SDK To Ease Access Of Openstack Storage Swift Cloud Services

Aniruddha Atre, Prasanna Lalingkar, Amit Rao, Pushkaraj Shingre

Abstract— OpenStack is an open source cloud computing platform which is a joint venture by Rackspace and NASA. OpenStack consists of many projects such as Nova (Compute), Swift (Storage), Glance (VM Images) etc. Swift is used as Object storage. Cloud Data Management Interface (CDMI) is a new standard defined by Storage Network Industry Associate (SNIA). It focuses on inter-operability among various cloud service providers. Currently all cloud services are not CDMI compliant but companies are working on it. CDMI implementation is available which adds support for the CDMI adaptor specification on OpenStack Object Storage (Swift) project. It is used as the base for this project.Project aims at developing a Software Development Kit (SDK) for a novice user which will use CDMI implementation to achieve ease of use for OpenStack cloud services. This project consists of firstly, a library providing CRUD functionalities for CDMI based object storage and secondly an appropriate Graphical User Interface to provide simplicity. Motivation of this project is to fill the gap for use of CDMI implementation over OpenStack Swift by any novice user by developing a .NET SDK for windows audience. Key features for this project include abstraction of complexity, simplicity and fault tolerance.

Index Terms— CDMI, Cloud, Metadata, OpenStack, REST, SNIA, Swift

1 Introduction

OPENSTACK is a software community which provides open source cloud computing platforms for both public and private clouds. OpenStack is an Infrastructure As A Service (IAAS) cloud computing project that is free open source project under terms and conditions of Apache License. OpenStack mission was started jointly by NASA and Rackspace Hosting in July, 2010 by combining code from their Nebula platform and cloud files platform respectively. This project is managed by OpenStack Foundation, a nonprofit corporate entity comprising of more than 180 companies including computer giants such as IBM, Yahoo, Cisco, VMware, Intel, Dell, HP, AMD, SUSE Linux, and Red Hat. This project targets to deliver solutions to all types of clouds and simultaneously providing simple implementation and massive scalability. This is not a single technology but set of interrelated components used to deliver solutions. Basic principles on which OpenStack believes to provide cloud offering include simple, elastic, consistent, scalable services. All the cloud services offered by OpenStack are open source and anybody can contribute to the project. Initially OpenStack operated at 3 month release cycle but later on further stable releases were released at time intervals of 6 months. Following are the release details:

- Austin - Oct, 2010
- Bexar - Feb, 2011
- Cactus - Apr, 2011
- Diablo - Sep, 2011
- Essex – Apr,2012
- Folsom – Sep,2012
- Grizzly, Apr-2013

With every successive release additional components are introduced. Different components of OpenStack project according to Folsom architecture are:

- *All the authors pursued their Bachelor Degree in Computer Engineering from Pune University in 2013 and they are now working with reputed IT companies.*
- *Authors can be reached at following email addresses: aniruddha9atre@gmail.com, pdlalingkar@gmail.com, pushkarajs10@gmail.com, admydam@gmail.com*

2.1 OpenStack Compute (Nova)

OpenStack Compute (Nova) is used to provide and manage large networks of virtual machines. It provides computing power through virtual machine and network management on demand. It gives the software, control panels and APIs required for managing a cloud, including running instances, networks and controlled access through users and projects. Compute architecture is horizontally as well as vertically scalable and supports multi-tenancy and authentication at various levels which are key features of the project.

2.2 OpenStack Image (Glance)

Glance provides discovery, storage and retrieval of virtual machine images which can also be used by Nova. It provides a standard REST interface for querying information about virtual machine images stored in various backend stores including OpenStack Object Storage (Swift).It allows uploading of public as well as private images in many different formats such as Raw, VHD (Hyper-V), VDI (Virtual Box) etc.

