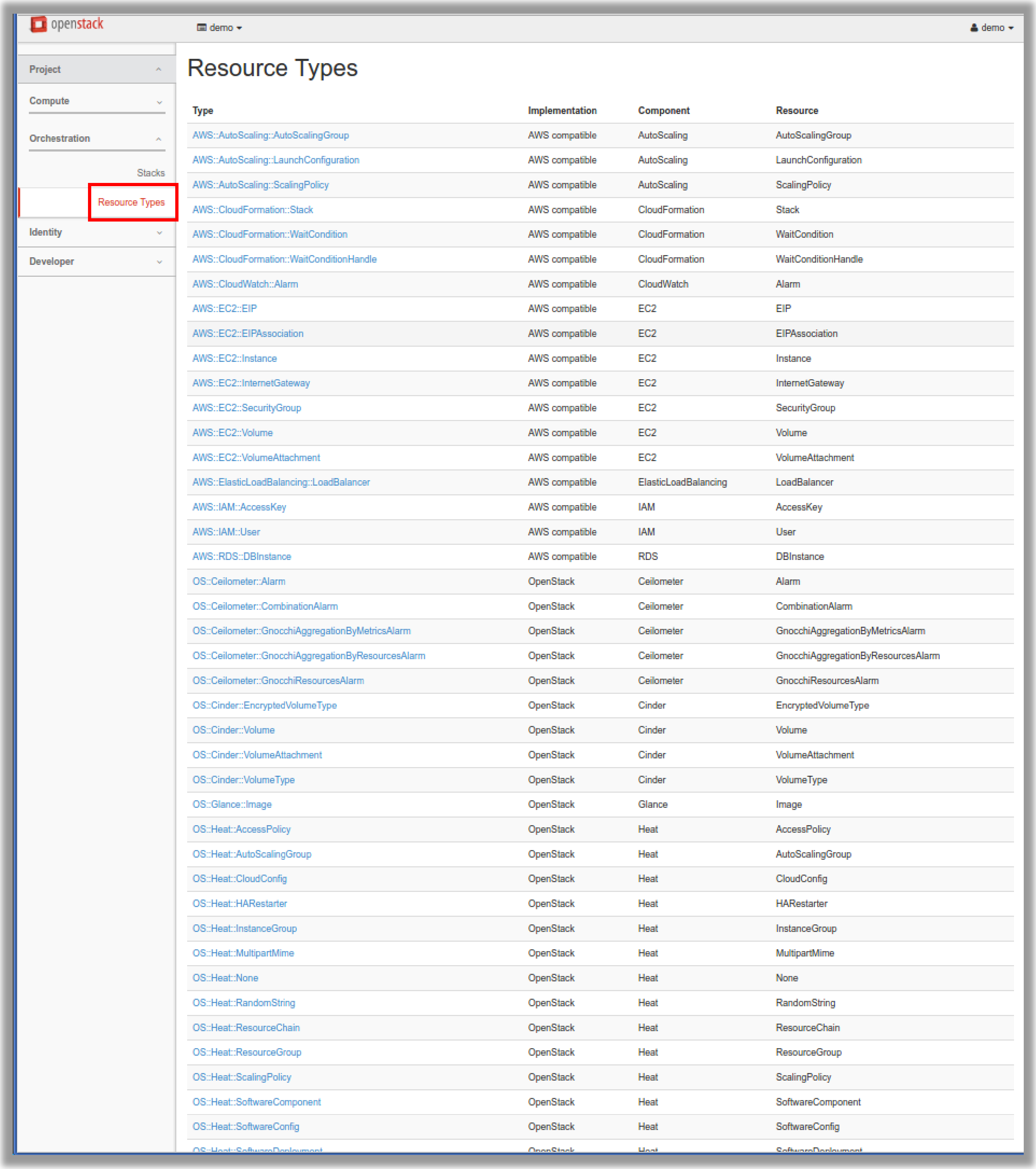
The highlighted portion is for enabling Orchestration. The other lines are useful for adding new images and/or extra features that can be used with Heat (like metering and alarms).

```
# Automatically download and register a Ubuntu QCOW2 image.
IMAGE_URL_SITE="http://cloud-images.ubuntu.com/"
IMAGE_URL_PATH="releases/14.04.4/14.04.4/"
IMAGE_URL_FILE="ubuntu-14.04-server-cloudimg-amd64-disk1.img"
IMAGE_URLS+=","$IMAGE_URL_SITE$IMAGE_URL_PATH$IMAGE_URL_FILE

# Enable Heat (Orchestration) Services.
enable_service h-eng h-api h-api-cfn h-api-cw

# Automatically download and register a VM image that heat can launch.
IMAGE_URL_SITE="http://download.fedoraproject.org/"
IMAGE_URL_PATH="pub/fedora/linux/releases/23/Cloud/x86_64/Images/"
IMAGE_URL_FILE="Fedora-Cloud-Base-23-20151030.x86_64.qcow2"
IMAGE_URLS+=","$IMAGE_URL_SITE$IMAGE_URL_PATH$IMAGE_URL_FILE

# Enable ceilometer and aodh for metering and alarms.
CEILOMETER_BACKEND=mongodb
enable_plugin ceilometer https://git.openstack.org/openstack/ceilometer
enable_plugin aodh https://git.openstack.org/openstack/aodh
```

