Command Line Tools

Much of the power of OpenStack is in the suite of application program interfaces (APIs), allowing the same functionality of the dashboard with the advantage of automating the provisioning process with only a few commands. Using the API means that all the actions can be added to a script that can be called to automatically and repeatedly build instances. You can also create template-like scripts so that with only minor modification to instance name, image and volume size, it is possible to build other instances and volumes quickly and without the need for clicking through the dashboard.

Installing command-line tools (Ubuntu)

Before the tools are installed, you will want an environment that can use Python, as all of the OpenStack command-line tools are written in that language. This can be your local desktop or it could be a virtual machine in the Rapid Access Cloud. The instructions outlined below will assume a Ubuntu 14.04 instance in the Rapid Access Cloud, a virtual machine running on your desktop or a native Ubuntu desktop.

From an Ubuntu command-line:

1. Install Python package management (pip) and required packages:

```
$ sudo apt-get install python3-pip python3-dev
```

2. Install Openstack command-line tools using pip:

```
$ sudo pip3 install python-openstackclient
```

Installing command-line tools (macOS)

1. Ensure you have the Developer Tools (Xcode) with the Command Line Tools for macOS:

```
$ sudo xcode-select --install
$ sudo pip3 install --upgrade pip
```

2. Open the Terminal application.
3. Install the openstack command line tool:

```
$ pip3 install --user python-openstackclient
```

   Note: We recommend installing this as a user to avoid running into issues with system updates.
4. Add the following to .zshrc:

```
export PATH=/Users/<username>/Library/Python/3.8/bin:$PATH
export PYTHONPATH=/Users/<username>/Library/Python/3.8/lib/python/site-packages
```

Openrc file

The "rc" in rc file stands for any or all of the following:

- run commands
- resource control
- run control
- runtime configuration

Using an rc file to outline your Rapid Access Cloud credentials allows you to pass your credentials to the cloud without needing to include them in your scripts. In fact, the rc file is a simple bash script that creates *environment variables* in your shell to be used by the `openstack` command (the 'open' in openrc). The environment variables, once set, exist only for the current session. If you open another concurrent session on the computer or log-out and back in, you will need to source the openrc file again.

You can download a pre-generated openrc file from the dashboard, or create one if you are unable to download from the dashboard.

Download openrc

1. Log-in to the Rapid Access Cloud dashboard at https://cloud.cybera.ca.
2. In the left-hand panel under “Compute”.
3. Click the “API Access” tab at the top then click the “Download Openstack RC file v3” on the right.
4. The file downloaded will be named after your account name, e.g. <email>@<domain>-openrc.sh. This can be renamed to openrc if you like, or you can copy it and modify it so you can have one for the Calgary region and one for the Edmonton region. The rc file downloaded will be keyed for the region you are signed into in the dashboard.

Create openrc

You can use the downloaded rc file as a template or create one from scratch. The OS_USERNAME and OS_PROJECT_NAME variables are the same; each will be the email address for your Rapid Access Cloud account.

Copy and paste the region-appropriate rc file. This openrc file will require you to enter your password each time you source the file, however you can modify the file if you like to include the password.

Keep this file secure if you are keeping your password in plain text!

Calgary openrc

```
#!/bin/bash
export OS_AUTH_URL=https://yyc.cloud.cybera.ca:5000/v3
export OS_PROJECT_NAME="<account_username>"
export OS_USERNAME="<account_username>"
export OS_USER_DOMAIN_NAME="Default"
export OS_REGION_NAME="Calgary"
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
echo "Please enter your OpenStack Password for project $OS_PROJECT_NAME as user $OS_USERNAME: "
read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT
```

Edmonton openrc

```
#!/bin/bash
export OS_AUTH_URL=https://yeg.cloud.cybera.ca:5000/v3
export OS_PROJECT_NAME="<account_username>"
export OS_USERNAME="<account_username>"
export OS_USER_DOMAIN_NAME="Default"
export OS_REGION_NAME="Edmonton"
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
echo "Please enter your OpenStack Password for project $OS_PROJECT_NAME as user $OS_USERNAME: "
read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT
```

Sourcing the rc File

Whether you downloaded the RC file or created your own, you will need to "source" into your shell environment:

```
source /path/to/openrc/file
```

Using the `openstack` client

This tutorial will walk through some of the steps outlined in the Basic Guide in building a single instance from scratch, with some notes about alternative functionality that each command offers, to show how flexible using the OpenStack command line client is.

The structure of the openstack command is fairly logical and intuitive. The OpenStack documentation project has a comprehensive outline on how the command is used, and help is available through the command for quick reference:

```
$ openstack help
```

Before getting started, be sure to source your openrc file.

