How to ensure OpenStack Swift & Amazon S3 Conformance for storage products & services supporting multiple Object APIs

Ankit Agrawal
Tata Consultancy Services Ltd.

30 May 2017
Focal Points of Discussion

1. Object Storage: Overview
2. Object Storage APIs: Overview
3. Conformance Testing Approach
4. Sample Test Cases
Unstructured Data Growth

- What is Unstructured Data?
- Why Unstructured Data is growing massively?
- Unstructured Data Growth Report
- Why Unstructured Data is so important?
Object Storage: Overview (1/2)

Why Object Storage for Unstructured Data

- Limitless Scalability
- Runs on Commodity Hardware
- Highly Available
- Anytime / Anywhere access
- Flat address space
- Unique ObjectID
- Manageability

REST API
Get/Post/Put/Delete

Data

Object Storage
Object Storage APIs: Overview

- Object Storage APIs?
- Why Amazon S3 & OpenStack Swift?
- Why Conformance to S3 & Swift is critical?
Conformance Testing Approach

1. OpenStack Swift

2. Amazon S3
Conformance Testing Approach

- Supports the REST API
- Supports Token Based Authentication
**Conformance Testing Approach**

**Discoverability Operations**
- GET /info
- lists the activated capabilities

**Endpoints Operations:**
- GET /v1/endpoints
- List endpoints

**Operations on the Accounts**
- Show account details and list containers
  - GET /v1/{account}
- Create, update, or delete account metadata
  - POST /v1/{account}
- Show account metadata
  - HEAD /v1/{account}
**Operations on the Containers**

- **Show container details and list objects**
  - GET /v1/{account}/{container}

- **Create container**
  - PUT /v1/{account}/{container}

- **Create, update, or delete container metadata**
  - POST /v1/{account}/{container}

- **Show container metadata**
  - HEAD /v1/{account}/{container}

- **Delete container**
  - DELETE /v1/{account}/{container}

**Operations on the Objects**

- **Get object content and metadata**
  - GET /v1/{account}/{container}/{object}

- **Create or replace object**
  - PUT /v1/{account}/{container}/{object}

- **Copy object**
  - COPY /v1/{account}/{container}/{object}

- **Delete object**
  - DELETE /v1/{account}/{container}/{object}

- **Show object metadata**
  - HEAD /v1/{account}/{container}/{object}

- **Create or update object metadata**
  - POST /v1/{account}/{container}/{object}
Test Cases:
OpenStack Swift APIs - Container Operations

**Test Case#1:**
Show container details and list objects

**Test Case#2:**
Show container details and list objects for container that does not exist

**Test Case#3:**
Create a Container using Swift API

**Test Case#4:**
Create a Container using custom metadata

**Test Case#5:**
Delete container metadata

**Test Case#6:**
Show container metadata

**Test Case#7:**
Create a container with an ACL to allow anybody to get an object in the particular container

**Test Case#8:**
Delete an empty Container

**Test Case#9:**
Delete a Container that does not exist.

**Test Case#10:**
Delete a non-empty Container

---

**Figure: 1**

- **POST/PUT/GET/DELETE**
- Container
- Storage
Sample Test Cases

Test Cases:
OpenStack Swift APIs - Object Operations

Object

Test Case#1:
Show object details for the particular object in the particular container

Test Case#2:
Show object details for the object, which does not exist, in the particular container

Test Case#3:
Create object using Swift API

Test Case#4:
Update existing Object.

Test Case#5:
Copy existing object from one container to other

Test Case#6:
Create object metadata

Test Case#7:
Show object metadata

Test Case#8:
Update object metadata

Test Case#9:
Copy non-existing object from one container to other

Test Case#10:
Delete existing object from the particular container

Test Case#11:
Delete non-existing object from the particular container

Test Case#12:
Delete static large object (segments & manifest object)

Figure: 2
Conformance Testing Approach

1. OpenStack Swift

2. Amazon S3
Conformance Testing Approach

- Current Version: 2006-03-01
- Supports the REST APIs
- Authentication - AWS Signature Version 4 Algorithm
- Authentication Methods
  - HTTP Authorization header
  - Query string parameters
Conformance Testing Approach

**Common Request Headers**
- Authorization
- Content-Length
- Content-Type
- Content-MD5
- Date
- Expect
- Host
- x-amz-content-sha256
- x-amz-date
- x-amz-security-token

