Implementing Cloud Storage with OpenStack Swift

Amar Kapadia
Sreedhar Varma
Kris Rajana

Chapter No. 4
"Using Swift"
In this package, you will find:

A Biography of the authors of the book
A preview chapter from the book, Chapter NO.4 "Using Swift" A synopsis of the book’s content
Information on where to buy this book

About the Authors

**Amar Kapadia** is a storage technologist and blogger based in the San Francisco Bay Area. He is currently the Senior Director of Strategy for EVault's Long-Term Storage Service, a subsidiary of Seagate. With over 20 years of experience in storage, server, and I/O technologies at Emulex, Philips, and HP, Amar's current passion is cloud and object storage technologies based on OpenStack Swift. He holds a Master's degree in Electrical Engineering from the University of California, Berkeley.

When not working on OpenStack Swift, Amar can be found working on Open Compute Platform technologies, MongoDB, PHP, AJAX, or jQuery. Amar's blogs can be found at [buildcloudstorage.com](http://buildcloudstorage.com).

I would like to thank my wife for tolerating my late night and weekend book-writing sessions. I would also like to thank the Long-Term Storage Service team at EVault who generously helped provide content and critique on various chapters.

For More Information:
**Sreedhar Varma** has more than 15 years of experience in the storage industry, developing storage software and solutions. He has worked on various storage technologies (such as SCSI, SAS, SATA, and FC), HBA drivers (Adaptec, Emulex, Qlogic, Promise, and so on), RAID, and storage stacks of various operating systems. He was involved in building system software for Stratus Fault Tolerant and High Availability systems. He has good working experience with SAN, NAS, and iSCSI networks as well as various storage arrays (Dothill, IBM, EMC, Hitachi, and Oracle Pillar). Sreedhar is currently involved with object storage implementations (Swift, Ceph) and developing software using corresponding REST APIs.

Sreedhar has a Master's degree in Computer Science from the University of Massachusetts.

He is presently working for Vedams Software (providing storage engineering services). In the past, he has worked for Stratus Technologies, Compaq, Digital Equipment Corp, and IBM.

---

I would like to thank my wife for her support and encouragement while I was writing the chapters for this book. I would also like to acknowledge the assistance of Vedams and EVault OpenStack teams in building and managing an OpenStack cluster. This enabled us to verify every aspect covered in this book, including installation, testing, and tuning with clear instructions on how-to.

---

**Kris Rajana** is an entrepreneur, passionate in building globally distributed teams to develop and maintain innovative products and solutions. His areas of interests include tape, DAS, NAS, SAN, and fast emerging technologies (Cloud, SDN, SDS, and Flash Arrays). Kris has over 20 years of experience in managing engineering teams in areas including space and aviation at BFGoodrich Aerospace and storage at Snap Appliance (currently Overland Storage) Adaptec, Xyratex, and Sullego. Currently, as the CEO of Vedams, Kris takes immense pride in his team and its development that leads to execution excellence. Kris's current passion is application of Big Data concepts to improve reliability and uptime of systems.

Kris is a student and sevak at San Jose Chinmaya Mission. Kris also serves on the board of the Pratham Bay Area Chapter. Kris and Vedams sponsor the Pratham Urban Learning Center in Hyderabad.

Kris earned his doctorate in engineering science from the Pennsylvania State University and keeps abreast with emerging management methodologies through his affiliation with Stanford University.

---

I would like to thank my family for their encouragement. Finally, I would like to thank the Vedams team and my mentors over the years.

---

For More Information:

Implementing Cloud Storage with OpenStack Swift

CIOs around the world are asking their teams to take advantage of cloud technologies as a way to slash costs and improve usability. OpenStack is a fast-growing open source cloud software with a number of projects. Swift is one such project that allows users to build cloud storage. With Swift, not only can users build storage using inexpensive commodity hardware, but they can also use the public cloud storage built using the same technology. Starting with the fundamentals of cloud storage and OpenStack Swift, this book will provide you with the skills to build and operate your own cloud storage or use a third-party cloud. This book is an invaluable tool if you want to get a head start in the world of cloud storage using OpenStack Swift. The readers of this book will be equipped to build an on-premise private cloud, manage it, and tune it.