Key Pairs

- A key pair is required before launching an instance, as the public key needs to be injected into the instance during the provisioning process. The following command will generate the pair, and output the private key:

  ```
  $ openstack keypair create <key_name> >> my_key.pem
  ```

- Or perhaps you have an existing RSA key pair that you would like to use instead:

  ```
  $ openstack keypair create --public-key </path/to/public_key> <key_name>
  ```

- Deleting a key pair is accomplished with:

  ```
  $ openstack keypair delete <key_name>
  ```

- And to list existing key pairs (you guessed it):

  ```
  $ openstack keypair list
  ```

Security Groups

- As outlined in the Basic Guide, modifying the default security group to permit ICMP and ssh means that you don't need to assign specific security groups to an instance in order to do simple troubleshooting (ping or traceroute) or gain access:

  ```
  $ openstack security group rule create --proto icmp --src-ip 0.0.0.0/0 default
  $ openstack security group rule create --proto icmp --src-ip ::/0 default
  $ openstack security group rule create --proto tcp --src-ip 0.0.0.0/0 --dst-port 22 default
  $ openstack security group rule create --proto tcp --src-ip ::/0 --dst-port 22 default
  ```

- Adding too many rules to the default group is not good security practice, so create separate groups for instances with different roles and requirements:

  ```
  $ openstack security group create <security_group_name>
  ```

- To remove an existing group rule from a group (without deleting the whole group) the rule ID is needed:

```
$ openstack security group rule list <security_group_name>
+-------+-------------+-----------+------------+-----------------------+
|    ID | IP Protocol | IP Range  | Port Range | Remote Security Group |
+-------+-------------+-----------+------------+-----------------------+
|  3737 | icmp        | 0.0.0.0/0 |            |                       |
|  3822 | icmp        | ::/0      |            |                       |
| 16374 | tcp         | 0.0.0.0/0 | 22:22      |                       |
| 16378 | tcp         | ::/0      | 22:22      |                       |
+-------+-------------+-----------+------------+-----------------------+
```

- Delete existing group rules with the rule ID. The security group name is not required, as the rule ID is unique across all security groups:

```
$ openstack security group rule delete <rule_id>
```

- When an entire security group is no longer required, the whole group is deleted with:

```
$ openstack security group delete <security_group_name>
```

Instances

Most of the commands for manipulating and managing instances is with `openstack server`. This command is quite extensive, allowing creation, deletion, starting, stopping, adding/removing security groups, adding/removing volumes and more. Refer to the OpenStack server command-object documentation for more information. `openstack flavor list, openstack flavor show <flavor_name>` and `openstack image list, openstack image show ‘<image_name>’` can provide information as to what flavors and images are available.

- Launch a new ‘small’ instance from an Ubuntu 14.04 image:

```
$ openstack server create --flavor m1.small --image ‘Ubuntu 14.04’ --key-name <key_pair_name>
<instance_name>
```

- Add security groups to an instance:

```
$ openstack server add security group <instance_name> <security_group_name>
```

- Stop, start and reboot instances:

```
$ openstack server stop <instance_name>,
$ openstack server start <instance_name>,
$ openstack server reboot <instance_name>
```

- Attach a volume:

```
$ openstack server add volume <instance_name> <volume_name>
```

- Detach a volume. This is not a destructive action and data on the volume is left intact:

```
$ openstack server remove volume <instance_name> <volume_name>
```

Floating IPs

As discussed previously, networking with respect to the assignment of IP addresses in the Rapid Access Cloud is handled automatically during provisioning, with the exception of public IPv4 addresses referred to as *floating IPs* in OpenStack. There is a single pool that floating IPs are allocated from, named ‘nova’ and by default each project is permitted a single IP from this pool:

- Allocate a floating IP to your project:

```
$ openstack ip floating create nova
```

- Assign an allocated floating IP to an instance:

```
$ openstack ip floating add <ip_address> <instance_name>
```

- List IP addresses allocated to project:

```
$ openstack ip floating list
```

- Remove floating IP from an instance:

```
$ openstack ip floating remove <ip_address> <instance_name>
```

- Return allocated IP address to pool:

```
$ openstack ip floating delete <ip_address>
```

Volumes

OpenStack instances are intended to be *ephemeral*, that is the storage associated with the instance is not meant to retain data for any length of time.

The root storage for an instance is solely for volatile application data such as kernel operations and applications. Rapid Access Cloud users are provided with a default 500GB of storage that can be used in the form of volumes to be attached to instances and treated no differently by the instance than an internal hard drive would be to a bare-metal server. Attaching volumes to instances is handled by the openstack server command detailed under managing instances. For volume backup, see managing snapshots and images.

- Create a volume:

```
$ openstack volume create --size 25 <volume_name>
```

- Clone an existing volume:

```
$ openstack volume create --source <existing_volume_id_or_name> <volume_name>
```

- Create a volume from a snapshot (see snapshots and images below):

```
$ openstack volume create --snapshot <snapshot_id_or_name> <volume_name>
```

- Delete a volume. This is a destructive action and all data associated with the volume is lost:

```
 $ openstack volume delete <volume_name>
```

Images

While it is true that instances in the Rapid Access Cloud are intended to be ephemeral, it is still possible to capture the state of an instance at any moment and save it as an image that can in turn be provisioned in a similar fashion to deploying a new instance using `openstack server create` as outlined above. This permits the ability to back-up an instance for safe keeping or create a base image with required applications already installed if you want to rapidly deploy many instances with a similar role.

- Create an image of an existing instance:

```
$ openstack server image create --name <image_name> <instance_name>
```

- Download an image to a local computer:

```
 $ openstack image save --file <file_name> <image_name>
```