OpenStack API Quick Start

Table of Contents

The OpenStack system has several key projects that are separate installations but can work together depending on your cloud needs: OpenStack Compute, OpenStack Object Storage, OpenStack Identity Service, and OpenStack Image Store. With a standalone OpenStack installation such as the Rackspace Rapid Deployment Program, the OpenStack Compute, OpenStack Identity, and OpenStack Image Store projects are all working together in the background of your installation.

OpenStack API Introduction

This page covers the basics for talking to your OpenStack cloud through the Compute API after authorizing with the Identity Service API. You can then build a cloud by launching images and assigning metadata to instances, all through the API. For an API reference of all the possible commands, see the OpenStack Compute API 1.1 specification and the Identity Service 2.0 Developer Guide published at docs.openstack.org.

Getting Credentials

Credentials are a combination of your username, password, and what tenant (or project) your cloud is running under. You only need to generate an additional token if you are interacting with your cloud directly with API endpoints, and not with a client. Your cloud administrator can give you a username and a password as well as your tenant identifier so you can generate authorization tokens. These tokens are typically good for 24 hours, and when the token expires, you will find out with a 401 (Unauthorized) error and can request another token programmatically. The general workflow goes something like this:

1. Begin API requests by asking for an authorization token from the endpoint your cloud administrator gave you, typically http://hostname:5000/v2.0/tokens. You send your username, password, and what group you are with (the "tenant" in auth-speak).

```
$ curl -X 'POST' -v http://hostname:5000/v2.0/tokens -d '{"auth":
{"passwordCredentials":{"username": "joecool", "password":"coolword"},
 "tenantId":"5"}}' -H 'Content-type: application/json'
```

2. The server returns a response in which the 24-hours token is contained :

```
* About to connect() to host.na.me port 5000 (#0)
*   Trying 55.51.11.198... connected
* Connected to host.na.me (55.51.11.198) port 5000 (#0)
> POST /v2.0/tokens HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7
 OpenSSL/0.9.8r zlib/1.2.3
> Host: host.na.me:5000v> Accept: */*
> Content-type: application/json
> Content-Length: 95
>
> HTTP/1.1 200 OK
> Content-Type: application/json; charset=UTF-8
> Content-Length: 935
> Date: Thu, 06 Oct 2011 19:59:49 GMT
>
* Connection #0 to host host.na.me left intact
* Closing connection #0
{"access": {"token": {"expires": "2011-10-07T14:59:49.644963", "id":
 "e83abbdc-a8c8-4f81-897b-541cbcd1dbb5", "tenant": {"id": "5", "name":
 "coolu"}}, "serviceCatalog": [{"endpoints": [{"adminURL": "http://55.
51.11.198:8774/v1.1/5", "region": "RegionOne", "internalURL": "http://55.
51.11.198:8774/v1.1/5", "publicURL": "http://55.51.11.198:8774/v1.1/5"}],
 "type": "compute", "name": "nova"}, {"endpoints": [{"adminURL": "http://
55.51.11.198:9292/v1.1/5", "region": "RegionOne", "internalURL": "http://
55.51.11.198:9292/v1.1/5", "publicURL": "http://55.51.11.198:9292/v1.1/5"}],
 "type": "image", "name": "glance"}, {"endpoints": [{"adminURL": "http:/
/55.51.11.198:35357/v2.0", "region": "RegionOne", "internalURL": "http:/
/55.51.11.198:5000/v2.0", "publicURL": "http://55.51.11.198:5000/v2.0"}],
 "type": "identity", "name": "keystone"}], "user": {"id": "4", "roles":
 [{"tenantId": "5", "id": "2", "name": "Member"}], "name": "joecool"}}
```

Use that token to send API requests with the X-Auth-Token included as an header field.

3. Repeatedly send API requests with that token in the x-auth-token header until either: 1) the job's done or 2) you get a 401 (Unauthorized) code in return.

4. Request a token again when you get a 401 response until the script's job is done.

For a typical OpenStack deployment you can request a token with this command in cURL:

```
$ curl -X 'POST' -v http://hostname:5000/v2.0/tokens -d
 '{"passwordCredentials":{"username": "joecool", "password":"coolword",
 "tenantId":"coolu"}}' -H 'Content-type: application/json'
```