demo-openrc.sh (OpenStack CLI environment file)

The libcloud package does not currently provide support for OpenStack Orchestration so all interaction will be achieved via the OpenStack CLI on the cloud host. However, before the host can understand OpenStack CLI commands, certain OpenStack environment variables need to be set within the host. To set the required environment variables for the OpenStack command-line clients, an environment shell file must be created. This file is typically named openrc.sh and is often automatically provided by OpenStack. The file can be downloaded from the OpenStack dashboard as an administrative user or any other user. This project-specific environment file contains the credentials that all OpenStack services use. When the file is sourced, the OpenStack CLI environment variables are set for the **current shell only**! These variables enable the OpenStack CLI commands to communicate with the OpenStack services that run in the cloud.

```
#!/usr/bin/env bash

# To use an OpenStack cloud you need to authenticate against the Identity
# service named keystone, which returns a **Token** and **Service Catalog**.
# The catalog contains the endpoints for all services the user/tenant has
# access to - such as Compute, Image Service, Identity, Object Storage, Block
# Storage, and Networking (code-named nova, glance, keystone, swift,
# cinder, and neutron).
#
# *NOTE*: Using the 2.0 *Identity API* does not necessarily mean any other
# OpenStack API is version 2.0. For example, your cloud provider may implement
# Image API v1.1, Block Storage API v2, and Compute API v2.0. OS_AUTH_URL is
# only for the Identity API served through keystone.
export OS_AUTH_URL=http://192.168.1.100:5000

# With the addition of Keystone we have standardized on the term **tenant**
# as the entity that owns the resources.
export OS_TENANT_ID=9d5f180a123a4c089afb7e5ca159f011
export OS_TENANT_NAME="demo"

# unsetting v3 items in case set
unset OS_PROJECT_ID
unset OS_PROJECT_NAME
unset OS_USER_DOMAIN_NAME

# In addition to the owning entity (tenant), OpenStack stores the entity
# performing the action as the **user**.
export OS_USERNAME="demo"

# With Keystone you pass the keystone password.
echo "Please enter your OpenStack Password: "
read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT

# If your configuration has multiple regions, we set that information here.
# OS_REGION_NAME is optional and only valid in certain environments.
export OS_REGION_NAME="RegionOne"
# Don't leave a blank variable, unset it if it was empty
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
```



