Module 10

Beyond the Dashboard

Introduction

This module is designed for advanced users who want to learn more about command line tools to control the NeCTAR resources. This can help to automate procedures, for example with scripts for backup processes. Worked examples of the OpenStack command line tool *openstack* are included in this Module.

The previous modules have covered all you need to know to get started with the Research Cloud — this module is going to show how all of this can be done using only a few command line tools. If you are not comfortable with the command line, rest assured that you already have all knowledge required to work with the Cloud using the Dashboard and the other tools discussed in previous Modules. If you like using the command line to get things done, you will *love* the extra information given in this module!

The following topics are going to be covered:

- Installation of OpenStack command line clients
- Launching and terminating an instance
- Taking snapshots of instances and relaunching from snapshots
- Creating and deleting volumes
- Making backups and taking snapshots of volumes
- Accessing the object store

Videos

The following videos go through most of the content in this module and offer a less in-depth description of the subject than the documentation does.

https://www.youtube.com/watch?v=t1nuK45_h0M

https://www.youtube.com/watch?v=4jrzt_fSzjo

https://www.youtube.com/watch?v=4_nWStMJw7Q

Conventions

The notation throughout the training documents can be interpreted as follows:

Words in *italics* are used for names and terminology, e.g. name of a software, or name of a computing concept. It may also just emphasise a word in the traditional way. Quotations are also written in italics and are put in between quotatioin marks.

Words in **bold** are used to highlight words which identify important concepts of a paragraph, to make it easier for users to skim through the text to find a paragraph which explains a certain idea, concept or technology.

> **Additional information** which is optional to read is displayed in *info boxes* like this one.

> **Important information** is displayed in boxes like this one.

> **Definition of terms** are displayed in boxes of this style.

> Possibly specific **prerequisites** for reading a particular section are contained in this type of box at the beginning of a section.

The use of command line tools is part of this course. In a *Terminal*, you will be directed to type in *commands*. Commands are formatted as follows:

```
command-name argument1 argument2 ... argumentn
```

Throughout the exercises, when you see a word in pointed brackets, like <*this-word*>, it means that you have to replace everything inside the brackets, and *including the brackets*, with whatever is described within the brackets.

For example, if you are prompted to run a command named *the-command* which takes two arguments:

```
the-command -f <yourfile>
```

Then you have to replace the second argument, <yourfile>, with the *file name* that is

being referenced in the exercise. For example

`the-command -f thefile.txt`

When editing a file, the contents of it will be displayed in a different font and with background colour as follows:

```
The content of the file
The next line in this file
```

Output on the command line terminal is printed in boxes formatted as follows:

```
NectarInstance:~ Jennifer$ whoami
Jennifer
```

OpenStack clients

In the previous modules, we have used the NeCTAR Dashboard to perform simple management tasks such as launching instances, creating volumes, accessing the object store, etc. In this module we are going to take a look at how you can do the same tasks with other *"clients"*.

- OpenStack provides **command line clients** which allow you to manage your instances, volumes and object store and more via command line tools only. In this tutorial, we will discuss usage of command line clients to control your resources.
- There are also **application programming interfaces (APIs)** available, mainly for Python, but also for other programming languages like e.g. C++ and Java. This is not part of this tutorial — for a list of known software development kits see OpenStack Python API Bindings and other OpenStack SDKs.

In a previous version of OpenStack, command line clients were split into several tools, called **nova, glance, cinder, swift, keystone** and more. Each of the tools was dedicated to certain tasks, e.g. *nova* for compute services, *swift* for accessing the object store, etc. Refer to the official OpenStack clients website for more details. These tools have later been brought together to **one** command line tool called **openstack**, which brings command sets for Compute, Identity, Image, Object Store and Volume APIs together in a single command set with a uniform command structure.

The next sections are going to discuss installation and usage of the ***openstack* command line client** with some working examples.

If you want to use the traditional OpenStack clients like *glance, cinder*, etc.

you still can. The following instructions for installation and configuration also apply to the traditional clients. Only the name *openstack* in the package name *python-openstackclient* has to be replaced by the respective tool's name in the installation command. For example, if you want to install *glance*, replace *python-openstackclient* by *python-glanceclient*.

You may install the command line tools on **any computer** connected to the Internet — this may be a Linux, Mac or Windows computer. Windows users must have access to the Windows command line (it may sometimes be blocked to users without administrator privileges). The following instructions are going to assume you are familiar with using the command line.

Of course, you may also install the clients on your **Ubuntu instance** — follow the instructions for a Linux system below and use the ssh terminal to your instance to type the commands.

Step 1. Installation of clients

This section is going to cover installation of the OpenStack command line clients. Installation for all packages is the same, only the package name has to be replaced. Please also refer to the Official OpenStack documentation for more information about installation of the OpenStack command line clients.

You will need to have Python installed to run the OpenStack command line clients. You may already have python installed. If you do, check your version:

```
python --version
```

You will need to have **at least version 2.7**, however **python 3 is not supported** yet!

If you don't have Python yet, the following instructions cover the installation as well. When using the Linux package managers, the right version will be installed automatically. On Mac/Windows, you have to manually install it.

- **Linux**: Installation of the OpenStack clients is simple via the built-in package manager: Ubuntu, RDO, openSUSE and SUSE Linux Enterprise have the client packages available through the package manager. For example, on Ubuntu, you may install the package with:

  ```
  sudo apt-get install python-openstackclient
  ```

This will install the *openstack* client with all the depending packages.

