Lab 2: Persistent Storage: Swift and Cinder

In this lab you will gain familiarity in accessing persistent storage in OpenStack through both Horizon and the cli. The projects you will explore are the two core OpenStack Persistent Storage components: Object Storage (Swift) and Block Storage (Cinder).

Section 1: Swift - OpenStack Object Store

In this section you will investigate the principal interaction models with Swift. The process of leveraging the programmatic interfaces directly will be explored in lab 3. You will look at the creation and management of objects in the Swift environment, and discover some of the differences between the Horizon UI and the Swift CLI.

Exercise 1: Swift "Object" panel in Horizon

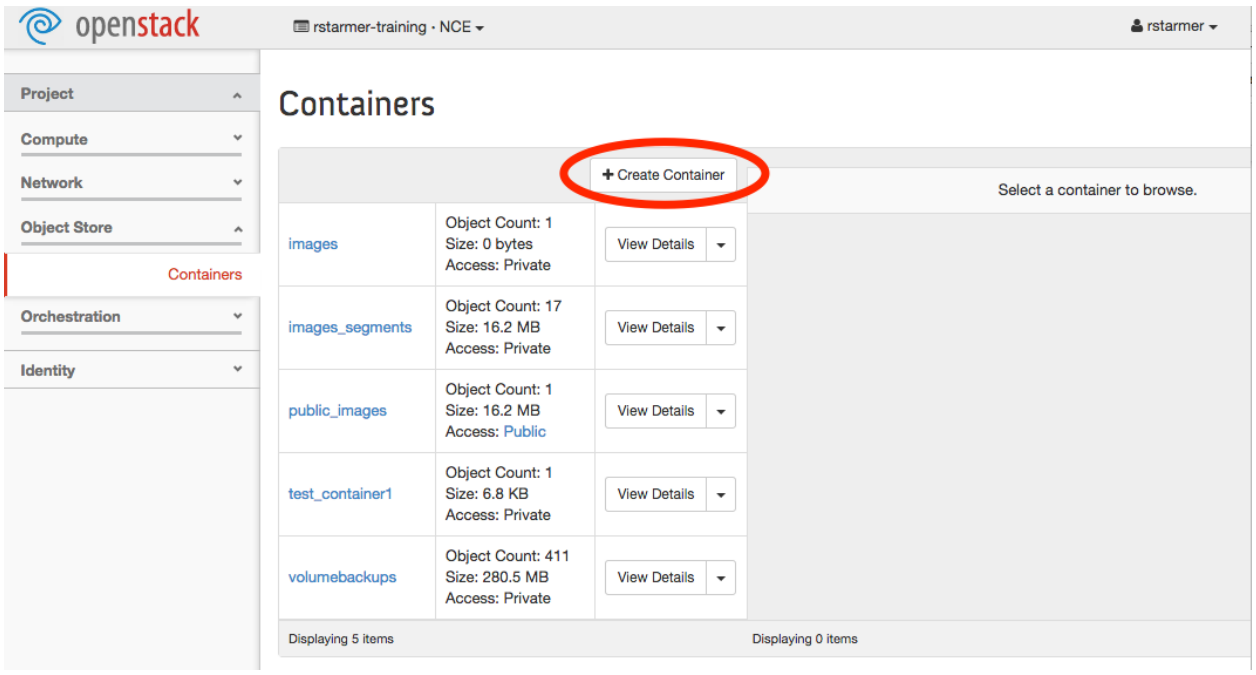
When configured as part of an OpenStack environment, Swift uses the same credentials as the rest of the system, so logging in via Horizon to access Swift is no different than when accessing any other Horizon-enabled OpenStack component.

One difference between a standalone swift environment and Swift with Keystone is that the "account" in Swift is actually tied directly to the tenant. In addition, while most Horizon interfaces are tied to one region or another, in a multi-node swift environment, it doesn't matter what region you connect with, as the replication model in Swift will allow the same operations, and the same data to exist in multiple regions.

Follow along and explore the Horizon integration with Swift. Can you identify the elements of Swift that are not directly available through Horizon?

Container Creation Page:

After successful login, a user must first create a container before uploading objects. In the CLI section we'll see that you can in fact create containers at the same time as a data upload request.