What This Book Covers

Chapter 1, Cloud Storage – Why Can’t I be Like Google?, introduces the need for cloud storage, the underlying technology of object storage, and an extremely popular open source object storage project called OpenStack Swift.

Chapter 2, OpenStack Swift Architecture, discusses the internals of the Swift architecture in detail and shows how elegantly Swift converts commodity hardware into reliable and scalable cloud storage.

Chapter 3, Installing OpenStack Swift, walks you through all the necessary steps required to perform a multi-node Swift installation and how to set it up along with the Keystone setup for authentication.

Chapter 4, Using Swift, describes the various ways you can access Swift object storage. It also provides examples for the various access methods.

Chapter 5, Managing Swift, provides details on the various options that are available to monitor and manage a Swift cluster. Some of the topics covered in this chapter include StatsD metrics, handling drive failures, node failures, and migrations.

Chapter 6, Choosing the Right Hardware, provides you with the information necessary to make the right decision in selecting the required hardware for your cloud setup.

For More Information:
Chapter 7, Tuning Your Swift Installation, walks you through a performance benchmarking tool and the basic mechanisms available to tune a Swift cluster. Users utilizing Swift will need to tune their installation to optimize performance, durability, and availability, based on their unique workload.

Chapter 8, Additional Resources, explores several use cases of Swift and provides pointers on operating systems, virtualization, and distribution tools being used across various Swift installations.

Appendix, Advanced Features, provides details on various commands that can be run from a Swift CLI session.

For More Information:
This chapter explains the various mechanisms that are available to access Swift. Using these mechanisms, we will be able to authenticate accounts, list containers, create containers, create objects, delete objects, and so on. Tools and libraries such as Swift Client CLI, cURL client, HTTP REST API, JAVA libraries, Ruby OpenStack libraries, and Python libraries use Swift APIs internally to provide access to the Swift cluster. In particular, we will be using the Swift Client CLI, cURL, and HTTP REST API to access Swift and perform various operations on containers and objects. Also, we will be using EVault's Long-Term Storage (LTS2) cloud storage to demonstrate the use of Swift.

Installing the clients

This section talks about installing cURL and Swift's client CLI command line tools. In this section we describe how to install these tools on a Ubuntu 12.04 Linux operating system. Please refer to the other Linux distribution command sets for installing the clients in those operating systems. Windows and Mac version of these tools are also available. The following commands are used to install the cURL and the Swift Client CLI:

- **cURL**: This is a command-line tool that can be used to transfer data using various protocols. The following command is used to install cURL:
  
  ```
  # apt-get install curl
  ```

- **Swift Client CLI**: This is a tool to access and perform operations on a Swift cluster. This tool is installed using the following command:
  
  ```
  # apt-get install python-swiftclient
  ```

- **REST API Client**: To access Swift services via the REST API, we can use third-party tools such as Fiddler web debugger that supports REST's architecture.

For More Information:

Creating a token using authentication

The first step in order to access containers or objects is to authenticate the user by sending a request to the authentication service and get a valid token that can then be used in subsequent commands to perform various operations. We are using Keystone authentication in our configuration and the examples shown in this chapter. There is another method of authentication called Swauth that can be used. It works in a slightly different way, but we don't deal with the details of Swauth here.

The following command is used to get the valid keystone authentication token:

```bash
# curl -X POST -i https://auth.lts2.evault.com/v2.0/Tokens -H 'Content-type: application/json' -d '{"auth":{"passwordCredentials":{"username":"user","password":"password"},"tenantName":"tenant1"}}'
```

In the preceding command, `https://auth.lts2.evault.com/v2.0` is EVault's authentication endpoint. Along with this the username, password, and the tenant name are also provided.