2.3 OpenStack Dashboard (Horizon)

Horizon was initially started as app to manage all OpenStack compute project. So initially it comprised of set of views, templates and API calls. As OpenStack projects grew, new projects were added to it and then additional features were included in Horizon as well. Now, Horizon is a Django-based project aimed to provide a complete OpenStack Dashboard along with extensible framework for building new components. It provides web based interface to all OpenStack projects. The extensible design helps to expose third party services such as billing and additional management tools.

2.4 OpenStack Identity (Keystone)

Identity provides Token, Catalog and Policy services for various components of OpenStack. It provides authentication and high level authorization. It currently supports token based authentication.

2.5 OpenStack Networking (Quantum)

This OpenStack project was added in Folsom release. It is a core OpenStack project that provides network connectivity abstraction layer to other OpenStack projects. It is a pluggable, scalable and API driven system for managing IP addresses and eventually networks. Like other cloud operators it can be

used by administrators and users both. It allows static IP's as well as DHCP.

2.6 OpenStack Block Storage (Cinder)

This OpenStack project was added in Folsom release. This was added to separate the existing nova block volume storage into a separate project. As size of nova is growing exponentially it will be difficult in future to manage that large amount of data and add new functionality to them, therefore a new block storage project that is Cinder was started. Cinder provides "Block Storage As A Service". It supports simple Linux server storage and many other storage platforms such as Ceph, NetApp, IBM storage etc.

2.6 OpenStack Object Storage (Swift)

In today's world large amount of data is in unstructured format and cannot be easily stored in block storage as it will result in more wastage of memory. Due to widespread of social networking, amount of unstructured data is increasing by passing day. To store such data, object storage is required. OpenStack Swift is multi-tenant, highly scalable object storage system. All objects in swift have a URL by which it can be accessed. All objects are replicated 3 times in cluster. Each object has its own metadata and can be located anywhere in the distributed cluster. To interact with object RESTful HTTP transaction is necessary and all these advantages of swift make it a very useful object store at low cost. Building blocks of Swift include Proxy Servers, Zones, Rings, Partitions, Accounts, Containers and Objects. Proxy servers are public face of Swift and handle all incoming API requests. They are also responsible for coordinating response, timestamp and handle failures. Ring maps partitions to a specific physical location on disk. Each partition in a ring is replicated thrice which can be used in case of failure. Zones in swift are configured so as to isolate failures. A zone can be a single disk or a large array of disks. As discussed earlier, any object is replicated thrice in Swift. This replication must be achieved in such a way that replicators must be from different zones and not the same zone. Account indicates an individual SQLite database and is distributed across the cluster. An account has many containers and objects within itself. Container is similar to directory in file system and contains many objects. Objects store actual data and they are similar to files in file system. A partition is a collection of object, container, and account databases. It is a core part of replication system. Swift typically runs on Ubuntu machine 10.04 onwards. To install and run swift successfully along with swift following software needs to be installed.

- Python 2.6
- Rsync 3.0

Few python libraries that must be installed are:

- Eventlet 0.9.8
- Setuptools
- Simplejson
- Xattr
- Sphinx
- Netifaces

Swift is typically installed on Ubuntu server edition but can be used on desktop edition also. We have installed it on desktop edition. Swift has its authorization system named tempAuth. One can use tempAuth for authorization or can also OpenStack keystone for authorization. We have used tempAuth for authorization.

2 CDMI

Every cloud service provider provides various services such as data storage, computational power and many more. Although every cloud service provider gives similar functionality, their implementation is bit different than every other provider. This problem affects any end user most, as user is not able to migrate from one provider to other. To address this issue SNIA formulated one pioneering standard called as CDMI. In simple terms, CDMI defines set of rules that are to be followed by every cloud implementation so that migration from one cloud provider to other would be easy. Model behind the Cloud Data Management Interface is as shown in fig. 1, "CDMI Object Model".

Fig.1: CDMI Object Model

CDMI specification tries to use RESTful principles whenever possible to ease the implementation of the interface. A CDMMI compliant system does not need to implement full specification but only a core set of requirements which are mandatory. Being a RESTful protocol, CDMI client system communicates with CDMI server using basic REST methods: GET, PUT, DELETE and POST. CDMI server responds back with corresponding Http response.

