Object Storage

- Some Use Cases
- Access
- RAC Dashboard
- Command Line (swift client)
- Swift API
- S3 API
    - s3cmd
- GUI Applications
    - Cyberduck
        - S3 Compatibility
        - OpenStack Swift Native
- In Depth Features / Tutorial
    - See usage
    - Create a Container
    - Upload a File
    - Add and Check metadata on a file
    - Download a File
    - Managing ACLs
    - Making Files Available via the Web
    - Versioning
    - Temporary URLs for Objects
    - Large/Segmented Object Warnings
    - Container Sync
        - Disabling Container Sync

Object Storage provides the ability to turn any file and any associated metadata into an object accessible by an HTTP API from anywhere. It ensures data integrity, provides data protection, and scales to millions of objects without requiring traditional filesystem management. Object Storage excels at handling unstructured data; and provides a location for you to store objects to be accessed by instances or if you wish generally online as well.

Object Storage doesn't have support for a deep hierarchy; objects are stored in containers but containers may not be inside another container. It's possible to mimic folders by using '/' in an object's name but from a system point of view there is no folder support.

Our implementation of object storage is powered by OpenStack's Swift.

Some Use Cases

- Static Web Assets
- Large amounts of unstructured data (eg. map tiles, photos, etc.)
- Archival (eg. tax records, backups, etc.)
- Consistency/Protection
- Apps that leverage extensive metadata on a file

Access

You can gain access to your containers and objects using several methods including the RAC Dashboard, the swift command line client, interacting with the Swift API itself, or using a popular GUI application. Of note at present we don't offer the S3 compatibility API used by some libraries to connect to Swift. (If you would like to use these feature please contact us at rac-admin@cybera.ca)

You can access your objects from anywhere in the world - whether it's on one or more of your instances (eg. a shared storage alternative), your personal computer, or elsewhere. By default only you (when provided with your credentials) can access your objects, and with ACLs you can manage read and write access for other users on a container level.

We offer Swift in both the Calgary region and the Edmonton region.

RAC Dashboard

You can create, delete, copy, and download objects and containers using the RAC Dashboard. Additional features such as setting metadata which is used for most advanced features are not available via the Dashboard.

Command Line (swift client)

If you haven't already, it's highly recommended you have reviewed Command-line Tools in the Advanced Guide. Installing the `python-openstackclient` should include installation of the `python-swiftclient`, but if you need to manually install it, follow the instructions below.

On Ubuntu:

```
pip install python-swiftclient
```

On MacOS, there may be a conflict with the Apple Swift compiler, so be sure the Python bin directory is first in your $PATH (as specified in OSX command line tools):

```
pip install --user python-swiftclient
```

After sourcing your openrc file you can then use the swift command to interact with the object storage. Examples can be found below.

Swift API

Additionally your application can leverage the Swift API directly - either using curl or existing libraries (such as Boto for Python).

S3 API

Our OpenStack Swift environment has support for S3 API compatibility. This means that any application or library which supports Amazon S3 *should* be able to be used in our Swift environment.

To begin using the S3 API, you first need to obtain a set of "EC2 Credentials". This is a set of credentials formatted in the same way as Amazon's credentials. You can download your EC2 Credentials by going here (https://cloud.cybera.ca/project/api_access/) and clicking "View Credentials".

Using a text editor of your choice, you'll want to create a new file (eg. s3cred.sh) with the following details:

```
export AWS_ACCESS_KEY_ID=foo
export AWS_SECRET_ACCESS_KEY=bar
```

Alternatively you can find these credentials via the OpenStack CLI tools with the command `openstack ec2 credentials list` .

The endpoint will be dependent on which region you are wishing to use - either `yyc.cloud.cybera.ca:8080` or `yeg.cloud.cybera.ca:8080`

s3cmd

For s3cmd the following example .cfg file can be used:

```
host_base = https://swift-yyc.cloud.cybera.ca:8080
host_bucket = %(bucket)s.swift-yyc.cloud.cybera.ca:8080
access_key = <access key>
secret_key = <secret key>
use_https = True
```

To use the Edmonton region, change the yyc in the URLs to yeg.

GUI Applications

Cyberduck

Cyberduck is available for Mac OS X and Windows.

S3 Compatibility

To use Cyberaduck using the standard S3 compatibility, first obtain your EC2 Credentials, as described in the S3 API of this document.