> If you need a newer version of an OpenStack client than is available on the Linux package manager, you can use Python's package installer ***pip*** instead. *pip* is a tool for easily installing and managing Python packages.
> If you don't have *pip* yet, do so with your package manager (search for *python-pip*). For example on Ubuntu:
> `sudo apt-get install python-pip`
> After installing *pip*, you can use *pip* to install the client package:
> `sudo pip install python-openstackclient`
> To upgrade the client at a later point:
> `pip install --upgrade python-openstackclient`

- **Mac OS X**:
  1. Install Python's package manager ***pip*** (unless you already have it). *pip* is a tool for easily installing and managing Python packages. It is recommended over *easy_install*. Python 2.7.9 and later include *pip* by default, so you may have *pip* already. Either of the three options work to get *pip*:
     - If you have the homebrew package manager, *pip* installs together with the Python packages, and conveniently also installs the *setuptools* which you also need:
       `brew install python`
     - If you don't have homebrew, the official Python download of package 2.7.10 or later also installs *pip* (*Note*: Python 3 is *not* supported yet).
     - *Alternative:* if you already have Python, but not *pip*, you may use the ***easy_install*** command to install *pip* — however there may be problems with running the clients when using this method. To install *pip* with *easy_install*:
       `sudo easy_install pip`
  2. Install *setuptools* if you don't already have it (*Note*: If you used *homebrew* to install python in the last step, you will already have *setuptools*). Follow instructions on the setuptools Website.
  3. Upgrade *setuptools*
     `sudo pip install --upgrade setuptools`
  4. Install your client
     `sudo pip install python-openstackclient`
     To upgrade the client at a later point:
     `pip install --upgrade python-openstackclient`

- **Windows**:
  1. Install Python 2.7 or later (*Note*: Python 3 is not supported yet!) if you don't have it yet. You can download it on the official Python Website. You may keep the default options in the installation wizard — the following instructions assume that you keep the default installation path *C:\Python27*. It is recommended that you install **Python 2.7.10 or later** which includes the *pip* tool which you will need.
  2. Install *setuptools*: follow the documentation provided on the setuptools website (the easiest is to download the ez_setup.py file and run it).
  3. Open a Windows Command line terminal (in the "Search" field, type *cmd* — on older Windows, you may find the search field when clicking on *Start*).
  4. Make sure you have included the package **pyOpenSSL (version 0.13 or higher)** in your installation: `pip install pyOpenSSL`
  5. Ensure that the *C:\Python27\Scripts* directory is defined in the PATH environment variable. To do this for the currently open terminal run the following command:
     `set PATH=%PATH%;C:\Python27\Scripts`

     > To *permanently* change an environment variable: On various Windows versions, this can be done in the **Advanced System Settings** (usually you can find it in *Control Panel* > *System* or similar), where you should find a button **Environment Variables**. Scroll through the *System variables* and find *PATH* (it's not case senstitive, so it may be *Path*). Select it and click on *"Edit…"*. At the end of the long text in the *Variable value* field, add the text *";C:Python27\Scripts"* (without the quotation marks). Close all windows with [OK].

  6. If you have *not* installed Python 2.7.10 or later *including the pip tool*, you will need to install *pip* separately. You can test whether you have *pip* by typing
     `pip --version`
     in your command line. If a version is printed, you have it. Otherwise, you may use the *easy_install* command from the *setuptools* package:
     `easy_install pip`
  7. Install the client:
     `pip install python-openstackclient`
     To upgrade the client at a later point:
     `pip install --upgrade python-openstackclient`

Step 2. Get your OpenStack credentials

Before you can use any of the clients, you will need to get your *OpenStack credentials*. In particular, you will need a script file called the *OpenStack RC file*. You will need to load the information from within this script before you can use the tools. While you are at it, you may also write down all your OpenStack login details.

You may already have done this in Module 7 when setting up access to your Object Store. In this case, make sure you have downloaded your *OpenStack RC file*, as you will need it now. You may then skip the following step.

You can download your **credentials** from the Dashboard

1. Your **tenant ID** is the name of your project as reported in the left hand panel of the dashboard (e.g. *pt-12345*)

2. Your **username** is reported at the top of the dashboard on the right (e.g, logged in as: *user@uni.edu.au*)

3. Open the *Dashboard API* tab: get there via *Compute > Access & Security > API Access*.

4. Look up what is specified for the *'Identity'* Service in the *Dashboard API* tab. It will be a URL like
   *https://**keystone.rc.nectar.org.au**:**5000**/v2.0/*
   Take note of the text between *https://* and the colon (*keystone.rc.nectar.org.au* in the example). This will be the **Server**. Take note of the number (*5000* in the example). This will be the **Port**. You will need the **Keystone Server and Port** to authenticate your access. Also take note of the keystone version which you are using, in this case its **v2.0**.

5. Get your **credential file** (this is called the *OpenStack RC file* for OpenStack, and *EC2 file* for Amazon). You can download by clicking on the buttons on the top right on the same window: **Download OpenStack RC File**. Download it and save it somewhere on your computer where you can easily find it later.

6. You will also need your **OpenStack password**. This is **not** the same password that you use to log in to the Dashboard! If you have never used the OpenStack password before, you need to generate it first. You can do this in the **Settings** of the Dashboard: click next to your user name (your e-mail) on the top right and a drop down menu will open. Click on *Settings*. You need to reset your password by clicking **Reset Password** on the left panel of the settings. A long combination of numbers and letters will be displayed. This is your OpenStack password. Copy the text and save it somewhere safe.

> Your OpenStack password will be a long combination of numbers. At the time of writing, unfortunately this password cannot be changed yet, but this should be changed in a future update of OpenStack on NeCTAR.

Step 3: Upload your RC file to the instance

> If you are configuring the client access on your ***local computer***, you can ***skip this step***. If you are using the ssh terminal to your instance instead, you need to do this step.

Before you continue, you need to copy the *OpenStack RC file* which you downloaded from the Dashboard to your instance.

You may use any method to copy files to your instance, as discussed in Module 7, e.g. using the FTP client *FileZilla*. In the following instructions, we are going to apply the *command line* option to copy our OpenStack RC file across.

- On **Linux or Mac OS X**, you can use the *scp* command to do this:

  ```
  scp -i ~/.ssh/Nectar_Key <path-to-your-openrc-file>
  ubuntu@NNN.NNN.NNN.NNN:
  ```

  replacing the N's with your IP address. This will have copied your openrc file in your ubuntu instance's *home* directory.

- On **Windows**, you may use the *PuTTY* command line tool *pscp*. Please refer to instructions in Module 7 on how to install and use *pscp*.

  > If you don't feel comfortable using the command line, you may use an FTP client like *FileZilla* to drag and drop your file into your *ubuntu* user's *home* directory (the *home* directory is the directory which opens up first when you connect with FileZilla). Instructions on how to use FileZilla are also included in Module 7.