2.1 Metadata

Key aspect to the simplicity of cloud is in use of metadata. To meet this, CDMI specification uses rich metadata capabilities. CDMI defines several types of metadata. Metadata is simply data about data. In object storage system, data stored is unstructured which makes handling of data more difficult. As data is not consistent, it cannot be stored in blocks and as data is generally accessed many times, it is a major challenge to improve and maintain efficiency of system. So concept of metadata is extensively applied in order to achieve efficiency. Metadata means storing information about data such as time of creation, time last modified and type of file and many such parameters. It helps system to improve search efficiency and handle data more carefully. There are various types of metadata in any system such as user defined metadata, system metadata.

(1)User Defined Metadata: CDMI allows users to add or attach metadata to their objects. It is in the form of key: value pairs that can be provided in PUT CDMI commands of container as well as data object.

(2)Data System Metadata: Data system metadata attributes influence the system's behavior with respect to resource for

which data system metadata was attached. Data system metadata attribute can be specified by user per object or container. These attributes are inherited. That is if the user does not specify the attribute for any resource then attribute is inherited from its parent.

(3)Storage System Metadata: Storage system is allowed to generate storage system metadata for any resource in CDMI specification. For example, “cdmi_size” attribute indicates size of the resource in bytes.

2.2 CDMI Objects

CDMI also provides description for performing various operations on any data element. Data elements are divided into five main object classes:

(1)Data Object: serves as an abstraction for a file or blob.

(2)Queue Object: serves as an abstraction for message queue.

(3)Container Object: corresponds to folders in a file system. It can be used to encapsulate objects.

(4) Domain Object: represents administrative ownership of data.

(5)Capability Object: represents the functionality supported by storage system.

3 Need of .Net Sdk for Cdmi Implementation Over OpenStack Swift

There are many API’s and SDK’s are available for Java and many other platforms but no such SDK exists for .NET. Many companies are building their complex applications over OpenStack Swift and they are not easily available to use for an end user due to complications in use. To use such applications, user must be having knowledge of CDMI and OpenStack Swift as well and this is not possible for each and every user. Due to this all benefits of OpenStack Swift has not reached to the windows audience. This SDK aims to fill this gap and reduce the complexity.

4 Working of Sdk

This SDK aims to abstract the complexity and make it an easy task for any end user to communicate with CDMI implementation over OpenStack Swift server. The working of this SDK is shown in fig. 2,”Work-flow of SDK”.