The token that is generated is shown as follows (it has been truncated for better readability):

```
token = MIIGIwYJKoZIhvcNAQcCoIIGFDCCBhACAgEExCTAHBqUrDgMCQjCCBhGkCSqGSIb3DQEHAaCCBGoEggRme..yJhY2Nlc..IoIIBIvRv2UjIoge..Jpc3NlZWRfYXQIoAiMjAxMy0xMS0yNQw6NjoxOD0M4izNTAONTcILCU+KNYN20G7KJO05bXb..5VF18z16JqAKKWENTr1KJvzFz0..-peLbwcKZXTp..JkxqK7Vpzc-NIySwPWjODs--0WTes+Cy0RD
```

This token is then used as a parameter in the commands accessing Swift, for example, in the following command:

```bash
curl -X HEAD -i https://storage.lts2.evault.com/v1/26cef4782cca4e5aabbb9497b8c1ee1b
-H 'X-Auth-Token: token' -H 'Content-type: application/json'
```

More details on the commands are provided in the upcoming sections.

Displaying metadata information for an account, container, or object

This section describes how we can obtain information about the account, container, or object.
Using the Swift Client CLI

The Swift Client CLI `stat` command is used to get information about the account, container, or object. The name of the container should be provided after the `stat` command to get container information. The name of the container and object should be provided after the `stat` command to get object information.

Execute the following request to display the account status:

```
# swift --os-auth-token=token --os-storage-url= https://storage.lts2.evault.com/v1/26cef4782cca4e5aabb9497b8c1ee1b stat
```

In the preceding commands, `token` is the generated token as described in the previous section and `26cef4782cca4e5aabb9497b8c1ee1b` is the account name.

The response shows the information about the account, which is as follows:

```
Account: 26cef4782cca4e5aabb9497b8c1ee1b
Containers: 2
Objects: 6
Bytes: 17
Accept-Ranges: bytes
Server: nginx/1.4.1
```

Using cURL

The following command shows how to obtain the same account information using cURL. It shows that the account contains two containers and six objects.

Execute the following request:

```
# curl -X HEAD -i https://storage.lts2.evault.com/v1/26cef4782cca4e5aabb9497b8c1ee1b
-H 'X-Auth-Token: token' -H 'Content-type: application/json'
```

The response to the preceding command is as follows:

```
HTTP/1.1 204 No Content
Server: nginx/1.4.1
Date: Wed, 04 Dec 2013 06:53:13 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 0
```
Using the REST API

Fiddler web debugger, which supports REST, was used to send the request and receive the HTTP response. Execute the following request:

Method : HEAD
URL    : https://storage.lts2.evault.com/v1/26cef4782cca4e5aabb9497b8c1ee1b
Header : X-Auth-Token: token
Data   : No data

The response is as follows:
HTTP/1.1 204 No Content
Server: nginx/1.4.1
Date: Wed, 04 Dec 2013 06:47:17 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 0

As you can see, this is a different mechanism of issuing the command, but is very similar to accessing the Swift cluster using cURL.

Listing containers

This section describes how to obtain information about the containers present in an account.

Using the Swift Client CLI

Execute the following request:

swift --os-auth-token=token --os-storage-url= https://storage.lts2.evault.com/v1/26cef4782cca4e5aabb9497b8c1ee1b list

The response to the preceding request is as follows:

cities
countries

For More Information:

Using cURL
The following command shows how to obtain the same containers information using cURL. It shows that the account comprises of two containers and six objects.

Execute the following request:

curl -X GET --i https://storage.lts2.evault.com/v1/26cef4782cca4e5aabb9497b8cleelb -H 'X-Auth_token: token'

The response to the request is as follows:

HTTP/1.1 200 OK
X-Account-Container-Count: 2
X-Account-Object-Count: 6

cities
countries

Here we see that the output has header and body, whereas in the previous example, we only had header and no body in the output.

Listing objects in a container
This section describes how to list the objects that are present in a container.