  Open the Windows Command prompt and type:

  ```
  pscp.exe -i <path-to-ssh-key> <path-to-openrc-file>
  ubuntu@NNN.NNN.NNN.NNN:
  ```

replacing the N's with your IP address. Specify the path where you have saved your nectar ssh key in <path-to-ssh-key> and the path to your *OpenStack RC* file in <path-to-openrc-file>.
This will copy your *OpenStack RC* file to your ubuntu instance's *home* directory.

Step 4: Load your OpenStack credentials from the RC file

Before we can use the command line client, we need to load the OpenStack RC file which contains our authentication details.

- **Linux / Mac OS X:** Load your credentials:

  `source <path-to-your-openrc-file>`

  using the above example for the RC files, this would be:
  `source ~/pt-12345-openrc.sh`

  You will be prompted for your OpenStack password.

  > You can try to copy and paste your password into the command line prompt. If this does not work in your command line, you may edit the OpenStack RC file and assign the password directly by replacing *$OS_PASSWORD_INPUT* by your password, and removing the line above (the *read* command). Then, do the *source* command above again.
  > **Attention**: Be aware that if you do this, your OpenStack password will be openly stored in the OpenStack RC file which everyone who may gain access to your computer can read! You are creating a **potential security leak**, so do this only if you are absolutely sure your computer is well protected and nobody else can read the file from where it is stored. At the very least, it is recommended that you restrict access to this file to your user account only:
  > `chmod 0600 <path-to-your-openrc-file>`

- **Windows**:
  You will need the *Windows PowerShell* installed (available on Windows 7 and later, or on XP with Service Pack 3).

  The OpenStack RC file is a script written for Unix/Mac OSX systems, so it will need to be *modified* to work with the Windows PowerShell. Open your OpenStack RC file in NotePad or WordPad (**not** Word!!) and change it so it looks like the snippet below (with your own values from your OpenStack RC file).

```
$env:OS_AUTH_URL="https://keystone.rc.nectar.org.au:5000/v2.0/"
$env:OS_TENANT_ID="f12d34....c"
$env:OS_TENANT_NAME="<your-tentant-name>"
$env:OS_USERNAME="<your-email>"
$env:OS_PASSWORD="<Your-OpenStack-Password>";
$env:OS_REGION_NAME="<Your-Preset-Region-Name>"
```

Make sure to save this as *a powershell* (*.ps1*) file in *plain text* (pre-select *.txt* if necessary and then change the suffix to *.ps1*). For simplicity, save the file as *openrc.ps1* in your user folder, which usually is *C:\Users\<Your-User-Name>*. Then, open the **Windows Powershell** by typing into the Windows Command line:

`powershell.exe`

And load your credentials:

`C:\Users\<Your-User-Name>\openrc.ps1`

Note: Alternatively, you may use the Windows PowerShell Terminal directly. Search for “powershell” in the App Search.

> Make sure only you can read the *openrc.ps1* file, because the OpenStack password is readable on it. Right-click on the file, select “Properties”, go to the “Security” Tab, click “Edit…” and remove all names but your own user name.

You will have to load your credentials **every time** after you reboot your computer or open up a new terminal, otherwise you cannot use the clients. Use the same command as above to load them afresh.

Step 5: Use your OpenStack client

You are now ready to use your OpenStack client in the command line session in which you loaded your OpenStack RC file in the last step.

Keep this terminal open to follow the instructions in the next sections.

> Whenever you start over (e.g. after you close the terminal), you only need to load (*“source”*) your OpenStack RC file in order to get ready to use the OpenStack client, as described in the last step:
>
> `source <path-to-your-openrc-file>`

You can display help for the *openstack* command with:

`openstack help`

And if you would like to see the more detailed help for a particular sub-tool, e.g. the *server* tool:

```
openstack help server
```

For supporting and further documentation throughout the next sections, please refer to the official openstack client documentation.

Accessing the object store

> *Prerequisite*: You need to have installed the *python-openstackclient* using the instructions given earlier. You should also be familiar with the terminology and concepts involved with the object store, as described in Module 6 and Module 7. You also must have *sourced* your OpenStack RC file on the command line you are using, as described earlier in this module.

> Before the **openstack** client was introduced, access to the object store was managed with the **swift** command, so you may still find references to it on the Internet. While you still can use *swift*, in this tutorial we stick to the more uniform and intuitive *openstack* client, which works similarly to the *swift* command. For more details, refer to the official OpenStack swift documentation.

You can display help for the *openstack* command with:

```
openstack help
```

This will print a list of options for this command. Each option listed can be seen as a ‘sub-command’. You may print more information about usage of the particular sub-command, e.g. for *object* and *container* which we are going to use in this section:

```
openstack help object
openstack help container
```

For supporting and further documentation, please refer to the official openstack client documentation.

In this section, we are going to learn how to

- List your containers and files
- Create and delete Containers

- Upload and download files

> The following instructions will assume that all the *Names* you choose for containers and files have **no whitespaces** in them. If the names *do* include white spaces, you have to put the name in quotation marks when using them in the commands, e.g. “Name of container”.

List your files

Type

`openstack container list`

and you should see the containers which you have already created.

```
+--------------------+
| Name               |
+--------------------+
| MyObjectContainer  |
+--------------------+
```

To display the contents of a container:

`openstack object list <your-container-name>`

will list the file(s) you have just uploaded to this container. If you have followed the tutorial in Module 7, you should see the file which you uploaded via the Dashboard.

```
+------------------+
| Name             |
+------------------+
| SampleObject.txt |
+------------------+
```

Manage containers and transfer files

To upload an object to the store, the file must be on the computer which you use to type the commands in the terminal.

As an example, create a text file named *MyNewTextFile.txt*, write some text in it, and save it in your home directory (or on Windows, place it in *D:\Users\<YourUserName>*). First, create the new container:

`openstack container create MyTestContainer`

You can now see your new container is listed with

```
openstack container list
```

```
+-------------------+
| Name              |
+-------------------+
| MyObjectContainer |
| MyTestContainer   |
+-------------------+
```

You may **upload a file** with the command:

```
openstack object create <container name> <path to local file>
```

In the example, upload the file you have created to the new container *MyTestContainer*:

```
openstack object create MyTestContainer ~/MyNewTextFile.txt
```

Note: On **Windows**, you have to use the Windows command line syntax to specify the path to the file. This tutorial assumes you are familiar with how to modify the commands to use the conventions of the Windows command line.