Containers

+ Create Container

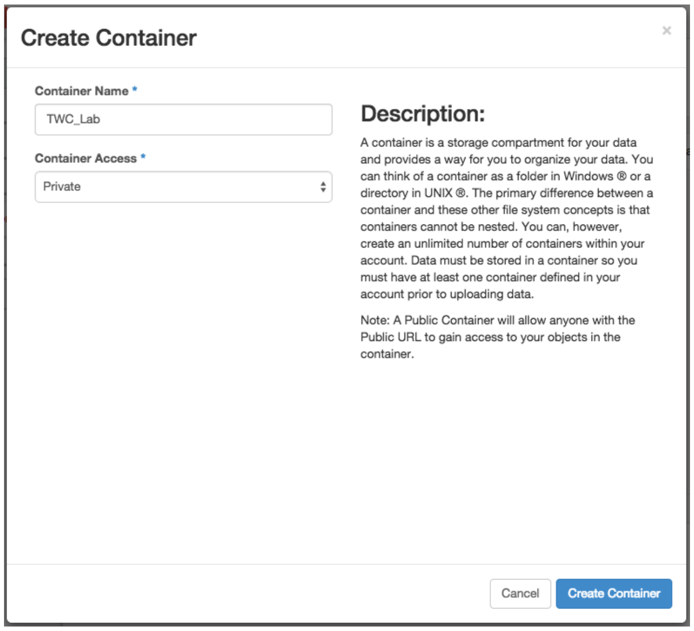
Select a container to browse.

images	Object Count: 1 Size: 0 bytes Access: Private	View Details
images_segments	Object Count: 17 Size: 16.2 MB Access: Private	View Details
public_images	Object Count: 1 Size: 16.2 MB Access: Public	View Details
test_container1	Object Count: 1 Size: 6.8 KB Access: Private	View Details
volumebackups	Object Count: 411 Size: 280.5 MB Access: Private	View Details

Displaying 5 items

Displaying 0 items

After navigating to the Project->Object Store->Containers section of Horizon, users can create a new container or modify an old one through the ‘View Details’ dropdown. On creation, you can select either Private or Public type of Container, and you can change between those states later. Public containers are available to any user without additional authentication restrictions, but the actual public path (not the originating tenants account and created container) needs to be released to those who would need access.



Create Container

Container Name *: TWC_Lab

Container Access *: Private

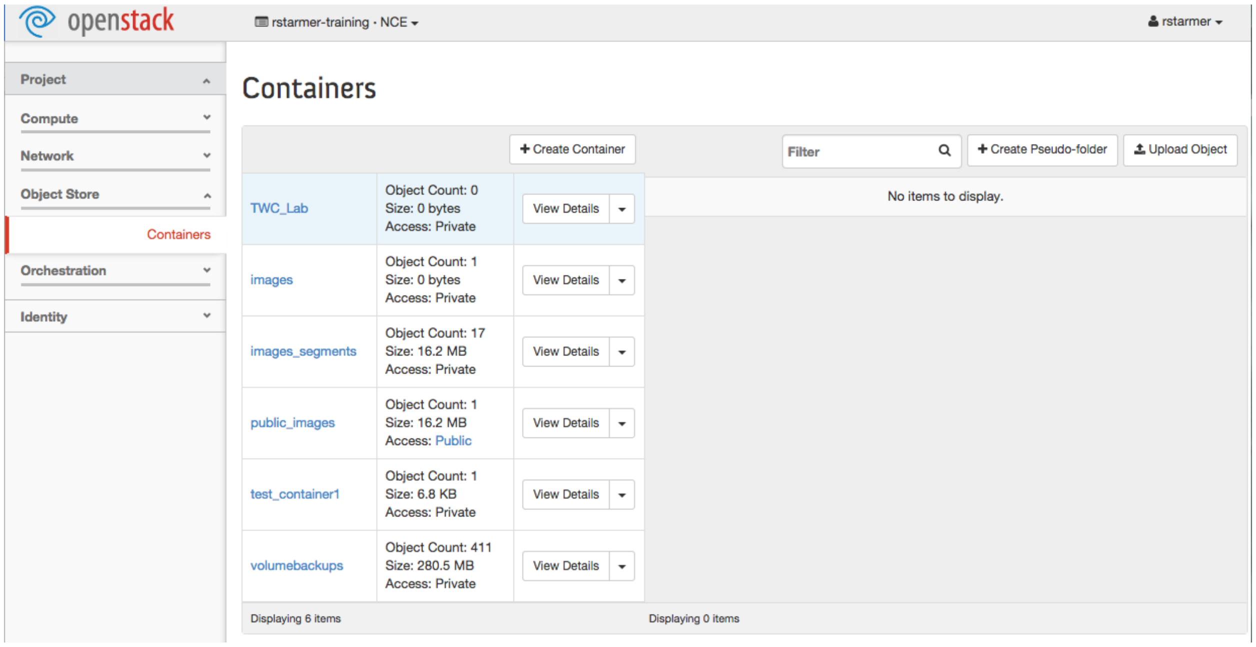
Description:

A container is a storage compartment for your data and provides a way for you to organize your data. You can think of a container as a folder in Windows ® or a directory in UNIX ®. The primary difference between a container and these other file system concepts is that containers cannot be nested. You can, however, create an unlimited number of containers within your account. Data must be stored in a container so you must have at least one container defined in your account prior to uploading data.