Using the Swift Client CLI
The following command shows how to list objects using the Swift Client CLI (in this example we are listing out the objects in the cities container):

Execute the following request:

swift --os-auth-token=token --os-storage-url= https://storage.lts2.evault.com/v1/26cef4782cca4e5aabb9497b8cleelb list cities

The response to the request is as follows:

London.txt
Mumbai.txt
NewYork.txt

For More Information:
Using Swift

Using cURL

The following command shows how to list objects using cURL. In this example, we list the objects in the Cities container.

Execute the following request:

curl -X GET -i https://storage.lts2.evault.com/v1/26cef4782cca4e5aabbb9497b8c1ee1b/cities
    -H 'X-Auth-Token: token'

The response of the request is as follows:

HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Content-Length: 34
X-Container-Object-Count: 3

London.txt
Mumbai.txt
NewYork.txt

Using the REST API

In this example, we list the objects in the countries container.

Execute the following request:

Method : GET
URL    : https://storage.lts2.evault.com/v1/26cef4782cca4e5aabbb9497b8c1ee1b/countries
Header : X-Auth-Token: token
Data   : No content

The response to the request is as follows:

HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Content-Length: 38
X-Container-Object-Count: 3

France.txt
India.txt
UnitedStates.txt

For More Information:
Updating the metadata for a container
This section describes how to add or update metadata for a container.

Using the Swift Client CLI
In this example, we are adding metadata for countries that we have visited.

Execute the following request:

```
swift --os-auth-token=token --os-storage-url=https://storage.lts2.evault.com/v1/26cef4782cca4e5aabbb9497b8clee1b post countries
-H "X-Container-Meta-Countries: visited"
```

Using the REST API
Here we are adding metadata using the REST API.

Execute the following request:

```plaintext
Method : POST
URL    : https://storage.lts2.evault.com/v1/26cef4782cca4e5aabbb9497b8cle1b/countries
Header : X-Auth-Token: token
          X-Container-Meta-Countries: visited
Data   : No content
```

Environment variables
The following environment variables can be used to simplify the CLI commands:

- `OS_USERNAME`: This contains the username to access the account
- `OS_PASSWORD`: This contains the password associated with the username
- `OS_TENANT_NAME`: This contains the name of the tenant
- `OS_AUTH_URL`: This contains the authentication URL

Once these environment variables are exported, we no longer have to pass these values as input parameters when running the Swift CLI tools.

For More Information:
Pseudo-hierarchical directories
OpenStack Swift object storage can simulate a hierarchical directory structure in containers by including a / (forward slash character) in the object's name.

Let's upload a file (AMERICA/USA/Newyork.txt) into the Continent container using the following command:

```
# swift upload Continent AMERICA/USA/Newyork.txt
```

Let's list the Continent container that has a few pseudo-hierarchical folders by using the following commands:

```
# swift list Continent
AMERICA/USA/Newyork.txt
ASIA/ASIA.txt
ASIA/China/China.txt
ASIA/INDIA/India.txt
Australia/Australia.txt
continent.txt
```

We can use / as the delimiter parameter to limit the displayed results. We can also use the prefix parameter along with the delimiter parameter to view the objects in the pseudo directory along with pseudo directories within that. The following are a couple of examples showing the use of these parameters:

```
# swift list Continent --delimiter /
AMERICA/
ASIA/
Australia/
continent.txt

# swift list Continent --delimiter / --prefix ASIA/
ASIA/ASIA.txt
ASIA/China/
ASIA/INDIA/

# swift list Continent --delimiter / --prefix ASIA/INDIA/
ASIA/INDIA/India.txt
```
Container ACLs

As we saw in the previous sections, in order to access containers and objects, a valid auth token has to be sent in the X-Auth-Token header with each request. Otherwise, an authorization failure code will be returned. In certain cases, access needs to be provided to other clients and applications for certain containers and objects. Access can be provided by setting a metadata element for the container called X-Container-Read. The following example sets this Access Control Lists (ACL) to the cities container:

First, let us list the container status that shows the lack of ACL. Run the following command with admin privileges (the admin user will have the permissions to run this command):

```
swift stat cities
```

The values for Read ACL and Write ACL in the following response indicates the lack of ACL:

```
Account: 26cef4782cca4e5aabbb9497b8c1ee1b
Container: cities
Objects: 3
Read ACL: 
Write ACL: 
Sync To: 
```

When the tenant1:user1 user, who does not have access to this container, tries to access this container, a forbidden error message is returned.

