Red Hat Ceph Storage 3

Developer Guide

Using the various application programming interfaces for Red Hat Ceph Storage

Last Updated: 2021-05-05

Red Hat Ceph Storage 3 Developer Guide

Using the various application programming interfaces for Red Hat Ceph Storage

Legal Notice

Abstract

This document provides instructions for Using the various application programming interfaces for Red Hat Ceph Storage running on AMD64 and Intel 64 architectures.

Table of Contents

CHAPTER 1. OBJECT GATEWAY ADMINISTRATION APPLICATION PROGRAMMING INTERFACE (API)

The Ceph Object Gateway exposes features of the **radosgw-admin** command-line interface in a RESTful API too. Red Hat recommends using the command-line interface when setting up the Ceph Object Gateway. When you want to manage users, data, quotas and usage, the Ceph Object Gateway's administrative API provides a RESTful interface that you can integrate with other management platforms. The administrative API provides the following functionality:

- Authentication Requests
- **User/Subuser Account Management**
  - Administrative User
  - Getting User Information
  - Creating
  - Modifying
  - Removing
  - Creating Subuser
  - Modifying Subuser
  - Removing Subuser
- **User Capabilities Management**
  - Adding
  - Removing
- **Key Management**
  - Creating
  - Removing
- **Bucket Management**
  - Getting Bucket Information
  - Checking Index
  - Removing
  - Linking
  - Unlinking
  - Policy
- **Object Management**
  - Removing

1.1. AUTHENTICATING REQUESTS

Amazon's S3 service uses the access key and a hash of the request header and the secret key to authenticate the request, which has the benefit of providing an authenticated request (especially large uploads) without SSL overhead.

Most use cases for the S3 API involve using open source S3 clients such as the **AmazonS3Client** in the Amazon SDK for Java or Python Boto. These libraries do not support the Ceph Object Gateway Admin API. You can subclass and extend these libraries to support the Ceph Admin API. Alternatively, you can create a unique Gateway client.

The CephAdminAPI example class in this section illustrates how to create an **execute()** method that can take request parameters, authenticate the request, call the Ceph Admin API and receive a response. **The CephAdminAPI class example is not supported or intended for commercial use. It is for illustrative purposes only.** The client code contains five calls to the Ceph Object Gateway to demonstrate CRUD operations:

- Create a User
- Get a User
- Modify a User
- Create a Subuser
- Delete a User

To use this example, get the **httpcomponents-client-4.5.3** Apache HTTP components. You can download it for example here: http://www.eu.apache.org/dist/httpcomponents/httpclient/binary/. Then unzip the tar file, navigate to its **lib** directory and copy the contents to the **/jre/lib/ext** directory of the **JAVA_HOME** directory, or a custom classpath.

As you examine the CephAdminAPI class example, notice that the **execute()** method takes an HTTP method, a request path, an optional subresource, **null** if not specified, and a map of parameters. To execute with subresources, for example, **subuser**, and **key**, you will need to specify the subresource as an argument in the **execute()** method.

The example method:

1. Builds a URI.

2. Builds an HTTP header string.
3. Instantiates an HTTP request, for example, **PUT**, **POST**, **GET**, **DELETE**.
4. Adds the **Date** header to the HTTP header string and the request header.
5. Adds the **Authorization** header to the HTTP request header.
6. Instantiates an HTTP client and passes it the instantiated HTTP request.
7. Makes a request.
8. Returns a response.

Building the header string is the portion of the process that involves Amazon's S3 authentication procedure. Specifically, the example method does the following:

1. Adds a request type, for example, **PUT**, **POST**, **GET**, **DELETE**.
2. Adds the date.
3. Adds the requestPath.

The request type should be upper case with no leading or trailing white space. If you do not trim white space, authentication will fail. The date MUST be expressed in GMT, or authentication will fail.

The exemplary method does not have any other headers. The Amazon S3 authentication procedure sorts **x-amz** headers lexicographically. So if you are adding **x-amz** headers, be sure to add them lexicographically. See S3 Authentication in this guide for additional details. For a more extensive explanation of the Amazon S3 authentication procedure, consult the Signing and Authenticating REST Requests section of Amazon Simple Storage Service documentation.

Once you have built the header string, the next step is to instantiate an HTTP request and pass it the URI. The examplary method uses **PUT** for creating a user and subuser, **GET** for getting a user, **POST** for modifying a user and **DELETE** for deleting a user.

Once you instantiate a request, add the **Date** header followed by the **Authorization** header. Amazon's S3 authentication uses the standard **Authorization** header, and has the following structure:

```
Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

The CephAdminAPI example class has a **base64Sha1Hmac()** method, which takes the header string and the secret key for the admin user, and returns a SHA1 HMAC as a base-64 encoded string. Each **execute()** call will invoke the same line of code to build the **Authorization** header:

```
httpRequest.addHeader("Authorization", "AWS " + this.getAccessKey() + ":" +
base64Sha1Hmac(headerString.toString(), this.getSecretKey()));
```

The following **CephAdminAPI** example class requires you to pass the access key, secret key and an endpoint to the constructor. The class provides accessor methods to change them at runtime.