This will have created a new container called *MyTestContainer* (unless you had a container with this name already). List the files which are there now:

```
openstack object list MyTestContainer
```

```
+----------------------------------+
| Name                             |
+----------------------------------+
| /home/yourname/MyNewTextFile.txt |
+----------------------------------+
```

Note that the path to your filename has been included in the name, which is not very convenient. Unfortunately, the only way around this is to upload a file which is in the *current working directory*. So let's delete the file, and then upload it without the path:

```
openstack object delete MyTestContainer
/home/yourname/MyNewTextFile.txt
cd ~
openstack object create MyTestContainer MyNewTextFile.txt
```

And list the files again to see that the file name stands alone now:

```
openstack object list MyTestContainer
```

```
+----------------------------------+
| Name                             |
+----------------------------------+
| MyNewTextFile.txt                |
+----------------------------------+
```

You may display information about your file with:

`openstack object show MyTestContainer MyNewTextFile.txt`

This will display properties and metadata of your file, e.g. *content-length* displays the length in bytes.

```
+---------------+--------------------------+
| Field         | Value                    |
+---------------+--------------------------+
| accept-ranges | bytes                    |
| account       | None                     |
| container     | MyTestContainer          |
| content-length | 112                     |
| content-type  | text/plain               |
| etag          | d41d8cd98f00b204e98...   |
| last-modified | Thu, 02 Jul 201...       |
| object        | MyNewTextFile.txt        |
| x-timestamp   | 1435842936.81309         |
| x-trans-id    | tx42ec77c1403b47c...     |
+---------------+--------------------------+
```

Now, let's **download** that same file again and save it as another name, to see that it still has the right content:

`openstack object save --file <destination path> <container name> <file name>`

The argument `--file` specifies the destination path on your computer where you want to save the file. If you want to keep the name of the file, this path can be the destination folder only. If you want to rename the file at the same time, specify a file name also. In our example, if we want to save the file as *MyDownloadedFile.txt* into our home directory, the command becomes:

`openstack object save --file ~/MyDownloadedFile.txt MyTestContainer MyNewTextFile.txt`

You may now open the file *MyDownloadedFile.txt* in your home directory and see that it has the same text it contained originally.

Finally, let's delete the container which we used for this example:

```
openstack container delete MyTestContainer
```

This will **delete all the files** which you have uploaded to this container!

Access to the Object Store from APIs

It is also possible to access your files in the Object Store from within program code. For example, there are APIs for C++ and Java.

Access via APIs is not part of this tutorial. Please refer to the NeCTAR support website and the official OpenStack SDKs website for more information.

Controlling an instance

Prerequisite: You need to have installed the *python-openstackclient* using the instructions given earlier. You should also be familiar with the terminology and concepts involved in launching an instance, as described in Module 7. You also must have *sourced* your OpenStack RC file on the command line you are using, as described earlier in this module.

Before the **openstack** client was introduced, managing compute services was done with the **nova** command, so you may still find references to it on the Internet. While you still can use *nova*, in this tutorial we stick to the more uniform and intuitive *openstack* client, which works similarly to the *nova* command. For more details, refer to the official OpenStack nova documentation.

The following instructions will assume that all the *Names* you choose for instances, volumes, etc. have **no whitespaces** in them. If the names *do* include white spaces, you have to put the name in quotation marks when using them in the commands, e.g. “Name of resource”.

You may display information about all the instances you are running:

```
openstack server list
```

This will output a line like this one for each running instance:

```
+--------------+----------------------+--------+---------------------------+
| ID           | Name                 | Status | Networks                  |
+--------------+----------------------+--------+---------------------------+
| be290858-... | MyFirstInstance      | ACTIVE | intersect-01=43.240.96.180 |
+--------------+----------------------+--------+---------------------------+
```

You can display more information about your instances with:

`openstack server show <instance ID>`

Where the instance ID is the long combination of numbers, dashes and letters shown in the *openstack server list* command. Alternatively, you may use the *name* of the instance, e.g.:

`openstack server show MyFirstInstance`

in the example above. This will display a lot of information about this particular instance.

You can display help for the *openstack* command with:

`openstack help`

This will print a list of options for this command. Each option listed can be seen as a ‘sub-command’. You may print more information about usage of the particular sub-command, e.g. for *server* which we are going to use in this section:

`openstack help server`

In the following, we are going to discuss *openstack server* commands for

- Launching an instance
- Taking a snapshot of an instance
- Launching a new instance from a snapshot

For supporting and further documentation, please refer to the official openstack client documentation.

Launching an instance

You can also launch an instance with *openstack server* instead of using the Dashboard as we have done in Module 7.

First, you may want to take a look at the flavors which are available and pick the one you'd like:

```
openstack flavor list
```

In this tutorial, let's pick the flavor *m1.small*. Take note of the ID (first column of the printed information) and name. When compiling this tutorial, the ID for the flavor *m1.small* was *0*, but in your case it may also be a long string of characters.

Now, you should pick an image to launch your instance from. Just as in Module 7, we will use the Ubuntu 14.10 image.

```
openstack image list
```

Because there are a lot of images, the list will be very long. To make it easier to find the right Ubuntu 14.10 image in the NeCTAR images, type:

```
openstack image list | grep 14.10
```

Find the image *NeCTAR Ubuntu 14.10 (Utopic) amd64" and take note of the ID, which may be a number or a long string of characters. When compiling this tutorial, the ID was *fc48b5bb-e67d-4e39-b9ba-b6725c8b0c88*.

You now have to configure the ssh keys you want to use. Your *Nectar_Key* should show up in the following command (with another name if you chose another launching the instance in Module 7):

```
openstack keypair list
```

We will use the same key in this tutorial, so take note that the **Keypair Name** (first column) is the one you chose for the key, i.e. *Nectar_Key*.

You may also add a new key. The command

```
openstack keypair create <name>
```

will generate a key with the <name> of your choice, and print the private key to the terminal. You have to copy and save this output **now** to a file and store it somewhere as your private key.