Open Cyberduck and create a new Amazon S3 connection. Fill in the following information:

- Server: `yyc.cloud.cybera.ca` (for the Calgary region) or `yeg.cloud.cybera.ca` (for the Edmonton region).
- Port: 8080
- Access Key ID: Your `EC2_ACCESS_KEY`
- Secret Access Key: Your `EC2_SECRET_KEY`

OpenStack Swift Native

Cyberduck provides built in support OpenStack Swift with Keystone V3. To create a connection, create a new OpenStack Swift (Version 3) connection with the details below:

More documentation is available on the Cyberduck Website.

Server	keystone-yyc.cloud.cybera.ca (Calgary) or keystone-yeg.cloud.cybera.ca (Edmonton)
Port	5000
Username	project-name:Default:user-name (eg. joe@cybera.ca:Default:joe@cybera.ca - note the colons in between the project, domain name, and user name)
Password/Secret Key	Your Password

⚠ DO NOT upload files larger than 2 GB using Cyberduck if you wish to share them via the Web.

In Depth Features / Tutorial

See Swift CLI reference for more information.

See usage

```
swift stat -v
swift stat -v container_name
swift stat container_name file_name
```

Create a Container

```
swift post container_name
```

Upload a File

```
echo 'Hello World' > file_on_my_computer.txt
swift upload container_name file_on_my_computer.txt
```

Add and Check metadata on a file

```
echo 'Another file' > another_file_on_my_computer.txt
swift upload container_name another_file_on_my_computer.txt
swift upload -m "X-Object-Meta-Hello: World" container_name another_file_on_my_computer.txt
swift stat -v container_name another_file_on_my_computer.txt
```

Download a File

```
swift download container_name file_on_my_computer.txt
```

Managing ACLs

```
# Set a container to be public
swift post -r '.r:*' container_name

# Allow another project to rw to container
swift post -w 'second_project_id:*' container_name
swift post -r 'second_project_id:*' container_name

swift stat -v container_name
```

The second project may then interact with the shared container by appending the storage-url for the container to their swift commands:

```
swift stat -v --os-storage-url https://swift-<region>.cloud.cybera.ca:8080/v1/AUTH_<first_project_id>
container_name
```

Making Files Available via the Web

Set container to be public to anyone

```
swift post -r '.r:*' container_name
```

Figure out URL: by finding the AUTH_xxxx string by running `swift stat`

eg. http://swift.cloud.cybera.ca/v1/AUTH_xxxx/filename.txt or by using Cyberduck (see above) and choose Copy URL.

Alternatively you can also enable web listings so you can see all the objects in a container:

```
swift post -r '.r:*,.rlistings' container_name
swift post -m 'web-listings: true' container_name
```

And then visit http://swift.cloud.cybera.ca/v1/AUTH_xxxx/container_name to view all the objects (changing AUTH_xxxx for your AUTH information)

Versioning

Another feature Swift offers is automatic versioning. This means every time an object is uploaded or updated it will store the old version in a special versions container that is created.

This means the older versions are still all accessible in a second container. You can enable this by setting the ‘X-Versions-Location’ header on your container:

```
swift post -m 'X-Versions-Location: myContainer-versions' myContainer
```

Temporary URLs for Objects

One last feature we wanted to highlight is the ability to offer temporary URLs for objects or expiring URLs as they are also called. This allows you to provide a URL that will stop functioning after a certain amount of time.

To do this you need to be able to set a secret key in a header on your account to allow Temporary URLs, and then create an HMAC-SHA1 signature generated from the HTTP method, the expiration timestamp, the path to the object, and the secret key set in the header before. It’s not as complicated as it sounds however.

The first step is to create the secret key for your account:

```
swift post -m 'X-Account-Meta-Temp-URL-Key: secretkeygoeshere'
```

Then we need to create our temporary URL. You can use a tool such as swift-temp-url (Python), or have your application create the link by creating a signature based on the method, expiration timestamp, and the path. The above linked tool uses the following code to do this:

```
import hmac
from hashlib import sha1
from time import time
method = 'GET'
expires = int(time() + 60)
path = '/v1/AUTH_xxxx/container/object'
key = 'mykey'
hmac_body = '%s\n%s\n%s' % (method, expires, path)
sig = hmac.new(key, hmac_body, sha1).hexdigest()
```

To use swift-temp-url - download the file to your computer, and set the file to be executable. You can then run it and get your URL. eg. a link that lasts for 10 minutes (600 seconds)

```
echo https://swift-REGION.cloud.cybera.ca:8080$(swift-temp-url GET 600 /v1/AUTH_xxxx/public_container
/my_fancy_object secretkeygoeshere)
```