Resource Types

Type	Implementation	Component	Resource
AWS::AutoScaling::AutoScalingGroup	AWS compatible	AutoScaling	AutoScalingGroup
AWS::AutoScaling::LaunchConfiguration	AWS compatible	AutoScaling	LaunchConfiguration
AWS::AutoScaling::ScalingPolicy	AWS compatible	AutoScaling	ScalingPolicy
AWS::CloudFormation::Stack	AWS compatible	CloudFormation	Stack
AWS::CloudFormation::WaitCondition	AWS compatible	CloudFormation	WaitCondition
AWS::CloudFormation::WaitConditionHandle	AWS compatible	CloudFormation	WaitConditionHandle
AWS::CloudWatch::Alarm	AWS compatible	CloudWatch	Alarm
AWS::EC2::EIP	AWS compatible	EC2	EIP
AWS::EC2::EIPAssociation	AWS compatible	EC2	EIPAssociation
AWS::EC2::Instance	AWS compatible	EC2	Instance
AWS::EC2::InternetGateway	AWS compatible	EC2	InternetGateway
AWS::EC2::SecurityGroup	AWS compatible	EC2	SecurityGroup
AWS::EC2::Volume	AWS compatible	EC2	Volume
AWS::EC2::VolumeAttachment	AWS compatible	EC2	VolumeAttachment
AWS::ElasticLoadBalancing::LoadBalancer	AWS compatible	ElasticLoadBalancing	LoadBalancer
AWS::IAM::AccessKey	AWS compatible	IAM	AccessKey
AWS::IAM::User	AWS compatible	IAM	User
AWS::RDS::DBInstance	AWS compatible	RDS	DBInstance
OS::Ceilometer::Alarm	OpenStack	Ceilometer	Alarm
OS::Ceilometer::CombinationAlarm	OpenStack	Ceilometer	CombinationAlarm
OS::Ceilometer::GnocchiAggregationByMetricsAlarm	OpenStack	Ceilometer	GnocchiAggregationByMetricsAlarm
OS::Ceilometer::GnocchiAggregationByResourcesAlarm	OpenStack	Ceilometer	GnocchiAggregationByResourcesAlarm
OS::Ceilometer::GnocchiResourcesAlarm	OpenStack	Ceilometer	GnocchiResourcesAlarm
OS::Cinder::EncryptedVolumeType	OpenStack	Cinder	EncryptedVolumeType
OS::Cinder::Volume	OpenStack	Cinder	Volume
OS::Cinder::VolumeAttachment	OpenStack	Cinder	VolumeAttachment
OS::Cinder::VolumeType	OpenStack	Cinder	VolumeType
OS::Glance::Image	OpenStack	Glance	Image
OS::Heat::AccessPolicy	OpenStack	Heat	AccessPolicy
OS::Heat::AutoScalingGroup	OpenStack	Heat	AutoScalingGroup
OS::Heat::CloudConfig	OpenStack	Heat	CloudConfig
OS::Heat::HARestarter	OpenStack	Heat	HARestarter
OS::Heat::InstanceGroup	OpenStack	Heat	InstanceGroup
OS::Heat::MultipartMime	OpenStack	Heat	MultipartMime
OS::Heat::None	OpenStack	Heat	None
OS::Heat::RandomString	OpenStack	Heat	RandomString
OS::Heat::ResourceChain	OpenStack	Heat	ResourceChain
OS::Heat::ResourceGroup	OpenStack	Heat	ResourceGroup
OS::Heat::ScalingPolicy	OpenStack	Heat	ScalingPolicy
OS::Heat::SoftwareComponent	OpenStack	Heat	SoftwareComponent
OS::Heat::SoftwareConfig	OpenStack	Heat	SoftwareConfig
OS::Heat::SoftwareDeployment	OpenStack	Heat	SoftwareDeployment

Figure 23: Heat (Orchestration) resource types

5.2 Deployment

With Orchestration deployed as part of OpenStack and the environment variables set, the services provided by Orchestration can be utilized. The Orchestration service provides a template-based way to describe a cloud application, then coordinates running the needed OpenStack API calls to run cloud applications. The templates enable the creation of most OpenStack resource types (see Figure 23), such as instances, networking information, volumes, security groups, and even users. It also provides more advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. To achieve this, Orchestration works directly with the HOT (Heat Orchestration Template) templating language. The template file below is used to deploy the Fractals application using Orchestration and Heat.

hello_faafo.yaml