**Common Response Headers**
- Content-Length
- Content-Type
- Connection
- Date
- Etag
- Server
- x-amz-delete-marker
- x-amz-id-2
- x-amz-request-id
- x-amz-version-id
GET Service:
- Returns a list of all buckets owned by the authenticated sender of the request.
- URI: GET /
### Operations on the Buckets (Create/Update)

**PUT Bucket**
- Creates a new bucket

**PUT Bucket accelerate**
- Set the Transfer Acceleration state of an existing bucket to enable to perform faster data transfers

**PUT Bucket acl**
- To set the permissions on an existing bucket using access control lists (ACL)

**PUT Bucket inventory**
- Adds an inventory configuration (identified by the inventory ID) to the bucket.

**PUT Bucket cors**
- Sets the cors configuration for your bucket

### Operations on the Buckets (Retrieve)

**GET Bucket (List Objects)**
- Returns some or all (up to 1,000) of the objects in a bucket.

**GET Bucket accelerate**
- Returns the Transfer Acceleration state of a bucket, which is either Enabled or Suspended.

**GET Bucket acl**
- Returns the access control list (ACL) of a bucket.

**GET Bucket inventory**
- Returns an inventory configuration (identified by the inventory configuration ID) from the bucket.

**GET Bucket cors**
- Returns the cors configuration information set for the bucket.
Conformance Testing Approach

Operations on the Buckets (Delete)

DELETE Bucket
- deletes the bucket named in the URI.

DELETE Bucket inventory
- deletes an inventory configuration (identified by the inventory configuration ID) from the bucket

DELETE Bucket cors
- Deletes the cors configuration information set for the bucket.
## Conformance Testing Approach

### Operations on Objects (Create)

**PUT Object**
- adds an object to a bucket.

**PUT Object - Copy**
- creates a copy of an object that is already stored

**PUT Object acl**
- Uses the acl subresource to set the access control list (ACL) permissions for an object that already exists in a bucket.

**PUT Object tagging**
- uses the tagging subresource to add a set of tags to an existing object.

### Operations on Objects (Retrieve)

**GET Object**
- retrieves objects from Amazon S3.

**GET Object ACL**
- uses the acl subresource to return the access control list (ACL) of an object.

**GET Object tagging**
- returns the tags associated with an object.

**GET Object torrent**
- uses the torrent subresource to return torrent files from a bucket.
## Conformance Testing Approach

### Operations on Objects (Delete)

**Delete Multiple Objects**
- delete multiple objects from a bucket using a single HTTP request.

**DELETE Object**
- removes the null version (if there is one) of an object
- If versioning enabled, permanently deletes the version

**DELETE Object tagging**
- uses the tagging subresource to remove the entire tag set from the specified object.

### Operations on Objects (Others)

**HEAD Object**
- retrieves metadata from an object without returning the object itself.
- retrieve metadata from a different version, use the versionId subresource.

**OPTIONS Object**
- A browser can send this preflight request to Amazon S3 to determine if it can send an actual request with the specific origin, HTTP method, and headers.
Sample Test Cases

<Test Case : Start>

- Compute and save authentication signature in “AUTH_SIGNATURE” variable using Secret Access Key and AWS Signature Version 4 Algorithm.
- Create a bucket named "TestBucket1" using Amazon S3 API
  
  ```
  PUT / HTTP/1.1
  Host: TestBucket1.cloud.example.com
  Content-Length: 0
  Date: Wed, 01 Mar 2006 12:00:00 GMT
  Authorization: AUTH_SIGNATURE
  ```

- Verify if bucket “TestBucket1” created successfully:
  - Check for HTTP status code: 200 OK returned
  - Location header should be: /TestBucket1
  - x-amz-id-2 and x-amz-request-id should be returned
- “GET /TestBucket1” should run successfully.
- Expected Result: Bucket "TestBucket1" should be created successfully.

<Test Case : End>

Description

<table>
<thead>
<tr>
<th>Test Case Name</th>
<th>Test Case Description</th>
<th>Pre-Test Dependencies</th>
</tr>
</thead>
</table>
| TestCase#1     | Create a new Bucket using Amazon S3 compatible APIs | Secret Access Key for Authentication
|                |                       | Object Storage End-Point (cloud.example.com) |
Sample Test Cases

**Test Case Name**

Test Case #2

**Test Case Description**

List Objects contained in bucket “TestBucket1” successfully.

