How to enable cross functionality testing for multiple cloud storage APIs such as CDMI, OpenStack Swift and Amazon S3

Ankit Agrawal
SPE Group, Hitech ISU
TCS

31 May 2016
Focal Points of Discussion

01. Object Storage: Overview
02. Major Object Storage APIs
03. Products supporting multiple Object APIs
04. Why Interoperability is critical??
05. Testing - Challenges and Best Practices
06. Object APIs Cross Functionality Testing - Test Cases
Object Storage: Overview

Unstructured Data Growth

- IDG: Unstructured data is growing at the rate of 62% per year.
- IDG: By 2022, 93% of all data in the digital universe was unstructured.
- Gartner: Data volume is set to grow 800% over the next 5 years and 80% of it will reside as unstructured data.

Object Storage

- A proven option for effectively managing unstructured data.
- Storage architecture that manages data as objects
- A data container capable of storing files and metadata about the files, which consists of the attributes for the actual data being stored.
Object Storage: Overview

**Characteristics**
- Manage data in form of Container & Data Object
- Flat address space
- Unique ObjectID
- HTTP/REST/SOAP
- CRUD

**Advantages**
- Security and reliability
- Platform independence
- Scalability
- Manageability
- suitable for cloud environment
Major Object Storage APIs

- First publicly available web service
- Introduced and originally offered by Amazon Web Services
- Abbreviation for Simple Storage Service.
- Supports REST, SOAP, and BitTorrent web services interfaces
- **Bucket** - fundamental container for data storage
- **Object**
  - upto 5 TB in size, up to 2 KB of Metadata
  - **Key**
    - used to identify object
    - Unique within each bucket
    - user-assigned
    - Unicode characters (UTF-8 encoding length 1024 bytes)
  - **version ID**
    - Used for Object Versioning
    - S3 generates a unique version ID and assigns it to the object
- Requests are authorized using an access control list associated with each bucket and object.

* All Logos/Images are copyrighted to their respective companies.
Major Object Storage APIs

OpenStack Swift:
- OpenStack Object Store project
- OpenSource, Highly Scalable, Highly Available & Production ready project
- Supports REST web services interfaces
- Containers - Provide name space where object stored
- Object
  - fundamental unit for data storage
  - Support size upto 5Gb
- Pseudo-Hierarchical Directories
  - Doesn’t support nested Containers
  - To manage huge number of object pseudo- hierarchical directories can be created.
- Object Versioning
  - Each PUT request to an object will result in the existing object being archived to a special “versions” container.
- Authentication
  - OpenStack Keystone
  - Flexible enough to integrate with other authentication mechanism
Major Object Storage APIs

SNIA CDMI:
- Introduced by SNIA (Storage Networking Industry Association)
- Abbreviation for Cloud Data Management Interface
- CDMI Specification defines functional interface for object storage that application can use to create, retrieve, update and delete data elements from the Cloud
- Foster Interoperability to avoid Vendor Lock-in
- An open international (ISO) standard
- Provide support for REST APIs

Data Management:
- Data is stored and managed using Containers and Data Objects.
- Data Objects are identified by user assigned name and system assigned Object ID.
- Object IDs are globally unique and native format of an object ID is a variable length byte sequence and shall be a maximum length of 40 bytes
- Supports Capability object that used to discover cloud storage offerings and functionalities.

Authentication:
- Relies on an authentication service (local or external) to validate client credentials.
- Supports following authentication methods
  - Anonymous/Basic/Digest/Kerberos
  - Certificate-based authentication via TLS
  - S3 API signed header authentication
  - OpenStack Identity API header authentication

* All Logos/Images are copyrighted to their respective companies.
Why Interoperability is critical??

01. Data Lock-in
02. Seamless Adoptability
03. Focus on enterprise level feature
04. Market Pace
05. Agility and flexibility
# Testing - Challenges and Best Practices

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Best Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does product support interoperability among different object APIs</td>
<td>Check for any other alternative or workaround in which Object APIs can be interoperable. Otherwise, this solution is not applicable for such products.</td>
</tr>
<tr>
<td>Different Object API supports different authentication method.</td>
<td>Prepare reusable components/methods for each object API authentication scheme. Check for any common authentication supported by Product.</td>
</tr>
<tr>
<td>Different Object API supports different size for large objects e.g. 5TB for S3 and 5GB for OpenStack Swift</td>
<td>Check for any alternative option/support for larger object size e.g. in OpenStack Swift larger object than 5 GB are supported using Segment/Manifest object. Check for these alternatives support in Product, to be tested.</td>
</tr>
</tbody>
</table>
### Testing - Challenges and Best Practices (Contd..)

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Best Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same feature but Different implementation: Object Versioning</td>
<td>Needs to be tested very carefully, to check if request is being diverted properly to correct container/bucket url in order to access particular version.</td>
</tr>
<tr>
<td>Object Versioning - In OpenStack Swift, older copies of Objects are kept in a particular container, while in S3 all versions are kept in same bucket only, with new versionID created on each update.</td>
<td>CDMI allows nested containers to be created, but S3/Swift doesn't.</td>
</tr>
<tr>
<td>Unique Feature - Nested Containers</td>
<td>Nested Containers to be mapped with Pseudo-Hierarchical directory</td>
</tr>
</tbody>
</table>
Object APIs Cross Functionality Testing - Test Cases

Test Case Classification (Based on complexity)

- **Simple Test Cases**
  - Covers CRUD operations only

- **Medium Test Cases**
  - Covers medium level features such as CRUD with Metadata, Object Versioning etc.