Note: A Public Container will allow anyone with the Public URL to gain access to your objects in the container.

List all Containers:

Once the container is created, the user can see the updated list of containers. Modifications can be made via the “view details” sub-menu. Take a look at the modifications you are able to make through Horizon. Note that in this view the container(s) are present, but none have been selected, so there are no Objects being displayed.



Containers

TWC_Lab	Object Count: 0 Size: 0 bytes Access: Private	View Details
images	Object Count: 1 Size: 0 bytes Access: Private	View Details
images_segments	Object Count: 17 Size: 16.2 MB Access: Private	View Details
public_images	Object Count: 1 Size: 16.2 MB Access: Public	View Details
test_container1	Object Count: 1 Size: 6.8 KB Access: Private	View Details
volumebackups	Object Count: 411 Size: 280.5 MB Access: Private	View Details

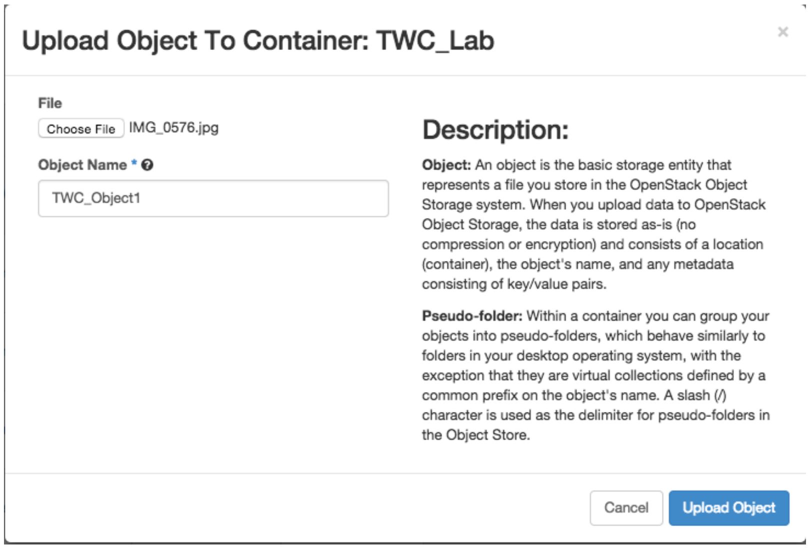
Displaying 6 items

No items to display.

Displaying 0 items

Create Objects:

On uploading an object, note that the Object Name and filename do not need to match, and often they don’t.



Upload Object To Container: TWC_Lab

File: IMG_0576.jpg

Object Name *: TWC_Object1

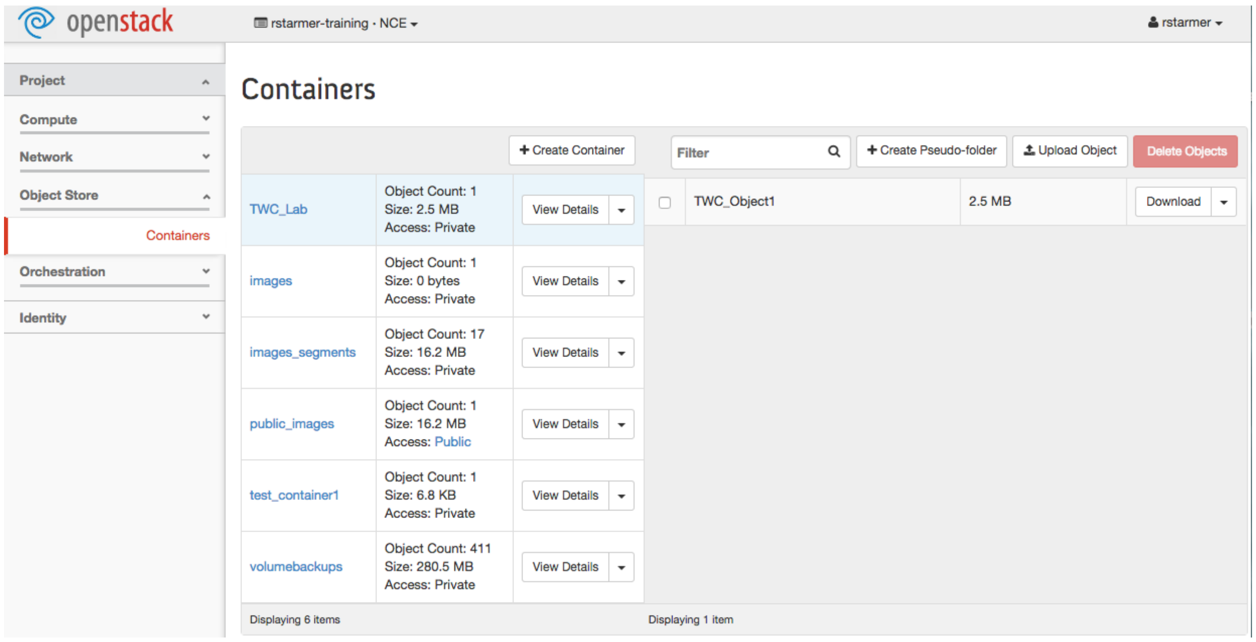
Description:

Object: An object is the basic storage entity that represents a file you store in the OpenStack Object Storage system. When you upload data to OpenStack Object Storage, the data is stored as-is (no compression or encryption) and consists of a location (container), the object's name, and any metadata consisting of key/value pairs.

Pseudo-folder: Within a container you can group your objects into pseudo-folders, which behave similarly to folders in your desktop operating system, with the exception that they are virtual collections defined by a common prefix on the object's name. A slash (/) character is used as the delimiter for pseudo-folders in the Object Store.

List all Objects in a Container:

Now that an object has been uploaded, you can select a specific container to get the current list of object(s) in the panel on the right of the Swift dashboard.



Containers

Container	Details
TWC_Lab	Object Count: 1 Size: 2.5 MB Access: Private
images	Object Count: 1 Size: 0 bytes Access: Private
images_segments	Object Count: 17 Size: 16.2 MB Access: Private
public_images	Object Count: 1 Size: 16.2 MB Access: Public
test_container1	Object Count: 1 Size: 6.8 KB Access: Private
volumebackups	Object Count: 411 Size: 280.5 MB Access: Private

Displaying 6 items

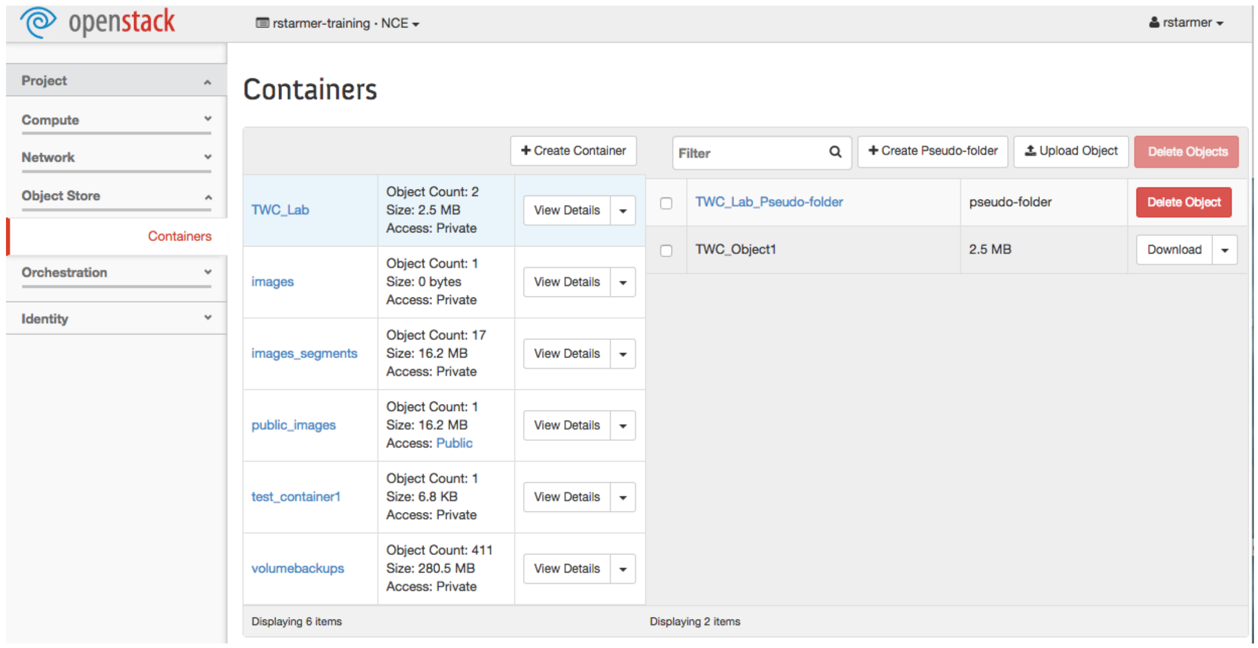
Object	Size
TWC_Object1	2.5 MB

Displaying 1 item

Create folder (pseudo-folder) inside Container:

It is possible to provide further “hierarchy”, though logical, to objects imported. This is especially useful if there is a high likelihood of name collision for content you would want to store in a single container.