This HOT template file will deploy the Fractals application in the same way that the previous Python scripts did, except that the deployment will be as a "stack" of resources instead of single, individualized resources like worker and controller instances. This template file was provided by the Fractal app development team, however, it was discovered that this specific template file could only be used with OpenStack deployments utilizing the Neutron networking service. Using the resource types list from the dashboard (see Figure 23), an equivalent type that was supported by this particular cloud system was successfully substituted for the `security_group` resource (highlighted below).

```
heat_template_version: 2014-10-16

description: |
  A template to bring up the faafo application as an all in one install

parameters:

  key_name:
    type: string
    description: Name of an existing KeyPair to enable SSH access to the instances
    default: id_rsa
    constraints:
      - custom_constraint: nova.keypair
        description: Must already exist on your cloud

  flavor:
    type: string
    description: The flavor the application is to use
    constraints:
      - custom_constraint: nova.flavor
        description:  Must be a valid flavor provided by your cloud provider.

  image_id:
```

```
    type: string
    description: ID of the image to use to create the instance
    constraints:
      - custom_constraint: glance.image
        description: Must be a valid image on your cloud

  faafo_source:
    type: string
    description: The http location of the faafo application install script
    default: https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh

resources:

#  security_group:
#    type: OS::Neutron::SecurityGroup
#    properties:
#      description: "SSH and HTTP for the all in one server"
#      rules: [
#        {remote_ip_prefix: 0.0.0.0/0,
#        protocol: tcp,
#        port_range_min: 22,
#        port_range_max: 22},
#        {remote_ip_prefix: 0.0.0.0/0,
#        protocol: tcp,
#        port_range_min: 80,
#        port_range_max: 80},]

# Use the following definition of the security_group resource if the Neutron networking service
# is not installed.
  security_group:
    type: AWS::EC2::SecurityGroup
    properties:
      GroupDescription: "SSH and HTTP for the all in one server"
      SecurityGroupIngress: [
        {CidrIp: 0.0.0.0/0,
        IpProtocol: tcp,
        FromPort: 22,
        ToPort: 22},
        {CidrIp: 0.0.0.0/0,
        IpProtocol: tcp,
        FromPort: 80,
        ToPort: 80},]

  server:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_id }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }
```

```
      security_groups:
        - {get_resource: security_group}
      user_data_format: RAW
      user_data:
        str_replace:
          template: |
            #!/usr/bin/env bash
            curl -L -s faafo_installer | bash -s -- \
            -i faafo -i messaging -r api -r worker -r demo
            wc_notify --data-binary '{"status": "SUCCESS"}'
          params:
            wc_notify: { get_attr: ['wait_handle', 'curl_cli'] }
            faafo_installer: { get_param: faafo_source }

  wait_handle:
    type: OS::Heat::WaitConditionHandle

  wait_condition:
    type: OS::Heat::WaitCondition
    depends_on: server
    properties:
      handle: { get_resource: wait_handle }
      count: 1
      # we'll give it 10 minutes
      timeout: 600

outputs:

  faafo_ip:
    description: The faafo url
    value:
      list_join: ['', ['Faafo can be found at: http://', get_attr: [server, first_address]]]
```