Execute the following request:

```
swift -V 2.0 -A https://auth.lts2.evault.com/v2.0 -U tenant1:user1 -K t1 list cities
```

A forbidden error message is returned as the response. This error is as follows:

```
Container GET failed: 403 Forbidden
Access was denied to this resource
```

In the preceding example, the username is provided using the -U option and the key to access the account is provided using the -K option.

Now, let's set the X-Container-Read metadata element and enable READ access for tenant1:user1. This operation can only be done by the admin user by using the following command:

```
swift post -x tenant1:user1 cities
```

For More Information:

To check the ACL permissions, we execute the following command:

```
swift stat cities
```

The response to the preceding command is as follows:

```
Account: 26cef4782cca4e5aabb9497b8c1ee1b
Container: cities
Objects: 3
Read ACL: tenant1:user1
Write ACL:
Sync To:
```

Now, when the `tenant1:user1` user tries to access this container, access is granted and the command is successfully executed.

Execute the following request:

```
swift -V 2.0 -A https://auth.1ts2.evault.com/v2.0 -U tenant1:user1 -K t1 list cities
```

The response to the request is as follows:

```
London.txt
Mumbai.txt
NewYork.txt
```

Since the `X-Container-Write` ACL is not set for the `tenant1:user1` user for the `cities` container, this user cannot write to the `cities` container. In order to allow write access, let's set the `X-Container-Write` ACL as follows:

```
swift post -w tenant1:user1 cities
```

To check the ACL permissions, we execute the following command:

```
swift stat cities
```

The response to the preceding command is as follows:

```
Account: 26cef4782cca4e5aabb9497b8c1ee1b
Container: cities
Objects: 3
Read ACL: tenant1:user1
Write ACL: tenant1:user1
Sync To:
```

Now the `tenant1:user1` user will be able to write objects into the `cities` container.

For More Information:

If we want to give access to a large number of users, ACLs such as .r:*, .rlistings can be used. The .r:* prefix allows any user to retrieve objects from the container and .rlistings turns on listing for the container.

**Transferring large objects**

As discussed in Chapter 2, *OpenStack Swift Architecture*, Swift limits a single object upload to 5 GB. Larger objects can be split into 5 GB or smaller segments by specifying the segment-size option in the swift CLI tool command-line argument and uploaded to a special container (created within the container where the object is being uploaded to).

Once the upload has been completed, a manifest object has to be created that contains information about the segments. The manifest file is of zero size with headers such as `X-Object-Manifest` identifying the special container in which the segments are stored and the name with which all the segments will start. For example, if we have to upload `France.txt`, which is of size 8 GB, to the `countries` container, then the `France.txt` object has to be split into two chunks (5 GB and 3 GB). The chunk object’s name will start with `France.txt` (`France.txt/../00000000` and `France.txt/../00000001`).

A special container called `countries_segments` will be created and the chunks will be uploaded to this container. A manifest object called `France.txt` will be created in the `countries` container. The manifest file will have zero size and will contain the following header. (It is not mandatory to have the segments placed in a special container and they can as well exist in the same container):

```
X-Object-Manifest: countries_segments/France.txt
```

When a download request is made for the large-sized object, Swift will automatically concatenate all the segments and download the entire large-sized object.

The Swift Client CLI has the `-S` flag, to specify the segment size, which can be used to split a large object into segments and upload. The following command is used to upload a file with a segment size of 5368709120 bytes:

Make the following request:

```
swift upload countries -S 5368709120 France.txt
```

The response to the preceding commands is as follows:

```
France.txt segment 0
France.txt segment 1
France.txt segment 2
France.txt
```

For More Information:

Using Swift

The following command can be used to list out the containers present:

```swift
Swift list
```

The response to the preceding command is as follows:

```swift
Countries
Countries_segments
cities
```

The following command lists the objects in the `countries_segments` container:

```swift
Swift list countries_segments
```

The response to the preceding command is as follows:

```swift
France.txt/1385989364.105938/5368709120/00000000
France.txt/1385989364.105938/5368709120/00000001
```

**Amazon S3 API compatibility**

Users familiar with the Amazon S3 API and accessing S3 buckets and objects can access Swift using S3 compatible APIs with the help of Swift3 middleware.