Creation of pseudo-folders simply requires you to provide a name for the folder after selecting the create button. The nesting of folders can go arbitrarily deep, though it’s usually just a single extra level of depth needed beyond a container. You will need to select the pseudo-folder you want to use prior to uploading an object you would like to store in that folder.



Containers

TWC_Lab	Object Count: 2 Size: 2.5 MB Access: Private
images	Object Count: 1 Size: 0 bytes Access: Private
images_segments	Object Count: 17 Size: 16.2 MB Access: Private
public_images	Object Count: 1 Size: 16.2 MB Access: Public
test_container1	Object Count: 1 Size: 6.8 KB Access: Private
volumebackups	Object Count: 411 Size: 280.5 MB Access: Private

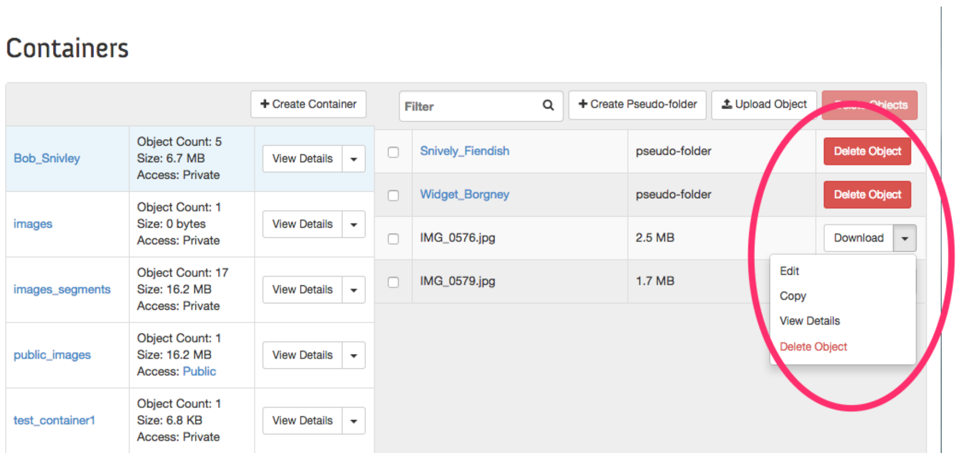
Displaying 6 items

TWC_Lab_Pseudo-folder	pseudo-folder
TWC_Object1	2.5 MB

Displaying 2 items

Download or Delete:

Files can be downloaded or deleted via the ‘Download’ drop down from the list of all folders and files. Note that containers and pseudo-folders must be empty before deletion.



Bob_Snivley	Object Count: 5 Size: 6.7 MB Access: Private	View Details
images	Object Count: 1 Size: 0 bytes Access: Private	View Details
images_segments	Object Count: 17 Size: 16.2 MB Access: Private	View Details
public_images	Object Count: 1 Size: 16.2 MB Access: Public	View Details
test_container1	Object Count: 1 Size: 6.8 KB Access: Private	View Details

Snively_Fiendish	pseudo-folder	Delete Object
Widget_Borgney	pseudo-folder	Delete Object
IMG_0576.jpg	2.5 MB	Download
IMG_0579.jpg	1.7 MB	

Exercise 2 Accessing Swift from the CLI

Prerequisites

Swift CLI

The Python based Swift Command Line Client (python-swiftclient) provides remote access to Swift Storage. You should have already installed this client in Lab 1.

Source your Credentials

If you have not already, source your openrc file to set your environment variables.

Verify your Installation

Once authentication parameters are configured, run

```
$ swift --help
```

to verify this service is working and to see a complete listing of available commands.

Exercise 2.1: Create a container and upload/download a file:

a. Create a container

Create a container and add an object requires passing both the container name and the filename simultaneously.

For example, to create a container called "test_contianer" upload a file called "test_file.txt" type, create a sample file to upload:

```
$ echo "Sample Data" > test_file.txt
```

Use “swift upload” command to upload your file:

Command Format:

```
$ swift upload [ContainerName] [FilePath]
```

Example:

```
$ swift upload test_container test_file.txt
```

Note: If you include the full path to the file (as in /tmp/test_file.txt), you will automatically have a /tmp pseudo folder created under the ContainerName. Normally one would upload from the current directory.

b. List all your Containers:

To list all the containers for an account, use “swift list” with no options or arguments.

```
$ swift list
```

Example Output:

```
test_container
```

c. List all objects of a container:

The swift list command will also display the contents of containers and accounts that are passed to it.