At this point, the stack can be create from the cloud hosts command-line using the OpenStack CLI (see Figure 24). When the stack has been successfully created (see Figure 25 – Figure 32), an architecture is in place to support features such as auto-scaling and nested stacks.

```
jap:~$ openstack stack create -t hello_faafo.yaml --parameter flavor=d2 --parameter key_name=jap_sse692_key --parameter image_id=a90eccaf-d019-4478-8c5f-5efa6248c488 hello_faafo
+---------------------+-----------------------------------------------------------------+
| Field               | Value                                                           |
+---------------------+-----------------------------------------------------------------+
| id                  | 17ac22e9-914d-4cf8-b4af-84cf334c0a16                            |
| stack_name          | hello_faafo                                                     |
| description         | A template to bring up the faafo application as an all in one install |
|                     |                                                                 |
| creation_time       | 2016-04-28T07:30:32                                             |
| updated_time        | None                                                            |
| stack_status        | CREATE_IN_PROGRESS                                              |
| stack_status_reason | Stack CREATE started                                            |
+---------------------+-----------------------------------------------------------------+
```

Figure 24: Orchestration CLI creating the stack

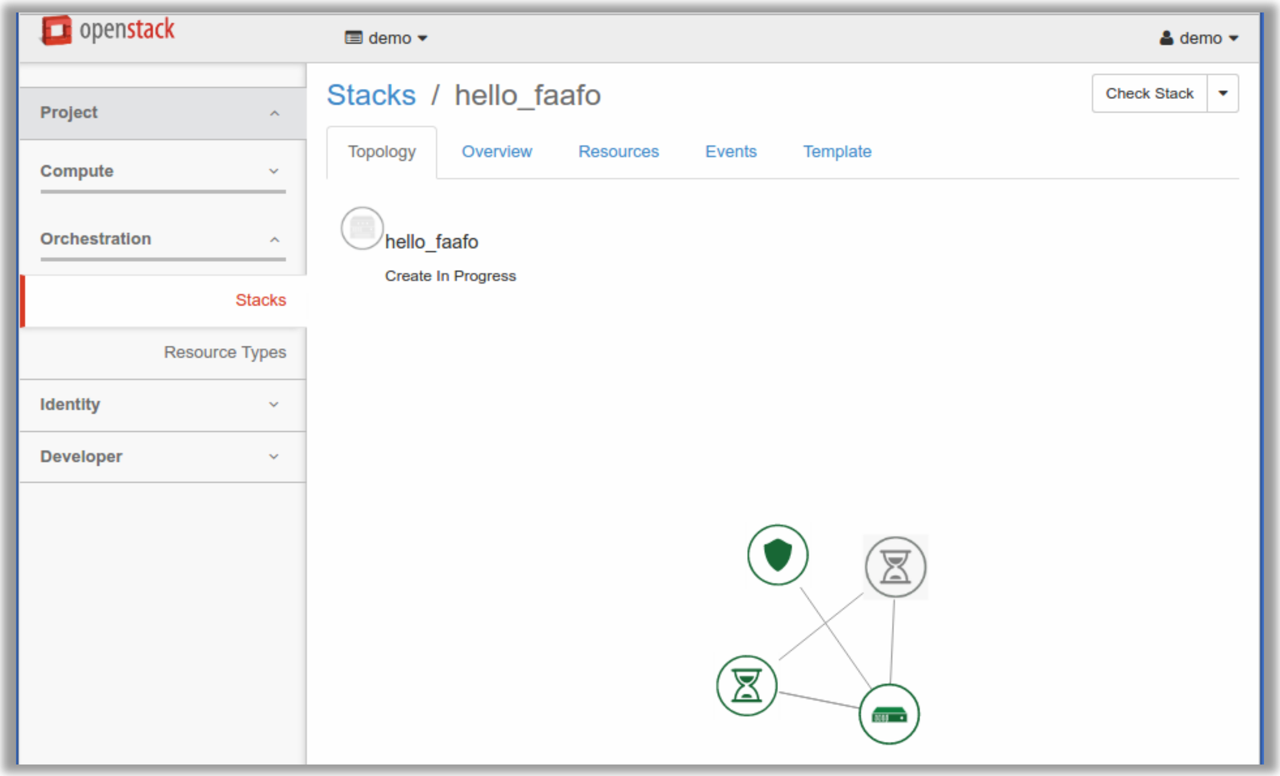


Figure 25: Orchestration stack creation in progress

```
jap:~$ openstack stack list
+--------------------------------------+-------------+-----------------+---------------------+--------------+
| ID                                   | Stack Name  | Stack Status    | Creation Time       | Updated Time |
+--------------------------------------+-------------+-----------------+---------------------+--------------+
| 17ac22e9-914d-4cf8-b4af-84cf334c0a16 | hello_faafo | CREATE_COMPLETE | 2016-04-28T07:30:32 | None         |
+--------------------------------------+-------------+-----------------+---------------------+--------------+
```