In return, you should get a 200 OK response with a token in the form of "id": "cd427a33-bb4a-4079-a6d7-0ae148bdeda9" and an expiration date 24 hours from now. Here's what it looks like:

```
* About to connect() to host.na.me port 5000 (#0)
*   Trying hostname... connected
* Connected to host.na.me (hostname) port 5000 (#0)
> POST /v2.0/tokens HTTP/1.1
```

```
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/
0.9.8r zlib/1.2.3
> Host: host.na.me:5000
> Accept: */*
> Content-type: application/json
> Content-Length: 85
>
> HTTP/1.1 200 OK
> Content-Type: application/json; charset=UTF-8
> Content-Length: 1213
> Date: Thu, 01 Sep 2011 19:27:30 GMT
>
* Connection #0 to host host.na.me left intact
* Closing connection #0
{"auth": {"token": {"expires": "2011-09-02T14:27:30.597385", "id": "cd427a33-
bb4a-4079-a6d7-0ae148bdeda9"}, "serviceCatalog": {"nova_compat": [{"adminURL":
 "http://hostname:8774/v1.0", "region": "RegionOne", "internalURL":
 "http://hostname:8774/v1.0", "publicURL": "http://hostname:8774/v1.
0/"}], "nova": [{"adminURL": "http://hostname:8774/v1.1", "region":
 "RegionOne", "internalURL": "http://hostname:8774/v1.1", "publicURL": "http:/
/hostname:8774/v1.1"}], "keystone": [{"adminURL": "http://hostname:5001/
v2.0", "region": "RegionOne", "internalURL": "http://hostname:5000/v2.0",
 "publicURL": "http://hostname:5000/v2.0"}], "glance": [{"adminURL": "http:/
/hostname:9292/v1.1/tenant", "region": "RegionOne", "internalURL": "http://
hostname:9292/v1.1/tenant", "publicURL": "http://hostname:9292/v1.1/tenant"}],
 "swift": [{"adminURL": "http://hostname:8080/", "region": "RegionOne",
 "internalURL": "http://hostname:8080/v1/AUTH_tenant", "publicURL":
 "http://hostname:8080/v1/AUTH_tenant"}], "identity": [{"adminURL": "http:/
/hostname:5001/v2.0", "region": "RegionOne", "internalURL": "http://
hostname:5000/v2.0", "publicURL": "http://hostname:5000/v2.0"}]}}}
```

Sending Requests to the API

You have a couple of options for sending requests to OpenStack through an API. Developers and testers may prefer to use cURL, the command-line tool from http://curl.haxx.se/. With cURL you can send HTTP requests and receive responses back from the command line.

If you like to use a more graphical interface, the REST client for Firefox also works well for testing and trying out commands, see https://addons.mozilla.org/en-US/firefox/addon/restclient/. You can also download and install rest-client, a Java application to test RESTful web services, from http://code.google.com/p/rest-client/.

You need to generate a token as shown above if you use cURL or a REST client.

For more serious scripting work, you can use a client like the python-novaclient or openstack-compute clients. The python-novaclient implements the Compute 1.1 API while the openstack-compute client works against the Rackspace Cloud Servers public cloud which is the OpenStack Compute 1.0 API. You only need a username and password to use the python-novaclient tool.

Setting Up python-novaclient

Installing the python-novaclient gives you a `nova` shell command that enables Compute API interactions from the command line. You install the client, and then provide your username

and password, set as environment variables for convenience, and then you can have the ability to send commands to your cloud on the command-line.

To install python-novaclient, download the tarball from http://pypi.python.org/pypi/python-novaclient/2.6.3#downloads and then install it in your favorite python environment.

```
$ curl -O http://pypi.python.org/packages/source/p/python-novaclient/python-
novaclient-2.6.3.tar.gz
$ tar -zxvf python-novaclient-2.6.3.tar.gz
$ cd python-novaclient-2.6.3
$ sudo python setup.py install
```

Now that you have installed the python-novaclient, confirm the installation by entering:

```
$ nova help

usage: nova [--username USERNAME] [--apikey APIKEY] [--projectid PROJECTID]
            [--url URL] [--version VERSION]
            <subcommand> ...
```

In return, you will get a listing of all the commands and parameters for the nova command line client. By setting up the required parameters as environment variables, you can fly through these commands on the command line. You can add –username on the nova command, or set them as environment variables:

```
export NOVA_USERNAME=joecool
export NOVA_API_KEY=coolword
export NOVA_PROJECT_ID=coolu
```

Using the Identity Service, you are supplied with an authentication endpoint, which nova recognizes as the NOVA_URL.

```
export NOVA_URL=http://hostname:5000/v2.0
export NOVA_VERSION=1.1
```