Here, we will show the steps required for one method that uses S3 APIs to access Swift's object store. These steps explain how to install the necessary tools and packages, create credentials, and update the configuration files.

The following steps are performed on the proxy-server node that is running the Ubuntu 12.04 Linux distribution:

1. First, the user requires EC2 credentials (access key and secret key). The `keystone user-list` and `keystone tenant-list` commands can be used to obtain the user ID and tenant ID of the user. The following command can be used to create these keys (these need to be run from the proxy server):

   ```
   keystone ec2-credentials-create --user-id 916673a90b8749e18f0ee3ec5bf17ab9 --tenant-id 6530edfe037242d1ac8bb07b7fd76046
   ```

   The response is as follows:

   ```
   +-----------+----------------------------------+
   |  Property |              Value               |
   +-----------+----------------------------------+
   |   access  | 1178d235dbd84d48b417170ec9aed72c |
   |   secret  | c4ea0a8f8f7d56f0d0f5c07d55b     |
   ```

For More Information:

2. Install the Swift3 package by running the following commands (these commands require Git to be installed on your system):

```
# sudo git clone https://github.com/fujita/swift3.git
# cd swift3
# sudo python setup.py install
```

3. Install the `libdigest-hmac-perl` package by running the following command (this package is used for integrity checking between two entities that share a secret key):

```
apt-get install libdigest-hmac-perl
```

4. Edit the `proxy-server.conf` file and make the following changes if you want to use the keystone authentication:

   - Change the pipeline line in the `proxy-server.conf` file to:
     ```
     [pipeline:main]
     pipeline = catch_errors cache swift3 s3token authtoken
     keystone proxy-server
     ```
   - Add a Swift3 WSGI filter to the `proxy-server.conf` file using the following command:
     ```
     [filter:swift3]
     use = egg:swift3#swift3
     ```
   - Add the `s3token` filter as in the following commands:
     ```
     [filter:s3token]
     paste.filter_factory = keystone.middleware.s3_token:filter_factory
     auth_port = 35357
     auth_host = 127.0.0.1
     auth_protocol = http
     ```
   - Restart the proxy service using the following command:
     ```
     Service swift-proxy restart
     ```

5. The following steps should be performed on the client that will access Swift Object Store:

   - Since we will use `s3curl` to execute the S3 commands, download `s3-curl.zip` from the following link:
     ```
     http://s3.amazonaws.com/doc/s3-example-code/s3-curl.zip
     ```

For More Information:
Using Swift

- Install the `wget` utility prior to running the following command:
  ```bash
  wget http://s3.amazonaws.com/doc/s3-example-code/s3-curl.zip
  ```

- Unzip `s3-curl.zip` and provide executable access to the `s3curl.pl` file.

- Create a `.s3curl` file and change the ID and key of personal account with the EC2 credentials (access and secret keys) that were given to the user. We are using vi editor to create the file as shown in the following:
  ```bash
  # vi ~/.s3curl
  %awsSecretAccessKeys = {
    # personal account
    personal => {
      id => '1178d235dbd84d48b417170ec9aed72c',
      key => 'c4ea0a8f6f7d4a69f6d0fb5c6b47d5b',
    },
    # corporate account
    work => {
      id => '1ATXQ3HHA59CYF1CVS02',
      key => 'WQY4SrSS95pJUT95V6zWea0lgBKCBCL6PI0cdxeH8',
    },
  };
  ```

**Accessing Swift using S3 commands**

In this section, we will give examples of S3 commands to perform various operations.