Results in a shareable link:

https://swift-yyc.cloud.cybera.ca:8080/v1/AUTH_xxxx/public_container/my_fancy_object?
temp_url_sig=af0cf3b7597f3bca86895e3796b834c6a93d6a12&temp_url_expires=1401991851

Be sure to set set your ACCOUNT ID, and then the path (proper container name and object name)

https://swift-yyc.cloud.cybera.ca:8080/v1/AUTH_xxxx/public_container/my_fancy_object?
temp_url_sig=af0cf3b7597f3bca86895e3796b834c6a93d6a12&temp_url_expires=1401991851

Large/Segmented Object Warnings

Swift segments files to support objects larger than 5 GB. If you want to upload a file larger than 5GB using the Swift CLI you will need to ensure you use the -S flag and set a segment size (below is 4GB segment size).

```
swift upload container_name -S 4294967296 my_large_fils.tar
```

By default Swift splits files larger than 5 GB while Cyberduck performs this for if the file is larger than 2GB. There are a couple caveats with the different approaches the two applications take to handling large files when it comes to accessing these files via a web browser (completely unrelated to the size they use to split files).

Object uploaded via Swift CLI:

Swift will create a <containername_segments> container that holds the separate files. You'll need to enable read access AND rlistings for this container in order for the download to work. For a container named public you'll need to make sure the following is set:

```
swift post -r '.rlistings' public_segments
```

Once both are set, downloading the object via your web browser as an unauthenticated user will work as expected. (Bug #1082835)

Object uploaded via Cyberduck:

Cyberduck creates a folder called .file_segments in your container and places the segments in there. It then writes JSON data to the manifest file so you can download the file later

Container Sync

Container sync offers the ability to synchronize the contents of two or more containers between the two regions of the Rapid Access Cloud or if supported, other OpenStack Swift installations. Every five minutes, a sync will run copying any new or changed metadata and objects (including deletions) from the source container to the destination container. Depending on the size of the objects it may take a couple more minutes to copy the object in it's entirety.

This is done by creating a sync relationship between any two swift containers. In the example below, we will create a container in Calgary (container1) and another in Edmonton (container2). Each container will sync to the other (two-way replication) which is useful if you want the objects to be highly available. We will need a key for each container to share for security (the key will simply be 'secret', but for production please use a complex password).

1. Determine your AUTH ID (Account ID, line 3):

```
$ source openrc-yyc
$ swift stat -v
StorageURL: https://swift-yyc.cloud.cybera.ca:8080/v1/AUTH_1a2b3c4d5e6f7890
Auth Token: this_is_not_a_real_token
Account: AUTH_1a2b3c4d5e6f7890
Containers: 1
Objects: 116
Bytes: 103445941
Containers in policy "policy-0": 1
Objects in policy "policy-0": 116
Bytes in policy "policy-0": 103445941
X-Timestamp: 1418065112.84439
X-Trans-Id: txed09fb697477491b9af8d-0055e5fcce
Content-Type: text/plain; charset=utf-8
Accept-Ranges: bytes
```

2. Create or modify a container in Calgary and include the `--sync-to` and `-k` attributes (the `swift post` command updates info for a container, or creates one if it doesn't exist):

```
$ swift post --sync-to '//rac/yeg/AUTH_1a2b3c4d5e6f7890/container2' -k 'secret' container1
```

3. Create or modify a container in Edmonton to sync to Calgary:

```
$ swift post --sync-to '//rac/yyc/AUTH_1a2b3c4d5e6f7890/container1' -k 'secret' container2
```

It is possible to do a one-way sync for disaster recovery, effectively creating a backup container; in our example above, if we wanted to sync one-way from yyc => yeg we create the container in Calgary the same, but the container in Edmonton will not have a `--sync-to` switch, only a `-k`:

```
$ swift post -k 'secret' container2
```

Now objects in Calgary will replicate to Edmonton, but not the other way.

Disabling Container Sync

At present disabling container sync is not available in the `openstack` command line tool at the time of writing. You must use the `swift` command.

One example of why you might want to do this if you previously had a two-way container sync with YEG YYC and only wish to sync one-way now.

To remove the sync setting, provide an empty `--sync-to` parameter instead versus filling in the parameter to set up sync as seen in the section above.

```
$ swift post --sync-to '' container
```

If you'd like to disable the sync entirely, run the command on each container in both regions.