**Pre-Test Dependencies**

- Secret Access Key for Authentication
- Object Storage End-Point (cloud.example.com)

**Description**

*<Test Case : Start>*

- Compute and save authentication signature in “AUTH_SIGNATURE” variable using Secret Access Key and AWS Signature Version 4 Algorithm.
- Create bucket TestBucket1 <<Refer: TestScript#1>> and add objects to it.
- List all objects contained in bucket “TestBucket1”, using Amazon S3 API
  
  **GET /?list-type=2 HTTP/1.1**
  **Host: TestBucket1.cloud.example.com**
  **x-amz-date: 20160430T233541Z**
  **Authorization: AUTH_SIGNATURE**
  **Content-Type: text/plain**

- Verify if bucket “GET /TestBucket1” executed successfully:
  - Check for **HTTP status code: 200 OK** returned
  - Response Body should list all objects contained in TestBucket1
- Expected Result: All objects contained in bucket “TestBucket1” should be listed successfully.
- Clean-up: Delete bucket “TestBucket1”

*<Test Case : End>*
Sample Test Cases

**Test Case Name**

Test Case#3

**Test Case Description**

Create a new bucket and configure access permission using a canned ACL

**Pre-Test Dependencies**

- Secret Access Key#1 for Account#1
- Secret Access Key#2 for Account#2
- Object Storage End-Point (cloud.example.com)

---

**Description**

**<Test Case : Start>**

- Compute and save authentication signature in “AUTH_SIGNATURE” variable using Secret Access Key#1 and AWS Signature Version 4 Algorithm.

- Create a bucket named “TestBucket1” using Amazon S3 API
  
  ```
  PUT / HTTP/1.1
  Host: TestBucket1.cloud.example.com
  Content-Length: 0
  x-amz-acl: private
  Date: Wed, 01 Mar 2006 12:00:00 GMT
  Authorization: AUTH_SIGNATURE
  ```

- Verify if bucket “TestBucket1” created successfully using Amazon S3 API:
  - Check for HTTP status code: 200 OK returned
  - Location header should be: /TestBucket1
  - x-amz-id-2 and x-amz-request-id must be returned

- Try to read bucket “GET /TestBucket1” using Access Key#2 (Refer: TestCase#2), it should return Error Code AccessDenied (403 Forbidden)

- Expected Result: Bucket “TestBucket1” should be created successfully.

- Clean-up: Delete bucket “TestBucket1”

**<Test Case : End>**
Sample Test Cases

**Test Case Name**

TestCase#4

**Test Case Description**

Delete an existing bucket

**Pre-Test Dependencies**

- Secret Access Key for Authentication
- Object Storage End-Point (cloud.example.com)

**Description**

<Test Case : Start>

- Compute and save authentication signature in “AUTH_SIGNATURE” variable using Secret Access Key and AWS Signature Version 4 Algorithm.
- Create bucket TestBucket1 <<Refer: TestScript#1>>
- Delete bucket named “TestBucket1“ using Amazon S3 API
  `DELETE / HTTP/1.1`
  `Host: TestBucket1.cloud.example.com`
  `Date: Wed, 01 Mar 2006 12:00:00 GMT`
  `Authorization: AUTH Signature`
- Verify if bucket “TestBucket1” deleted successfully:
  Check for HTTP status code: **204 No Content** returned
  x-amz-id-2 and x-amz-request-id must be returned
- Try to read bucket “GET /TestBucket1”, it should return Error Code NoSuchBucket (404 Not Found)
- Expected Result: Bucket "TestBucket1" should be deleted successfully.

<Test Case : End>
Sample Test Cases

**Test Case Name**

Test Case#5

**Test Case Description**

Create a new Bucket using Amazon S3 compatible APIs

**Pre-Test Dependencies**

- Secret Access Key for Authentication
- Object Storage End-Point (cloud.example.com)

**Description**

<Test Case : Start>

- Compute and save authentication signature in "AUTH_SIGNATURE" variable using Secret Access Key and AWS Signature Version 4 Algorithm.
- Create a bucket named "TestBucket1" using Amazon S3 API
  
  ```
  PUT / HTTP/1.1
  Host: TestBucket1.cloud.example.com
  Content-Length: 0
  Date: Wed, 01 Mar 2006 12:00:00 GMT
  Authorization: AUTH_SIGNATURE
  ```

- Verify if bucket "TestBucket1" created successfully:
  - Check for HTTP status code: 200 OK returned
  - Location header should be: /TestBucket1
  - x-amz-id-2 and x-amz-request-id should be returned
  - “GET /TestBucket1” should run successfully.
  - Expected Result: Bucket "TestBucket1" should be created successfully.

<Test Case : End>
Questions?
Thank You

Email: ankit29.a@tcs.com