- **List buckets**: This command lists all the buckets for this user. Buckets in S3 are similar to containers in Swift.
  ```bash
  # ./s3curl.pl --id=personal -- https://auth.lts2.evault.com --v
  ```
  The response is as follows:

  ```xml
  <?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult xmlns="http://doc.s3.amazonaws.com/2006-03-01"><Buckets>
    <Bucket><Name>cities</Name><CreationDate>2009-02-03T16:45:09.000Z</CreationDate></Bucket>
    <Bucket><Name>countries</Name><CreationDate>2009-02-03T16:45:09.000Z</CreationDate></Bucket>
  </Buckets></ListAllMyBucketsResult>
  ```

For More Information:

• **List objects in a bucket**: This command lists all the objects present in the specified bucket. Let us list all the objects in the `cities` bucket by using the following command:

```
# ./s3curl.pl --id=personal -- https://auth.lts2.evault.com/cities -v
```

• **Create a Bucket**: The following command creates a bucket called `continents`:

```
# ./s3curl.pl --id=personal --createBucket -- -v https://auth.lts2.evault.com/continents
```

• **Delete a Bucket**: The following command deletes a bucket called `continents`:

```
# ./s3curl.pl --id=personal --delete -- -v https://auth.lts2.evault.com/continents
```

### Accessing Swift using client libraries

There are several libraries available in Java, Python, Ruby, PHP, and other programming languages to access the Swift cluster. Applications can be simplified using these libraries. Let us explore a few libraries.

#### Java

The Apache jclouds library ([http://jclouds.apache.org/documentation/quickstart/rackspace/](http://jclouds.apache.org/documentation/quickstart/rackspace/)), particularly the `org.jclouds.openstack.swift.CommonSwiftClient` API can be used to write applications in Java to connect to Swift and perform various operations on accounts, containers, and objects.

A sample code is shown as follows:

```java
import org.jclouds.ContextBuilder;
import org.jclouds.blobstore.BlobStore;
import org.jclouds.blobstore.BlobStoreContext;
import org.jclouds.openstack.swift.CommonSwiftAsyncClient;
import org.jclouds.openstack.swift.CommonSwiftClient;

BlobStoreContext context = ContextBuilder.newBuilder(provider)
    .endpoint("http://auth.lts2.evault.com/")
    .credentials(user, password)
    .modules(modules)
    .buildView(BlobStoreContext.class);
storage = context.getBlobStore();
```

For More Information:

Using Swift

```swift
swift = context.unwrap();
containers = swift.getApi().listContainers();
objects = swift.getApi().listObjects(myContainer);
```

Python

The python-swiftclient library provides Python language bindings for OpenStack Swift. After authentication, the following sample code shows how to list containers:

```python
#!/usr/bin/env python

http_connection = http_connection(url)
cont = get_container(url, token, container, marker, limit, prefix,
delimiter, end_marker, path, http_conn)
```

More information about the library is provided at https://github.com/openstack/python-swiftclient/.

Ruby

The ruby-openstack library (https://github.com/ruby-openstack/ruby-openstack) provides ruby bindings for the OpenStack cloud. The following sample code shows how to list containers and objects:

```ruby
Lts2 = OpenStack::Connection.create(:username => USER, :api_key =>
API_KEY, :authtenant => TENANT, :auth_url => API_URL, :service_type =>
"object-store")
Lts2.containers
=>["cities", "countries"]
Cont = Lts2.container("cities")
Cont.objects
=>["London.txt", "Mumbai.txt", "NewYork.txt"]
```

Summary

In this chapter, you learned how to use various Swift clients to interact with Swift clusters and get information on accounts, containers, and objects. You were introduced to ACLs, large object transfers, and also to various Swift client libraries that can be used to write applications in your desired language such as Java, Ruby, and Python.

The next chapter talks about managing Swift and things to consider while replacing or expanding disks, nodes, and zones. It also provides information on various tools that can be used to gather information on the object storage behavior.

For More Information:

**Where to buy this book**


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](http://www.packtpub.com/implementing-cloud-storage-with-openstack-swift/book).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.