Listing Images

Before you can go about the business of building your cloud, you want to know what images are available to you by asking the image service what kinds of configurations are available. The image service could be compared to iTunes for your cloud - you can view the playlist of images before using your favorite image to create a new instance in the cloud. To get the list of images, their names, status, and ID, use this command:

```
$ nova image-list

+----+---------------------------------------------------------+--------+
| ID |                          Name                           | Status |
+----+---------------------------------------------------------+--------+
| 1  | aki-tty                                                 | ACTIVE |
| 2  | ari-tty                                                 | ACTIVE |
```

```
| 3  | ami-tty                                                 | ACTIVE |
| 6  | CentOS_5.4_x64                                          | ACTIVE |
| 14 | maverick-kernel                                         | ACTIVE |
| 15 | maverick                                                | ACTIVE |
| 20 | ubuntu-kernel                                           | ACTIVE |
| 21 | ubuntu-ramdisk                                          | ACTIVE |
| 22 | ubuntu                                                  | ACTIVE |
| 24 | CentOS_5.6_x64_v5.7.14_Dev1                             | ACTIVE |
+----+---------------------------------------------------------+--------+
```

Next you need to know the relative sizes of each of these.

```
$ nova flavor-list

+----+-----------+-----------+------+----------+-------+------------
+----------+
| ID |    Name   | Memory_MB | Swap | Local_GB | VCPUs | RXTX_Quota | RXTX_Cap
 |
+----+-----------+-----------+------+----------+-------+------------
+----------+
| 1  | m1.tiny   | 512       |      | 0        |       |            |
 |
| 2  | m1.small  | 2048      |      | 20       |       |            |
 |
| 3  | m1.medium | 4096      |      | 40       |       |            |
 |
| 4  | m1.large  | 8192      |      | 80       |       |            |
 |
| 5  | m1.xlarge | 16384     |      | 160      |       |            |
 |
| 89 | Boom      | 2         |      | 20       |       |            |
 |
+----+-----------+-----------+------+----------+-------+------------
+----------+
```

You can also narrow down the list by using grep to find only the CentOS images with a command like this:

```
$ nova image-list | grep 'CentOS'

| 6  | CentOS_5.4_x64                                          | ACTIVE |
| 24 | CentOS_5.6_x64_v5.7.14_Dev1                             | ACTIVE |
```

With the information about what is available to you, you can choose the combination of image and flavor to create your virtual servers and launch instances.

Launching Instances

To launch a server, you choose an image you want to match up to a size, find the ID for the image and the ID for the flavor so you can size it, and create the command with the IDs. From the information we got previously, we know that an Ubuntu Maverick image has

an ID of 15, and if you want to start small with about 2 GB of memory and 20 GB of disk space, you'd choose the m1.small flavor which has an ID of 2. Put those parameters in with the "boot" command and you can create a new virtual server.

```
 $ nova boot --flavor=2 --image=15 testserver

 +-----------+--------------------------------------+
|  Property |                Value                 |
+-----------+--------------------------------------+
| adminPass | ****************                     |
| created   | 2011-09-01T21:40:41Z                 |
| flavor    | m1.small                             |
| hostId    |                                      |
| id        | 1805                                 |
| image     | maverick                             |
| metadata  | {}                                   |
| name      | testserver                           |
| progress  | 0                                    |
| status    | BUILD                                |
| updated   | 2011-09-01T21:40:41Z                 |
| uuid      | ce044452-f22e-4ea4-a3ec-d1cde80cf996 |
+-----------+--------------------------------------+
```

Now, you can view this server in your new cloud by using the nova list command:

```
$ nova list
+------+------------+--------+--------------------------------+
|  ID  |    Name    | Status |            Networks            |
+------+------------+--------+--------------------------------+
| 1805 | testserver | ACTIVE | private=10.4.96.81             |
+------+------------+--------+--------------------------------+
```

There are three statuses you may see - ACTIVE, BUILDING, and UNKNOWN. The BUILDING status is transient and you likely will not see it. If you see UNKNOWN, run `nova list` again until it goes away.

To view all the information about a particular server, use nova show with the ID of the server that you got from the nova list command.

```
$ nova show 1805

+-----------------+------------------------------------------------------------+
|     Property    |                           Value                            |
+-----------------+------------------------------------------------------------+
| created         | 2011-09-01T21:40:41Z                                       |
| flavor          | m1.small                                                   |
| hostId          | 58a7430169aa42cde5ce2456b0cb5bb5ac1ab0703bab6420e8a49e6e |
| id              | 1805                                                       |
| image           | maverick                                                   |
| metadata        | {}                                                         |
| name            | testserver                                                 |
| private network | 10.4.96.81                                                 |
| progress        | 100                                                        |
| status          | ACTIVE                                                     |
```

```
| updated          | 2011-09-01T21:40:46Z                                    |
| uuid             | ce044452-f22e-4ea4-a3ec-d1cde80cf996                    |
+------------------+---------------------------------------------------------+
```

You can now launch that image again, but add more information to the server when you boot it so that you can more easily identify it amongst your ever-growing elastic cloud. Use the -meta option with a key=value pair, where you can make up the string for both the key and the value. For example, you could add a description and also the creator of the server.

```
$ nova boot testserver --meta description='Use for testing purposes' --meta
 creator=joecool
```