Command Format:

```
$ swift list [ContainerName]
```

Example input

```
$ swift list test_container
```

Example output

```
test_file.txt
```

d. Download object from a container:

To download an object from container:

Format:

```
$ swift download [ContainerName] [ObjectName]
```

Example:

```
$ swift list test_container test_file.txt
```

Example output:

```
test_file.txt [auth 0.372s, headers 0.465s, total 0.465s, 0.000 MB/s]
```

e. Get Temporary URL:

User can obtain a temporary URL with “swift-temp-url.”

Format:

```
$ swift tempurl [method] [seconds] [path] [key]
```

Where:

[method] is the method to allow; GET for example.
[seconds] are the number of seconds from command execution to allow requests.
[path] is the full path to the resource. For example: /v1/AUTH_account/c/o
[key] is the X-Account-Meta-Temp-URL-Key for the account.

Note: [path] includes your container “account” which is in the form either of “AUTH_…” or “KEY_…” depending on whether you’re on OpenStack Swift or on SwiftStack. You can determine which value your system is using with:

```
$ swift stat
```

Example output (partial):

```
              Account: KEY_7779a8fa71df44619f71d4fdb273ee04
           Containers: 4
              Objects: 24
                Bytes: 34030460
   Meta Temp-Url-Key: secret
                  ... : ...
```

The default is called "**secret**." Create a unique Temp-URL-Key for your account:

```
$ swift post -m "Temp-URL-Key:my_own_secret"
```

Now your default key is called: **my_own_secret** or whatever you chose as your secret. It is not the 'Meta Temp-Url-Key' part and you can see that it was set appropriately if you again run:

```
$ swift stat
```

Now to get a temporary url:

```
$ swift tempurl GET 300 /v1/KEY_[TenantID]/[ContName]/[ObjectName] my_own_secret
```

Example Output:

```
/v1/KEY_[TenantID]/[ContName]/[ObjectName]?temp_url_sig=[12345abcd]&temp_url_ex
pires=[12345]
```

And try to download it either in a browser, take the output from above and concatenate it with the Swift endpoint (e.g. https://chrcnc-api.os.cloud.twc.net) all on one line:

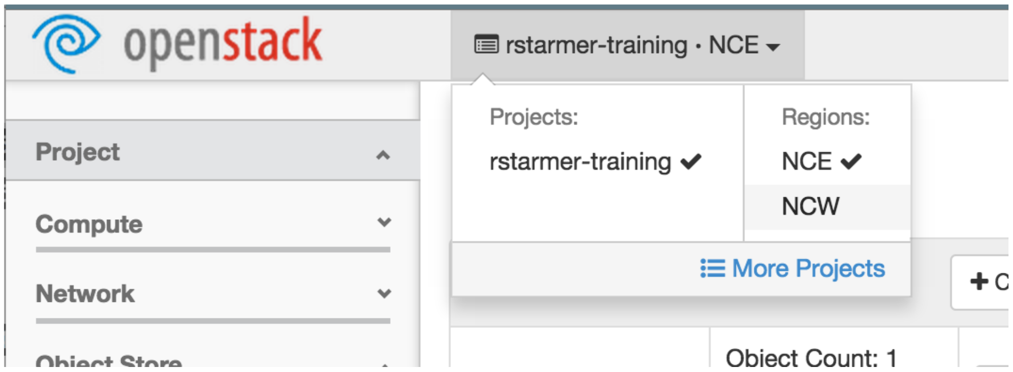
```
wget "https://chrcnc-api.os.cloud.twc.net/v1/KEY_[TenantID]/[ContName]/\

[ObjectName]?temp_url_sig=[12345abcd]&temp_url_expires=[12345]"
```

Note, that on the command line you must put the full URL in "" or the & in the URL will be interpreted by the command line!

You can also just paste the full url into a browser window (and then you don't need the "")

Exercise 2.2 Now for something duplicate… Regions

You have two regions available to deploy applications into, NCE and NCW. By default, swift objects will be loaded into the swift cluster in the region that is either set in Horizon (if you are using the UI), or in your openrc.sh script (or equivalent) from the command line. If you want to point at NCW instead of NCE (which appears to be the default), in Horizon, you toggle the location in the pulldown at the top of the screen:



In the openrc.sh script, the differences are both in the AUTH_URL and in the REGION:

```
$ diff nce.sh ncw.sh

33c33
< export OS_REGION_NAME="NCE"
---
> export OS_REGION_NAME="NCW"
```