If you want to select an existing key that you generated earlier, you have to specify the path to your *public* key, which you must be able to access from your terminal:

```
openstack keypair create --public-key <path-to-your-pub-key>
```

`<name>`

You can display the help for *keypair create* with

`openstack help keypair create`

You may display information about a specific key with:

`openstack keypair show <keypair-name>`

using the keypair name for which you want to display the information.

Let's now display information about the security groups:

`openstack security group list`

This should display something like this:

```
+-------+---------+-------------+
|    ID | Name    | Description |
+-------+---------+-------------+
| 86498 | default | default     |
| 90284 | http    | Allow http  |
| 86501 | icmp    | Allow icmp  |
| 86504 | ssh     | Allow ssh   |
+-------+---------+-------------+
```

Take note of the security group *names* (2nd column) for *icmp* and *ssh*.

The last thing we may need to specify before we launch an instance is the *Availability Zone*. List all the available zones with:

`openstack availability zone list`

Take note of your *Zone Name* (first column), if you want to use a particular zone. Otherwise, you won't need to specify the availability zone and you will be assigned the most suitable one automatically.

Now, you may launch your instance with the command *openstack server create*. Display help about this command:

`openstack help server create`

which will describe the options available. Launch your instance:

`openstack server create --flavor <flavor-id> --key-name <keypair-name> --image <image-id> --security-group <group name> --`

`availability-zone <zone name> <name-of-instance>`

As a name for the instance, choose *ClientLaunchedInstance*. For the flavor, you may specify either the *name* or the *id* of that flavor. If you assign several security groups, you must repeat the *–security-group* argument before each group name. If you don't want to use a specific availability zone, you may skip the *–availability-zone* argument. At the time this tutorial was created, the command (without availability zone) was:

`openstack server create --flavor m1.small --key-name Nectar_Key --security-group icmp --security-group ssh --image fc48b5bb-e67d-4e39-b9ba-b6725c8b0c88 ClientLaunchedInstance`

It will display information about the instance being created, similar to these *extracts* of output:

```
+------------------------------+------------------------------------------+
| Field                        | Value                                    |
+------------------------------+------------------------------------------+
| OS-DCF:diskConfig            | MANUAL                                   |
| OS-EXT-AZ:availability_zone  | intersect                                |
| OS-EXT-STS:power_state       | 0                                        |
| OS-EXT-STS:task_state        | scheduling                               |
| OS-EXT-STS:vm_state          | building                                 |
| flavor                       | m1.small (0)                             |
| hostId                       |                                          |
| id                           | b720630a-e0dd-4ecd-9ceb-5e3519e69edc     |
| image                        | NeCTAR Ubuntu 14.10 (Utopic) amd64 (fc4..) |
| key_name                     | Nectar_Key                               |
| name                         | ClientLaunchedInstance                     |
| security_groups              | [{u'name': u'icmp'}, {u'name': u'ssh'}]  |
| status                       | BUILD                                    |
+------------------------------+------------------------------------------+
```

The state (*vm_state*) will probably still be *"building"*. Once the instance has been created, it will show up in the output of *openstack server list* and *openstack server show*.

You can check if the status of the instance is **Active** with the command:

`openstack server list`

Once it is active, you may look up the IP address for *ClientLaunchedInstance* with

`openstack server show ClientLaunchedInstance`

Congratulations!! You have now launched your instance.

You may now try to login to your instance with the SSH command, as described in

Module 7. Use the IP address which is displayed for your new instance to login via ssh. For example, on a Linux or Mac:

```
ssh -i .ssh/Nectar_Key ubuntu@NNN.NNN.NNN.NNN
```

replacing the N's with the IP address.

For more information, see also the official OpenStack Client documentation.

You can now go to the Dashboard and observe that your new instance will be listed there as well.

You may reboot the instance with

```
openstack server reboot <instance-name>
```

and *terminate* (delete) it with

```
openstack server delete <instance-name>
```

If you want to add a new security group, you can easily do this after launching as well:

```
openstack server add security group <group name>
```

To get more information about the *openstack server* command:

```
openstack help server
```

Taking a snapshot of an instance

Taking a snapshot of the instance is easy. Using your *Instance Name*:

```
openstack server image create --name <Image-Name> <Instance-Name>
```

The snapshot will be **saved as an Image** with the name you choose. In the example given in this tutorial, the command is:

```
openstack server image create --name ClientLaunchedSnapshot
ClientLaunchedInstance
```

This process may take a while. In the beginning, the 'status' of your snapshot will be *queued*.

You can check on the status of your snapshot with:

```
openstack image show <Image-Name>
```

> It is possible to take a **"Live snapshot"** (take a snapshot of the machine that is

currently running). In most cases, there should be no problem. However, if the instance is running while the snapshot is being taken, the resulting snapshot *may* be inconsistent. This is due to programs writing on the file system while a snapshot is taken. There are a few options to prevent this inconsistency from happening:

- running “sync” before starting the snapshot, or
- using a “file system freeze” utility which block programs writing on the filesystem,
- shutting down or pausing the instance before snapshotting.

The easiest option is probably to pause or shut down the instance and then take the snapshot. For more information, refer to the OpenStack documentation.

Launch an instance from a snapshot

Because your snapshot was saved as an *Image*, you can easily launch the instance from this image. Instead of the *NeCTAR Ubuntu Image* which we used earlier, you can now specify your own image to launch a new instance. The new instance will have all the software you installed before taking the snapshot, and all the data you saved the on-instance storage (on the primary disk, *not* the secondary!).

You can print a list of all the images (instance and volume snapshots) you have created, if you would like a reminder for the snapshot name:

```
openstack image list --private
```

And more details about the images can be displayed with

```
openstack image show <Image Name>
```

You may launch an instance from your own image with the same command we used earlier to launch an instance:

```
openstack server create --flavor <flavor-id> --key-name <keypair-name> --image <snapshot-name> --security-group <group name> --availability-zone <zone name> <name-of-instance>
```

In our particular example, the command becomes:

```
openstack server create --flavor m1.small --key-name Nectar_Key --security-group icmp --security-group ssh --image ClientLaunchedSnapshot CopyOfClientLaunchedInstance
```

List the details of your new instance with

```
openstack server show CopyOfClientLaunchedInstance
```

You have now two copies of your original instance running. You may log in with ssh to *CopyOfClientLaunchedInstance* as well.