Figure 26: Verifying the stack was successfully created

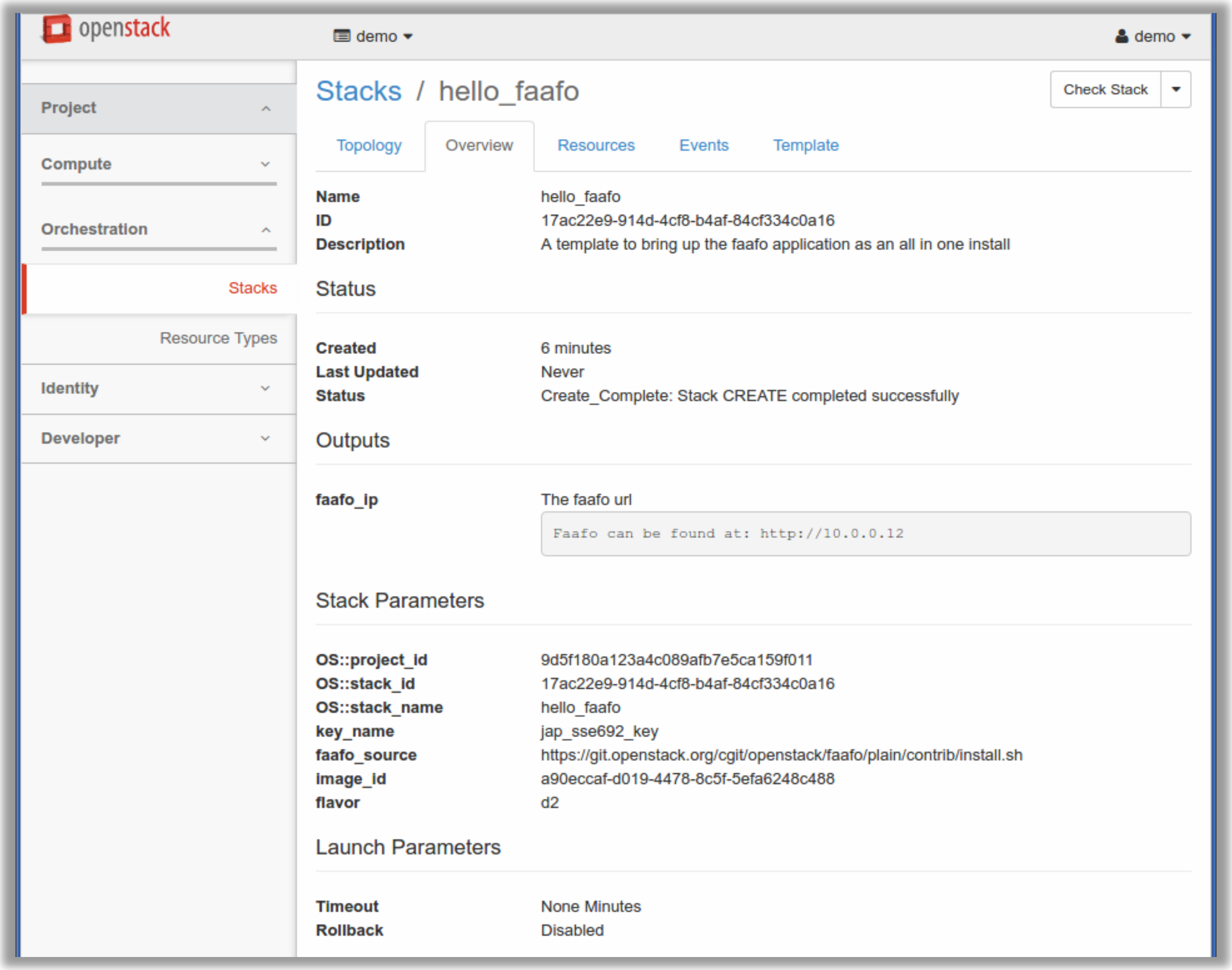


Figure 27: Overview of created Orchestration stack

```
jap:~$ openstack stack show hello_faafo
+-----------------------+--------------------------------------------------------------------------------------------------------------------------+
| Field                 | Value                                                                                                                    |
+-----------------------+--------------------------------------------------------------------------------------------------------------------------+
| id                    | 17ac22e9-914d-4cf8-b4af-84cf334c0a16                                                                                     |
| stack_name            | hello_faafo                                                                                                              |
| description           | A template to bring up the faafo application as an all in one install                                                    |
|                       |                                                                                                                          |
| creation_time         | 2016-04-28T07:30:32                                                                                                      |
| updated_time          | None                                                                                                                     |
| stack_status          | CREATE_COMPLETE                                                                                                          |
| stack_status_reason   | Stack CREATE completed successfully                                                                                      |
| parameters            | OS::project_id: 9d5f180a123a4c089afb7e5ca159f011                                                                         |
|                       | OS::stack_id: 17ac22e9-914d-4cf8-b4af-84cf334c0a16                                                                       |
|                       | OS::stack_name: hello_faafo                                                                                              |
|                       | faafo_source: https://git.openstack.org/cgit/openstack/faafo/plain/contrib/install.sh                                    |
|                       | flavor: d2                                                                                                               |
|                       | image_id: a90eccaf-d019-4478-8c5f-5efa6248c488                                                                           |
|                       | key_name: jap_sse692_key                                                                                                 |
|                       |                                                                                                                          |
| outputs               | - description: The faafo url                                                                                             |
|                       |   output_key: faafo_ip                                                                                                   |
|                       |   output_value: 'Faafo can be found at: http://10.0.0.12'                                                                |
|                       |                                                                                                                          |
| links                 | - href: http://192.168.1.100:8004/v1/9d5f180a123a4c089afb7e5ca159f011/stacks/hello_faafo/17ac22e9-914d-4cf8-b4af-84cf334c0a16 |
|                       |   rel: self                                                                                                              |
|                       |                                                                                                                          |
| disable_rollback      | True                                                                                                                     |
| parent                | None                                                                                                                     |
| tags                  | null                                                                                                                     |
|                       | ...                                                                                                                      |
|                       |                                                                                                                          |
| stack_user_project_id | fbd32d3ed4734016a0d4eeb79cf3ea3b                                                                                         |
| capabilities          | []                                                                                                                       |
| notification_topics   | []                                                                                                                       |
| timeout_mins          | None                                                                                                                     |
| stack_owner           | None                                                                                                                     |
+-----------------------+--------------------------------------------------------------------------------------------------------------------------+
```