If you configure two scripts, one for each region, you can then easily switch back and forth between them as needed. Since your password doesn't change, and it's never a good idea to store your password in a file, you might actually create three scripts:

1) A script to configure the base parameters, and to ask you for your password (to store it as an environment variable.
2) A script to set NCE region and Auth
3) A script to set NCW region and Auth

The easiest approach to this is to copy the Horizon provided openrc.sh to something like default-openrc.sh (perhaps with NCE credentials), and then create the following two files:

```
cat > nce.sh <<EOF
#!/bin/bash
export OS_REGION_NAME="NCE"
EOF

cat > ncw.sh <<EOF
#!/bin/bash
export OS_REGION_NAME="NCW"
EOF
```

Now, you can upload an object into NCE, and see it magically appear in NCW:

```
source {project-name}-openrc.sh
source nce.sh
swift upload test_container_1 test_file.txt
swift list test_container_1
```

```
source ncw.sh
mv test_file.txt test_file.txt.orig
swift list test_container_1
swift download test_container_1 test_file.txt
diff test_file.txt.orig test_file.txt
```

Note, generating temporary URLs will work with this switch as well, as the hostname portion of the URL is not part of the tokenized verification of authorization, so you can redirect your application user to the “closer” location to download their data, or provide a backup in case the initial path fails.

Exercise 2.3 And let’s not forget Expiration!

A feature of Swift that can also be very useful is to upload an image with an expiration date, or a Time-To-Live. This is accomplished on the CLI with the same swift upload command as before, with the addition of passing an additional header. The following will upload a file to SWIFT and will delete it 5 minutes later (300 seconds):

```
swift upload [CONTAINER] [FILE] --header X-Delete-After:300
```

You can also use the X-Delete-At header, which takes the Unix Epoch time for deletion. In bash, you should be able to get the epoch version of a date with the following (for example 16:15-27-Feb-15):

```
date -j -f "%H:%M-%d-%B-%y" 16:15-27-FEB-15 +%s

1425078910
```

Which you can then use as:

```
swift upload [CONTAINER] [FILE] --header X-Delete-At:1425078910
```

If you check either after the TTL expires, or after the date/time specified as an Epoch, you should no longer see your object!

Section 3: Cinder: OpenStack Block Storage

In this Lab Exercise, you will create a Cinder storage volume and attach it to a VM using the CLI. Using the Horizon UI is left as an exercise for the you and it should be straightforward after completing this lab.

In this lab you will also look at creating a snapshot of this instance and turning the snapshot into a new image for re-deployment.

Exercise 3.1 Access Cinder from the CLI

Prerequisites

Cinder CLI

The Python based Cinder Command Line Client (python-cinderclient) provides remote access to Cinder storage. You should have already installed this client in Lab 1.

Source your Credentials

If you have not already, source your openrc file to set your environment variables.

Verify Service

Once authentication parameters are configured, run

```
$ cinder list
```

to verify this service is working and to see a complete listing of available commands.

a. Create a volume:

Now you are ready to create a new volume using “cinder create”

Command Format:

```
$ cinder create --display-name [Volume_Name] [Required_Size_of_Volume_in_GB]
```

Example:

```
$ cinder create --display-name “MyVolume” 10
```

Use “cinder help create” to see the full syntax of the command as well as optional switches you can use.

Example Output:

```
+---------------------+---------------------------------------+
|       Property      |                 Value                 |
+---------------------+---------------------------------------+
|     attachments     |                  []                   |
|  availability_zone  |                 nova                  |
|       bootable      |                 false                 |
|      created_at     |      2015-01-30T18:07:15.277023       |
| display_description |                 None                  |
|     display_name    |              “MyVolume”               |
|      encrypted      |                 False                 |
|          id          | ab88d5c7-da0a-4c9e-a759-6ccc0b395907 |
|       metadata       |                  {}                  |
|         size         |                  2                   |
|     snapshot_id      |                 None                 |
|     source_volid     |                 None                 |
|        status        |               creating               |
|     volume_type      |                 None                 |
+----------------------+--------------------------------------+
```

b. List all Volumes:

Now when executing "cinder list" the new volume should be displayed:

```
$ cinder list
```

```
+-------------------+-----------+--------------+------+-------------+----------+-------------+
|        ID         |   Status  | Display Name | Size | Volume Type | Bootable | Attached to |
+-------------------+-----------+--------------+------+-------------+----------+-------------+
| ab88d5c7-...95907 | available |  "MyVolume"  |  2   |     None    |  false   |             |
+-------------------+-----------+--------------+------+-------------+----------+-------------+
```

c. Create bootable volume from Glance Image:

A volume is a detachable block storage device, similar to a USB hard drive. By default a volume can only be attached to one instance. A bootable volume can be created with any image and by using that volume only one instance can be launched.