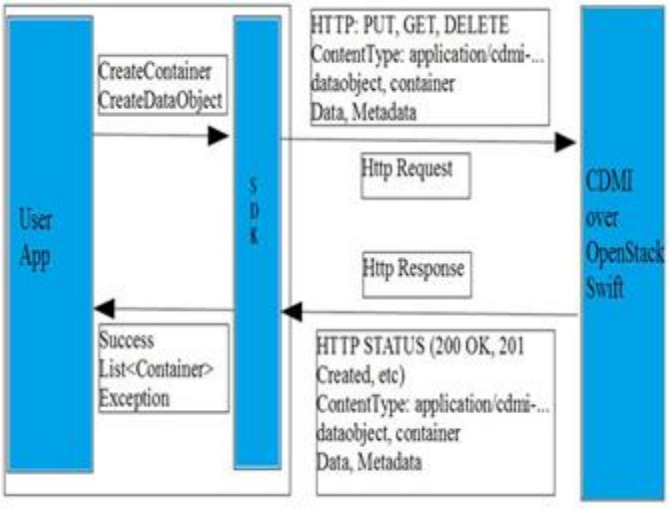

Fig.2: Work-flow of SDK

When a user requests to create a container, a web request is to be sent to CDMI implementation over OpenStack Swift server with all the required headers, authentication token and followed by URI. But by using this SDK, user now has to just enter name of container or data object to be created and all other things are taken care of by our SDK. Whenever a create container or create object request comes to SDK, first it checks for authentication, if authenticated further procedure is implemented otherwise not. Now once authentication token is validated, a PUT request is sent over HTTP with all necessary headers and URI. Response returned from CDMI implementation over OpenStack Swift is a complex JSON string. This JSON string is then parsed and only necessary information is shown to the user. In this way, all the complexity is abstracted from user. All necessary functionalities supported by CDMI implementation over OpenStack Swift are provided in this SDK. User defined metadata functionality is also handled. Now, by use of this SDK any novice user can access CDMI implementation over OpenStack Swift without having any knowledge about it. Many complex applications can be built for CDMI implementation over OpenStack Swift using this SDK without much effort.

5 Components of Sdk

This SDK is divided into 3 layers to provide perfect abstraction and simplicity while working with the SDK. Modules in this SDK are divided as client modules, data layer modules and utility modules. Structure of SDK is shown in fig. 3,”Components of SDK”.

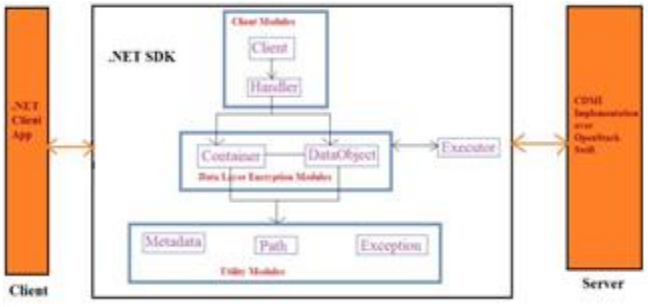

Fig.3: Components of SDK

Client and handler modules are intended to authenticate a user and provide flexibility to use underlying modules and functionalities. Data layer modules represent CRUD functionalities for container and data object. Utility modules are used to add functionality of metadata, exception handling and JSON parsing.

6 Software Development

In the process of development of SDK, incremental software development cycle is used. Initially basic implementation of the system is designed and modeled. Later new functionalities were added to existing model. All added functionalities contributed to more simplicity in user interaction and helped to abstract complexity. .

7 Qualitative Parameters

7.1 Authentication Migration

CDMI implementation over OpenStack Swift uses tempAuth as inbuilt authentication system. Therefore, this SDK is designed to work with tempAuth system. Now, if authentication system is changed to OpenStack keystone or OpenAuth then few changes are needed in this SDK's authentication mechanism. But overall structure of SDK remains same. This means that in future, this SDK can be used for various different types of implementations for CDMI implementation over OpenStack Swift.

7.2 Simplicity

User interface provided in this SDK is very simple to understand and to use for any naïve user. Therefore, CDMI implementation over OpenStack Swift can be easily reached out to all end users in windows background.

7.3 Abstraction of Complexity

All complexity of OpenStack Swift is abstracted from user. All users are shown is a very basic things they must be aware of that is, container or object name, password, user name.

7.4 Fault Tolerance

As exception handling at various levels is taken care of by this SDK, system is able to catch all exceptions which may arise. All exception handling is done as a separate utility part of SDK therefore it does not affect overall performance of the system as well.

7.5 Performance Penalties

Complete SDK is written in such a way that no performance penalties will occur. As it uses simple HTTP request response session to communicate with CDMI implementation over OpenStack Swift server, only reason for performance degradation is due to network problems such as congestion in network, packet loss, server not reachable etc.

8 Conclusion

CDMI implementation over OpenStack Swift is an open source object store and it can be made simpler to use by using this SDK for windows users. This SDK can also be used to develop similar SDK for any other authentication system. Also many complex applications can be built by using this SDK.

References

[1] SNIA, Information Technology - Cloud Data Management Interface (CDMI.) Version 1.0.1, Sept 2011, http://snia.org/sites/default/files/CDMI_SNIA_Architecture_v1.0.1.pdf

[2] SNIA, Information Technology - Cloud Data Management Interface (CDMI.) Version 1.0.2, June 2012, http://snia.org/sites/default/files/CDMI%20v1.0.2.pdf