> Note that you may also choose a **different flavor** when you launch a new instance from the snapshot! This allows you to expand your resources (though you can do this more easily by following the process outlined in *openstack help server resize*). You should not choose a flavor with less storage available on the primary on-instance disk, otherwise the launching of your instance may fail due to lack of disk space.

Managing volumes

> *Prerequisite*: You need to have installed the *python-openstackclient* using the instructions given earlier. You should also be familiar with the terminology and concepts involved for managing volumes, as described in Module 6 and Module 7. You also must have *sourced* your OpenStack RC file on the command line you are using, as described earlier in this module.

> Before the **openstack** client was introduced, managing volumes was done with the **cinder** and **nova** commands, so you may still find references to it on the Internet. While you still can use *cinder* and *nova*, in this tutorial we stick to the more uniform and intuitive *openstack* client, which works similarly to the *cinder* and *nova* commands. For more details, refer to the official OpenStack cinder and nova documentation.

You can display help for the *openstack* command with:

```
openstack help
```

This will print a list of options for this command. Each option listed can be seen as a ‘sub-command’. You may print more information about usage of the particular sub-command, e.g. for *volume* which we are going to use in this section:

```
openstack help volume
```

For supporting and further documentation, please refer to the official openstack client documentation.

In this section we are going to deal with managing *Volumes*:

- Creating and deleting volumes
- Attaching and detaching volumes from instances
- Taking snapshots and backups of volumes, and
- Restoring volumes from snapshots or backups

> You cannot use Volumes if you are using the NeCTAR Trial Account. You need to submit an allocation request to get access to volume storage. You may read more about allocations in Module 5.

> The following instructions will assume that all the *Names* you choose for instances, volumes, etc. have **no whitespaces** in them. If the names *do* include white spaces, you have to put the name in quotation marks when using them in the commands, e.g. “Name of resource”.

Create a new volume

You may list the volumes you already have with:

```
openstack volume list
```

This will print information about your existing volumes (or print nothing if you don’t have any volumes yet):

```
+------------------+----------+--------------+------+-------------+----------+---
|  ID              |  Status  | Display Name | Size | Volume Type | Bootable |  A
+------------------+----------+--------------+------+-------------+----------+---
| 494e6104-a282-...|  in-use  | MyVolume     |  1   |  intersect  |  false   | a8
+------------------+----------+--------------+------+-------------+----------+---
```

Creating a new volume is easy using the *openstack* client. You can display help for volume creation with:

```
openstack help volume create
```

to see the syntax of the command and some optional arguments.

Now, create a new Volume called *MyNewStorage* sized *1GB* with the following command (specifying your own availability zone):

```
openstack volume create --description "Description of the volume" --availability-zone <zone name> --size 1 MyNewStorage
```

> You will need to specify the *same* availability zone in which the instance you want to attach it to is running, or you will not be able to attach the volume to the instance.
>
> To list all availability zones:
> `openstack availability zone list`

A summary will be printed which shows the details of the volume you are creating (extracts shown in output):

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | intersect                            |
| bootable            | false                                |
| display_description | Description of the Volumme           |
| display_name        | MyNewStorage                         |
| encrypted           | False                                |
| id                  | 42385945-e2...                       |
| properties          |                                      |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | intersect                            |
+---------------------+--------------------------------------+
```

The *status* of the volume should switch to *'Available'* soon. Re-check the status with

`openstack volume show MyNewStorage`

After the volume has been created, when you type

`openstack volume list`

you should see your new volume in the list.

If you want to *delete* a volume again, use this command:

```
openstack volume delete <volume name or ID>
```

> When you delete a volume, you will **lose all data and access** to this volume. Make sure you **back up** all data, and **securely erase** the data on the volume (as described in Module 9) *before* you delete it!

Attaching/Detaching a volume to an instance

To attach and detach a volume to an instance, you will use the command tool *openstack server*. You can display help about this command by running the following:

```
openstack server help | grep volume
```

First, find the instance name that you want to attach the volume to by listing all your instances:

```
openstack server list
```

```
+--------------------+----------------------------+--------+----------------------
| ID                 | Name                       | Status | Networks
+--------------------+----------------------------+--------+----------------------
| af87a5c8-e751-...  | CopyOfClientLaunchedInstance | ACTIVE | intersect-02=137.9
| b720630a-e0dd-...  | ClientLaunchedInstance       | ACTIVE | intersect-02=137.9
+--------------------+----------------------------+--------+----------------------
```

To attach a volume, you may use this command:

```
openstack server add volume <Server-Name> <Volume-Name>
```

Server-Name: the name (or ID) of instance which you want to attach the volume to. Let's attach it to the instance we launched in the last section, which is named *ClientLaunchedInstance*.

Volume-Name: The name (or ID) of the volume you want to attach. We will attach the volume *MyNewStorage* which we just created.

with the examples used in this tutorial the command becomes:

```
openstack server add volume ClientLaunchedInstance MyNewStorage
```

> The `openstack server add volume` command also offers an argument `--device`. This *optional* argument is intended to set the device mapping, e.g. */dev/mydevice*. OpenStack **currently ignores** this option when attaching volumes, so you should skip this option at this stage (it may be that

your command does not work if you specify this option).

The command will take a while to execute. When it is finished, when you run

`openstack volume list`

the output should show that your volume is attached:

```
+--------------+--------------+--------+------+---------------------------------
| ID           | Display Name | Status | Size | Attached to
+--------------+--------------+--------+------+---------------------------------
| 42385945-... | MyNewStorage | in-use |    1 | Attached to ClientLaunchedInstan
+--------------+--------------+--------+------+---------------------------------
```

OpenStack adds the volume as the lowest available device name: For a standard flavor instance (with a primary & secondary drive) the first volume will be attached as */dev/vdc*. You can see where your device has been mapped to in the last column of the output.

> You may attach *several* volumes to one instance. However you may only attach one volume to *one* instance at a time.

To **detach** a volume, use a similar command:

`openstack server remove volume <Server-Name> <Volume-Name>`

> Before you detach a volume, you have to **unmount** it! Module 7 includes information about mounting and unmounting drives.

Now, detach the volume again that we have just attached:
`openstack server remove volume ClientLaunchedInstance MyNewStorage`

After detaching, if you run

`openstack volume list`

you will see that the status of the volume is no longer *“in-use”*, instead it is *“available”*.

Backups and Snapshots

You may create a *Backup* of a volume, or take a *Snapshot* of it. The difference

between Backup and Snapshot has been explained in Module 9: The key difference is that a Snapshot creates an *Image* of the size your Volume has, whereas a Backup creates a *backup file* which has the size of your *used* storage, and saves it in the *Object Store*. Another important difference is that Backups can be used to restore the state onto an *existing* volume, whereas with a Snapshot, it is only possible to create an entirely *new* volume from it.

1. Backup and restore a volume

To manage your volume Backups, you will use the command set *openstack backup*:

```
openstack help backup
openstack help backup create
```

> Backups will back up the entire state of your Volume. If you volume is encrypted, the encryption will be backed up with it. So after you restore this backup onto a new volume, you will also need to install the volume encryption and activate it with your password!

Before you can create a Backup, you must *detach* the volume from any instances. You need to first *unmount* the device (if you have mounted it from an instance), and then detach it, as described earlier. A volume is *detached* when it is in the status *"available"*. Check this with

```
openstack volume list
```

You will see *"Backup1"* listed along with its current status.

To create a backup of the volume named *MyNewStorage*, and save it into the object store container *"Backups"* naming it *"Backup1"* use the following command:

```
openstack backup create --container Backups --name Backup1 --
description "Backup MyNewStorage" MyNewStorage
```

While the argument `--description` is optional, it is recommended you fill in some meaningful information about this backup here, as it will make it easier to identify it when you want to restore it.

If the container *"Backups"* did not already exist, it will have been created. After creating the backup, your volume will be have the status *"backing-up"*, which can take a while. Type

```
openstack backup list
```

to see the status of your Volume backup.

```
+--------------------+---------+---------------------+-----------+------+
| ID                 | Name    | Description         | Status    | Size |
+--------------------+---------+---------------------+-----------+------+
| eb4d23e7-861e-4... | Backup1 | Backup MyNewStorage | available |    1 |
+--------------------+---------+---------------------+-----------+------+
```

After the backup has been done, you can see it listed in your Object Store:

`openstack container list`
`openstack object list Backups`

The Backups container has been created (unless you already had it). You will see a rather long list of files in the *Backups* container — which means your Backup has been saved in several object files. This should not concern you, however, as you can restore the data with only one command.

To restore your Volume to the state we just saved in *Backup1*, first obtain the *ID* of this backup, which is displayed in the first column of the output:

`openstack backup list`

You will need the *ID* (in this case, the *Name* does not work) to display some more information about this Backup:

`openstack backup show <Backup-ID>`

```
+-------------------+-----------------------+
| Field             | Value                 |
+-------------------+-----------------------+
| availability_zone | intersect             |
| container         | Backups               |
| created_at        | 2015-07-...           |
| description       | Backup MyNewStorage   |
| fail_reason       | None                  |
| id                | eb4d23e7-8 ...        |
| name              | Backup1               |
| object_count      | 22                    |
| size              | 1                     |
| status            | available             |
| volume_id         | 42385945-e2d2...      |
+-------------------+-----------------------+
```

Note that the Volume ID (field *volume_id*) is displayed in the information to remind us which volume the backup is from. The ID is not very intuitive, so it is a good idea to add a meaningful description to the backup when you create it.

To restore a backup, first display the help for the *backup restore* tool:

```
openstack help backup restore
```

To restore the volume *MyNewStorage* to the state we saved in *Backup1*, use the command:

```
openstack backup restore <Backup-ID> MyNewStorage
```

> You may also restore the backup to a volume other than the Volume you took the Backup of — the Backup is **not bound to the original Volume** you took the Backup of!
>
> When you restore to **any Volume**, you must keep the following in mind:
>
> - all data on the Volume you choose as the restore target **will be lost**! The state of the Volume after the restore is going to be **exactly the state** which your Volume was in when you made the Backup.
> - the Volume needs to be large enough to fit your backup data.
> - the Volume you restore the backup onto will get the name of the originally backed up Volume. For example if you did a Backup of *MyNewStorage*, then restored this backup onto the Volume *MySecondStorage*, the volume *MySecondStorage* will be renamed to *MyNewStorage* (however the original *ID* of the Volume will be kept).

Backups take space in your object store, so you may want to delete old Snapshots every now and then with:

```
openstack backup delete <Backup-ID>
```

2. Snapshot a volume and re-create it

Before taking a snapshot, you should make sure that your volume is detached from any instance: It has to be in in “*available*” status. You can check this with

```
openstack volume list
```

To create a Snapshot of a volume:

```
openstack snapshot create --name <Snapshot-Name> --description "Describe the snapshot" <Volume-Name>
```

As the volume name you may specify the name or ID. You should name and describe the snapshot such that it makes sense to you. Let's for example create a snapshot of *MyNewStorage* and name it *MyNewStorageSnapshot1*:

```
openstack snapshot create --name MyNewStorageSnapshot1 --description "First snapshot" MyNewStorage
```

Now, you should be able to see your Snapshot in the list:

```
openstack snapshot list
```

Take note of the *ID* of your snapshot, you will need it to restore the snapshot state.

> When you want to restore the state of the snapshot, you need to create a *new* volume — it is not possible to restore the state onto an existing volume, as it can be done with a Backup.

You can create a *new* volume of this snapshot using the same command as described earlier to create a volume, but adding the `--snapshot` argument to specify the ID (the *Name* will not work) of your snapshot.