Figure 28: Overview of created stack from Orchestration CLI

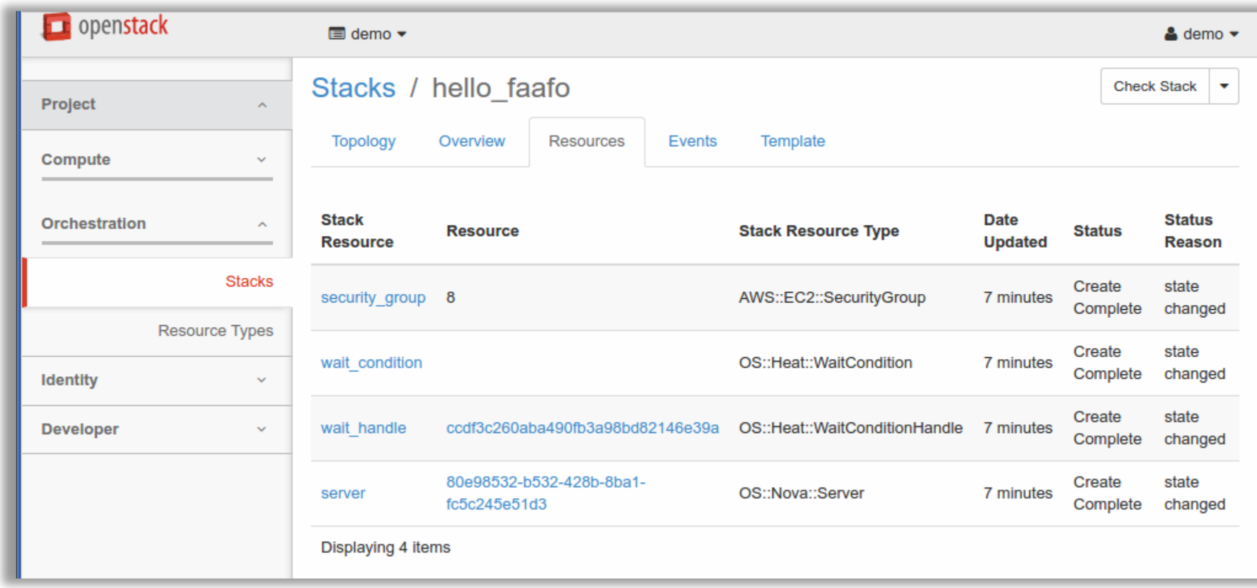
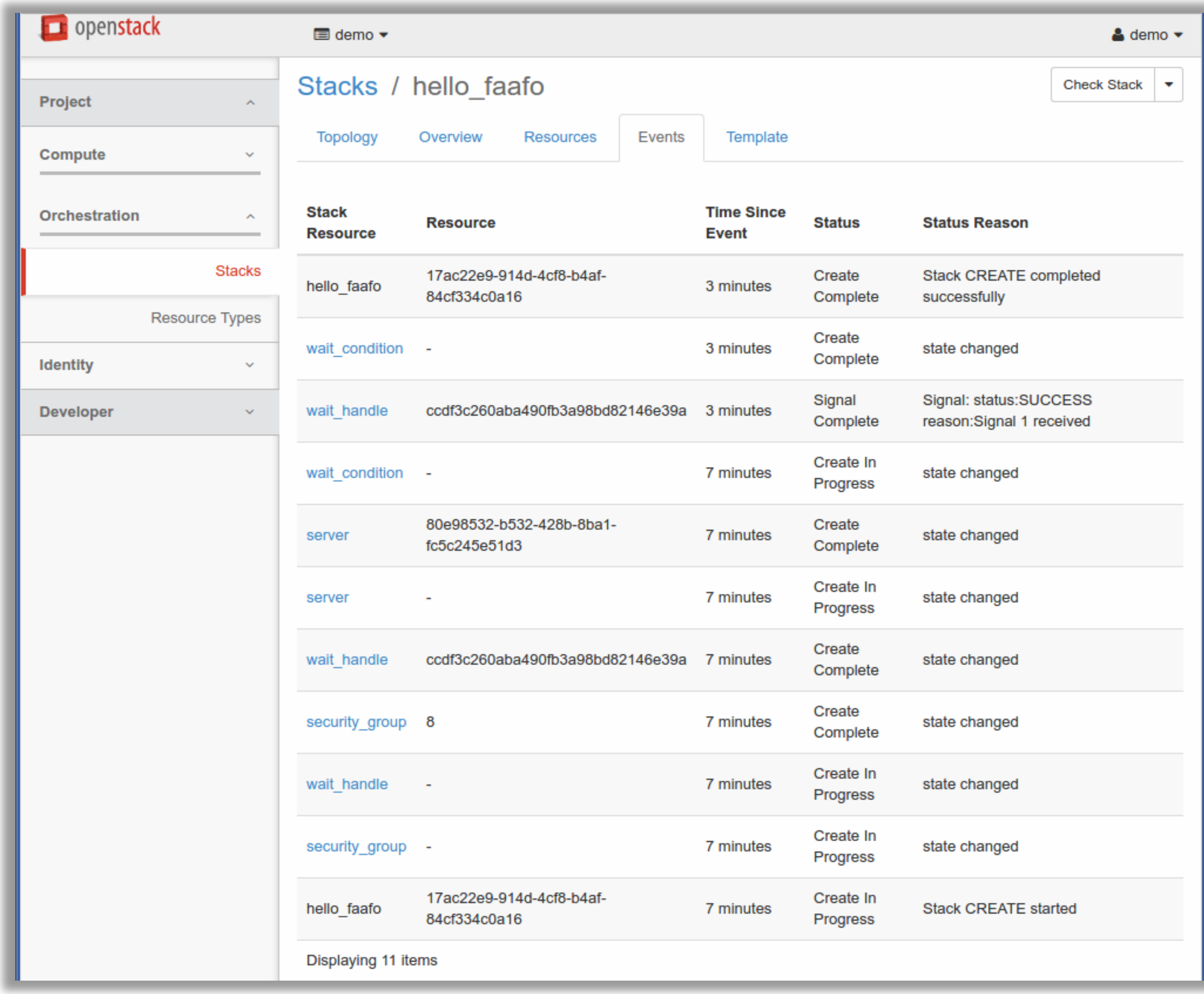