In our first lab we used `nova image-list` to determine which disc images were available. Run this command again and then run the OpenStack Image Service equivalent:

```
$ glance image-list
```

Take a moment to look at the differences in the information provided by both commands and then make a note of the ID for the Ubuntu-Server-LTS14.04 image to incorporate into the volume you are about to create.

To create a bootable volume:
Command Format:

```
$ cinder create --image-id [Glance_Image_ID] --display-name [Volume_Name] [Size]
```

Example Input:

```
$ cinder create --image-id 3bbddb4b-9115-4da4-b198-aac18e3d7104 --display-name
"BootVolume" 20
```

Example Result

```
+----------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                 nova                 |
|       bootable      |                 true                 |
|      created_at     |      2015-01-30T18:10:35.714785      |
| display_description |                 None                 |
|     display_name    |              BootVolume              |
|      encrypted      |                False                 |
|          id         | c505fd02-b0c4-4413-b78a-faaa4b5a3f57 |
|       image_id      | 3bbddb4b-9115-4da4-b198-aac18e3d7104 |
|       metadata      |                  {}                  |
|         size        |                  1                   |
|     snapshot_id     |                 None                 |
|     source_volid    |                 None                 |
|        status       |               creating               |
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

The volume will need to be detached from any instances in order to either snapshot it or create a backup.

d. Create backup of a volume:

This makes a backup of the data on the volume and places it in Object Storage. The backup can then be restored to a new Volume. If you only have 1GB of data on a 2GB Volume, your backup will only be ~1GB, but the volume will still be considered as a 2GB file

Command format to backup a volume:

```
$ cinder backup-create [volume_id]
```

e. Restore Volume Backup:

Find the backup id:

```
$cinder backup-list
```

Restore from a volume backup:

```
$ cinder backup-restore [backup_id]
```

f. Delete Volume Backup:

Command format to delete backup:

```
$ cinder backup-delete [backup_id]
```

g. Snapshot:

Two things to keep in mind when doing snapshots:

1. It is recommended that the volume be detached and be in ’**available’** status in order for you to take a snapshot. You will receive an error if you try to snapshot a volume that's in-use, but you can use the --force True flag if you really need to execute the snapshot.
2. You must keep the original volume for the snapshot to function properly. If the original volume is deleted then the snapshot will become unusable.

h. Create snapshot of a volume:

This creates a replica of the volume and stores it in the Image service. New volumes can then be created based on the snapshot. A 2GB volume will have a 2GB snapshot even if you're only using 1GB worth of data on it.

To create snapshot of a volume:

```
$ cinder snapshot-create  <Volume_ID>
```

```
+----------------------+--------------------------------------+
|       Property       |                Value                 |
+----------------------+--------------------------------------+
|      created_at      |      2015-01-30T18:14:58.461270      |
| display_description  |                 None                 |
|     display_name     |                 None                 |
|          id          | 4a647cda-346b-43d2-8215-b69cdd39ac90 |
|       metadata       |                  {}                  |
|         size         |                  2                   |
|        status        |               creating               |
|      volume_id       | ab88d5c7-da0a-4c9e-a759-6ccc0b395907 |
+----------------------+--------------------------------------+
```

i. Create volume from Snapshot:

Create a new volume using the base command without setting the variables.

```
$ cinder create --snapshot-id 4a647cda-346b-43d2-8215-b69cdd39ac90 20
```

```
+----------------------+--------------------------------------+
|       Property       |                Value                 |
+----------------------+--------------------------------------+
|     attachments      |                  []                  |
|  availability_zone   |                 nova                 |
|       bootable       |                false                 |
|      created_at      |      2015-01-30T18:18:18.175236      |
| display_description |                 None                 |
|     display_name    |                 None                 |
|      encrypted      |                False                 |
|          id         | 49999974-bf7d-49a2-b276-c6f35a67c914 |
|       metadata      |                  {}                  |
|         size        |                  20                   |
|     snapshot_id     | 4a647cda-346b-43d2-8215-b69cdd39ac90 |
|     source_volid    |                 None                 |
|        status       |               creating               |
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

j. Delete Volume:

Delete a volume using this command: “cinder delete”

```
$ cinder delete <volume_id>
```

Section 4: Access Cinder from Horizon

Exercise 4.1

Now logon to the portal and try to replicate the tasks you just completed through the CLI in Horizon. Are there any tasks you are not able to replicate? Do you find the CLI or dashboard easier to complete tasks with? What if you had to do this tasks repeatedly would you go with dashboard or CLI?