```
openstack volume create --snapshot <snapshot-ID> --description "My restored Volume" --size <Size-in-GB> [--availability-zone <zone>] <New-Volume-Name>
```

You may create a Volume which has a larger storage size than the original Snapshot, but you cannot create one with less.

For example, we can create a new volume out of the snapshot *MyNewStorageSnapshot1* and call it *MyRestoredVolume*, and set the size to 2GB as follows:

```
openstack volume create --snapshot <ID of MyNewStorageSnapshot1> --description "My restored Volume" --size 2 MyRestoredVolume
```

> The original Volume on which the Snapshot was based *must still exist*, or the Snapshots of it become useless.
>
> OpenStack does not let you delete Volumes which have "depending Snapshots", so losing Snapshots by accidentally deleting a Volume is not a worry. However you have to be aware that the Snapshots are only usable while you keep your Volume in existence. This is different to *Backups*.

The new volume will now be listed with your other volumes:

```
openstack volume list
```

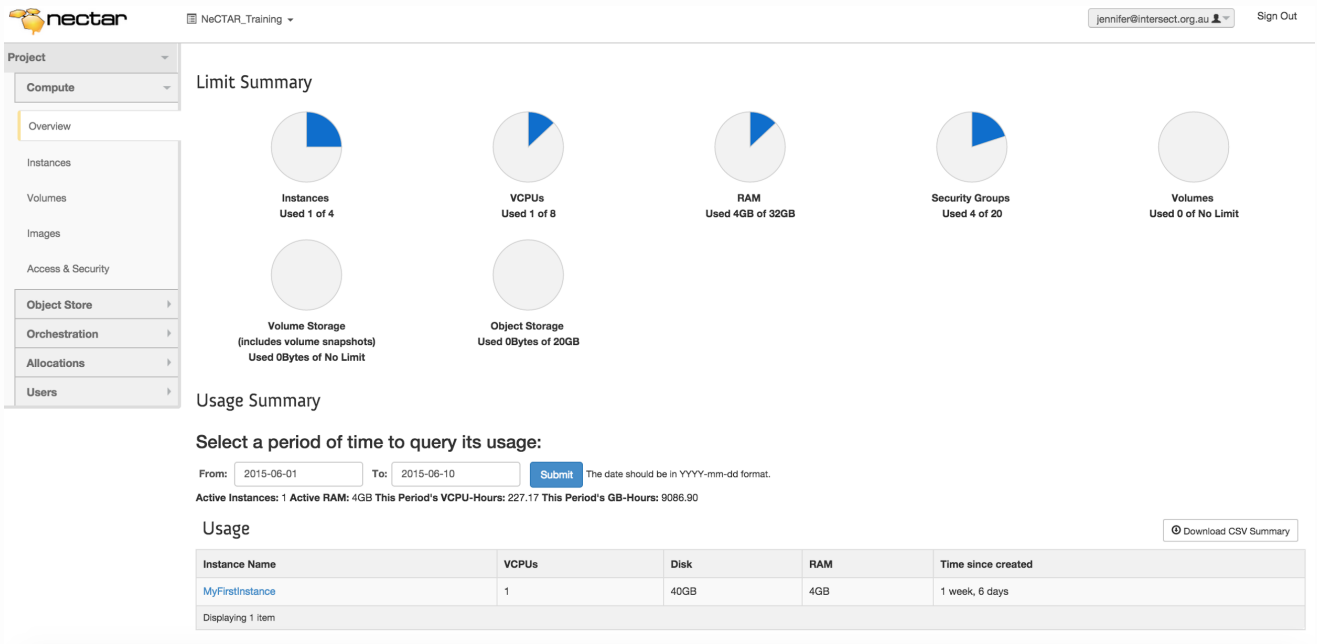
```
+----------------------+------------------+-----------+------+-------------+
| ID                   | Display Name     | Status    | Size | Attached to |
+----------------------+------------------+-----------+------+-------------+
| 9f8af2bd-d5d4-45d4-... | MyRestoredVolume | available |    2 |             |
| 80cec22c-8c0c-4bd4-... | MyNewStorage     | available |    1 |             |
+----------------------+------------------+-----------+------+-------------+
```

Snapshots take a good amount of your storage capacity, because they have the same size as the volume that was snapshotted. So you may want to delete old Snapshots every now and then:

```
openstack snapshot delete <Snapshot Name>
```

Tracking utilization

On the Dashboard, you can see the amount of resources you are using. You can also see how long your instances have been up and running.



You may want to *monitor* your virtual machine(s), i.e. keep track of CPU usage, memory usage, etc. You may also want to set up system alerts. The Dashboard does not provide any tools to do this; you will have to use tools for virtual machine *monitoring*. Because the NeCTAR cloud uses OpenStack to manage the cloud, we require a monitoring tool which is compatible with OpenStack.

OpenStack has a tool called *Ceilometer* which collects measurements within

OpenStack. *Ceilometer* is actively developed and is now an integrated part of OpenStack. The Ceilometer project is a framework for **monitoring and metering** the OpenStack cloud.

Ceilometer is a service offered by OpenStack which you can query via the command line. It collects and monitors *performance* and *usage* data:

- Compute resources (e.g. CPU, Memory use, number of instances used)
- Network activity, including data transfer (in/out) by location.
- Storage (including object storage and volumes).

Ceilometer commands require installation of the client package. Given you have already used the *openstack* command in the previous sections, all you need to do is

```
pip install python-ceilometerclient
```

or if you haven't used *pip* but your Linux package manager, install the client with your package manager as well, e.g. on Debian/Ubuntu:

```
sudo apt-get install python-ceilometerclient
```

You will also need to *source* your OpenStack RC file as described earlier.

If you haven't installed any OpenStack command line clients, follow instructions described earlier and replace *openstackclient* with *ceilometerclient*.

Detailed instructions on how to use *ceilometer* is out of scope of this tutorial. You can read up on the ceilometer command line reference in the official OpenStack documentation and command line reference. Good documentation and useful links are also given on the NeCTAR support website and in this good introductory article.

Summary

Congratulations!!

You have completed the last module of this course. You now know how to fully control your NeCTAR resources via the command line. In particular, you have learned how to:

- Install OpenStack command line clients
- Launch and terminate an instance
- Take snapshots of instances and relaunch them from snapshots
- Create and delete volumes
- Make backups and take snapshots of volumes

- Access the object store
- Track utilization

We hope you have enjoyed this course. You are fully prepared to get started with using the NeCTAR Research Cloud for your research!

Don’t hesitate to contact NeCTAR support if you have any further questions on how to set up and use the NeCTAR services.