Figure 29: Resources in use by Orchestration stack

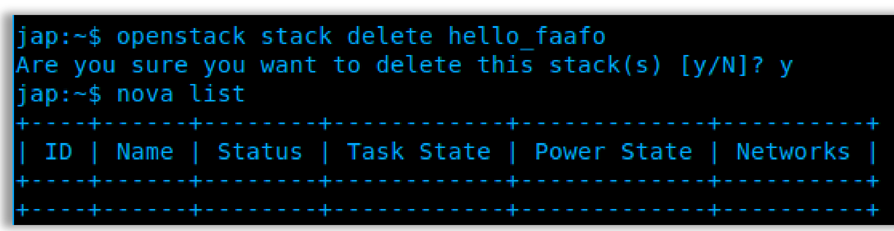
```
jap:~$ nova list
+--------------------------------------+-----------------------------------+--------+------------+-------------+-------------------+
| ID                                   | Name                              | Status | Task State | Power State | Networks          |
+--------------------------------------+-----------------------------------+--------+------------+-------------+-------------------+
| 80e98532-b532-428b-8ba1-fc5c245e51d3 | hello_faafo-server-sgx7tqoqnils   | ACTIVE | -          | Running     | private=10.0.0.12 |
+--------------------------------------+-----------------------------------+--------+------------+-------------+-------------------+
```

Figure 30: Displaying the nova instance created by the stack from the Orchestration CLI



Stack Resource	Resource	Time Since Event	Status	Status Reason
hello_faafo	17ac22e9-914d-4cf8-b4af-84cf334c0a16	3 minutes	Create Complete	Stack CREATE completed successfully
wait_condition	-	3 minutes	Create Complete	state changed
wait_handle	ccdf3c260aba490fb3a98bd82146e39a	3 minutes	Signal Complete	Signal: status:SUCCESS reason:Signal 1 received
wait_condition	-	7 minutes	Create In Progress	state changed
server	80e98532-b532-428b-8ba1-fc5c245e51d3	7 minutes	Create Complete	state changed
server	-	7 minutes	Create In Progress	state changed
wait_handle	ccdf3c260aba490fb3a98bd82146e39a	7 minutes	Create Complete	state changed
security_group	8	7 minutes	Create Complete	state changed
wait_handle	-	7 minutes	Create In Progress	state changed
security_group	-	7 minutes	Create In Progress	state changed
hello_faafo	17ac22e9-914d-4cf8-b4af-84cf334c0a16	7 minutes	Create In Progress	Stack CREATE started

Figure 31: Orchestration stack events



```
jap:~$ openstack stack delete hello_faafo
Are you sure you want to delete this stack(s) [y/N]? y
jap:~$ nova list
+----+------+--------+------------+-------------+----------+
| ID | Name | Status | Task State | Power State | Networks |
+----+------+--------+------------+-------------+----------+
+----+------+--------+------------+-------------+----------+
```

Figure 32: Deleting the stack and verifying that it no longer exists

Non-Direct Activity Report

Date	Duration (minutes)	Specific Task / Activity
23-Mar-2016	180	Work on project #3
26-Mar-2016	273	Work on project #3
2-Apr-2016	32	Work on project #3
21-Apr-2016	85	Work on project #3
22-Apr-2016	182	Work on project #3
23-Apr-2016	157	Work on project #3
24-Apr-2016	452	Work on project #3
25-Apr-2016	34	Work on project #3
27-Apr-2016	297	Work on project #3
28-Apr-2016	360	Work on project #3
29-Apr-2016	150	Work on project #3
23-Mar-2016	180	Work on project #3
26-Mar-2016	273	Work on project #3
2-Apr-2016	32	Work on project #3
21-Apr-2016	85	Work on project #3
22-Apr-2016	182	Work on project #3
23-Apr-2016	157	Work on project #3
24-Apr-2016	452	Work on project #3
25-Apr-2016	34	Work on project #3
27-Apr-2016	297	Work on project #3
28-Apr-2016	360	Work on project #3
29-Apr-2016	150	Work on project #3
Sum for Report #1	**1511**	**/ 1500 (5 weeks @ 300/wk)**
Sum for Report #2	**3869**	**/ 1500 (5 weeks @ 300/wk)**
Sum for Report #3	**2202**	**/ 1500 (5 weeks @ 300/wk)**
Sum for Class	**7582**	**/ 4500 (15 weeks @ 300/wk)**