- **Complex Test Cases**
  - Covers complex features such as Retention policies, large object etc.
Simple Test Cases

CDMI and OpenStack Swift APIs Cross Functionality testing

**Figure: 1**

**Test Case#1:**
Create a container using Swift APIs and Retrieve through CDMI

**Test Case#2:**
Create a container using Swift APIs and Delete through CDMI

**Test Case#3:**
Create a container using Swift APIs and Update through CDMI

**Figure: 2**

**Test Case#4:**
Create a container using CDMI APIs and Retrieve through Swift

**Test Case#5:**
Create a container using CDMI APIs and Update through Swift

**Test Case#6:**
Create a container using CDMI APIs and Delete through Swift
Simple Test Cases

CDMI and OpenStack Swift APIs Cross Functionality testing

**Figure: 3**

**Test Case #7:**
Create a Container/Data Object using Swift APIs and Retrieve through CDMI

**Test Case #8:**
Create a Container/Data Object using Swift APIs and Update through CDMI

**Test Case #9:**
Create a Container/Data Object using Swift APIs and Delete through CDMI

**Figure: 4**

**Test Case #10:**
Create a Container/Data Object using CDMI APIs and Retrieve through Swift

**Test Case #11:**
Create a Container/Data Object using CDMI APIs and Update through Swift

**Test Case #12:**
Create a Container/Data Object using CDMI APIs and Delete through Swift
Simple Test Cases

CDMI and OpenStack Swift APIs Cross Functionality testing

**Test Case#13:**
Create a Container using CDMI APIs and create Data object within CDMI Container through Swift

**Test Case#14:**
Create a Container using Swift APIs and create Data object within Swift Container through CDMI

**Test Case#15:**
Create a Container (Swift API)/Data Object (CDMI) and read Data object through Swift API
Sample Test Cases

Description

<Test Case : Start>

- Generate and save authentication token “SWIFT_AUTH_TOKEN” for Swift using OpenStack Keystone or supported authentication service.
  
```
curl --X POST --H "Content-Type: application/json" --d '{"auth": {"tenantName": "XXX", "passwordCredentials": {"username": "XXX", "password": "XXX"}}}" AUTH_URL
```

- Create a container named "TestContainer1" using swift API
  
```
curl --X PUT --H "Content-Length: 0" --H "X-Auth-Token: $SWIFT_AUTH_TOKEN" $SWIFT_URL/TestContainer1
```

- Verify if container created successfully using swift API:
  
```
curl --X GET --H "X-Auth-Token: $SWIFT_AUTH_TOKEN" $SWIFT_URL/TestContainer1
```

Check for HTTP status code: 200 OK returned.

- Set valid login credentials for CDMI request.

- Retrieve Container created in above step using CDMI API:
  
```
GET <CDMI_URL>/TestContainer1 HTTP/1.1
Host: cloud.example.com
Accept: application/cdmi-container
X-CDMI-Specification-Version: 1.0.2
```

- Verify response code:
  
```
Check for HTTP status code: 200 OK returned.
```

- Expected Result: Container "TestContainer1" should be created and retrieved successfully.

Cleanup created container

<Test Case : End>

Test Case Name

TestCase#1

Test Case Description

Create a Container using Swift API and Retrieve the same through CDMI

Pre-Test Dependencies

- Product must support interoperability between OpenStack Swift and CDMI APIs
- Swift Service End-Point (SWIFT_URL)
- CDMI Service End-Point (CDMI_URL)
Sample Test Cases

Test Case Name

Test Case#2

Test Case Description

Create a Container using Swift API and Delete the same through CDMI

Pre-Test Dependencies

- Product must support interoperability between OpenStack Swift and CDMI APIs
- Swift Service End-Point (SWIFT_URL)
- CDMI Service End-Point (CDMI_URL)

<Test Case : Start>

- Generate and save authentication token “SWIFT_AUTH_TOKEN” for Swift using OpenStack Keystone or supported authentication service.
  
curl -X POST -H "Content-Type: application/json" -d '{"auth": {"tenantName": "XXX", "passwordCredentials": {"username": "XXX", "password": "XXX"}}}’ AUTH_URL

- Create a container named "TestContainer1" using swift API
  
curl -X PUT -H "Content-Length: 0" -H "X-Auth-Token: $SWIFT_AUTH_TOKEN" $SWIFT_URL/TestContainer1

- Verify if container created successfully using swift API:
  
curl -X GET -H "X-Auth-Token: $SWIFT_AUTH_TOKEN" $SWIFT_URL/TestContainer1
  Check for HTTP status code: 200 OK returned

- Set valid login credentials for CDMI request.

- Delete Container, created in above step, using CDMI API:
  
  DELETE <CDMI_URL>/TestContainer1/ HTTP/1.1
  Host: cloud.example.com

  X-CDMI-Specification-Version: 1.0.2

- Verify response code:
  Check for HTTP status code: 204 No Content returned.

- Expected Result: Container "TestContainer1" should be created and deleted successfully.

<Test Case : End>

Description

TestCase#2

Test Case Name

Create a Container using Swift API and Delete the same through CDMI

Pre-Test Dependencies

- Product must support interoperability between OpenStack Swift and CDMI APIs
- Swift Service End-Point (SWIFT_URL)
- CDMI Service End-Point (CDMI_URL)
References

https://en.wikipedia.org/wiki/Amazon_S3

http://www.scality.com/ring/object-storage-overview/


Thank You