```
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.time.OffsetDateTime;
import java.time.format.DateTimeFormatter;
```

```
import java.time.ZoneId;

import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.Header;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.apache.http.client.utils.URIBuilder;

import java.util.Base64;
import java.util.Base64.Encoder;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Mac;

import java.util.Map;
import java.util.Iterator;
import java.util.Set;
import java.util.Map.Entry;

public class CephAdminAPI {

 /*
  * Each call must specify an access key, secret key, endpoint and format.
  */
 String accessKey;
 String secretKey;
 String endpoint;
 String scheme = "http"; //http only.
 int port = 80;

 /*
  * A constructor that takes an access key, secret key, endpoint and format.
  */
 public CephAdminAPI(String accessKey, String secretKey, String endpoint){
  this.accessKey = accessKey;
  this.secretKey = secretKey;
  this.endpoint = endpoint;
 }

 /*
  * Accessor methods for access key, secret key, endpoint and format.
  */
 public String getEndpoint(){
  return this.endpoint;
 }
```

```
public void setEndpoint(String endpoint){
 this.endpoint = endpoint;
}

public String getAccessKey(){
 return this.accessKey;
}

public void setAccessKey(String accessKey){
 this.accessKey = accessKey;
}

public String getSecretKey(){
 return this.secretKey;
}

public void setSecretKey(String secretKey){
 this.secretKey = secretKey;
}

/*
 * Takes an HTTP Method, a resource and a map of arguments and
 * returns a CloseableHTTPResponse.
 */
public CloseableHttpResponse execute(String HTTPMethod, String resource,
                                     String subresource, Map arguments) {

 String httpMethod = HTTPMethod;
 String requestPath = resource;
 StringBuffer request = new StringBuffer();
 StringBuffer headerString = new StringBuffer();
 HttpRequestBase httpRequest;
 CloseableHttpClient httpclient;
 URI uri;
 CloseableHttpResponse httpResponse = null;

 try {

  uri = new URIBuilder()
   .setScheme(this.scheme)
   .setHost(this.getEndpoint())
   .setPath(requestPath)
   .setPort(this.port)
   .build();


  if (subresource != null){
   uri = new URIBuilder(uri)
    .setCustomQuery(subresource)
    .build();
  }


  for (Iterator iter = arguments.entrySet().iterator();
  iter.hasNext();) {
```

```
  Entry entry = (Entry)iter.next();
  uri = new URIBuilder(uri)
   .setParameter(entry.getKey().toString(),
                 entry.getValue().toString())
   .build();

 }

 request.append(uri);

 headerString.append(HTTPMethod.toUpperCase().trim() + "\n\n\n");

 OffsetDateTime dateTime = OffsetDateTime.now(ZoneId.of("GMT"));
 DateTimeFormatter formatter = DateTimeFormatter.RFC_1123_DATE_TIME;
 String date = dateTime.format(formatter);

 headerString.append(date + "\n");
 headerString.append(requestPath);

 if (HTTPMethod.equalsIgnoreCase("PUT")){
  httpRequest = new HttpPut(uri);
 } else if (HTTPMethod.equalsIgnoreCase("POST")){
  httpRequest = new HttpPost(uri);
 } else if (HTTPMethod.equalsIgnoreCase("GET")){
  httpRequest = new HttpGet(uri);
 } else if (HTTPMethod.equalsIgnoreCase("DELETE")){
  httpRequest = new HttpDelete(uri);
 } else {
  System.err.println("The HTTP Method must be PUT,
  POST, GET or DELETE.");
  throw new IOException();
 }

 httpRequest.addHeader("Date", date);
 httpRequest.addHeader("Authorization", "AWS " + this.getAccessKey()
 + ":" + base64Sha1Hmac(headerString.toString(),
 this.getSecretKey()));

 httpclient = HttpClients.createDefault();
 httpResponse = httpclient.execute(httpRequest);

} catch  (URISyntaxException e){
 System.err.println("The URI is not formatted properly.");
 e.printStackTrace();
} catch (IOException e){
 System.err.println("There was an error making the request.");
 e.printStackTrace();
}
 return httpResponse;
}

/*
 * Takes a uri and a secret key and returns a base64-encoded
 * SHA-1 HMAC.
 */
public String base64Sha1Hmac(String uri, String secretKey) {
```

```
  try {

   byte[] keyBytes = secretKey.getBytes("UTF-8");
   SecretKeySpec signingKey = new SecretKeySpec(keyBytes, "HmacSHA1");

   Mac mac = Mac.getInstance("HmacSHA1");
   mac.init(signingKey);

   byte[] rawHmac = mac.doFinal(uri.getBytes("UTF-8"));

   Encoder base64 = Base64.getEncoder();
   return base64.encodeToString(rawHmac);

  } catch (Exception e) {
   throw new RuntimeException(e);
  }
 }

}
```

The subsequent **CephAdminAPIClient** example illustrates how to instantiate the **CephAdminAPI** class, build a map of request parameters, and use the **execute()** method to create, get, update and delete a user.

```
import java.io.IOException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.HttpEntity;
import org.apache.http.util.EntityUtils;
import java.util.*;


public class CephAdminAPIClient {

 public static void main (String[] args){

  CephAdminAPI adminApi = new CephAdminAPI ("FFC6ZQ6EMIF64194158N",
                          "Xac39eCAhlTGcCAUreuwe1ZuH5oVQFa51lbEMVoT",
                          "ceph-client");

  /*
   * Create a user
   */
  Map requestArgs = new HashMap();
  requestArgs.put("access", "usage=read, write; users=read, write");
  requestArgs.put("display-name", "New User");
  requestArgs.put("email", "new-user@email.com");
  requestArgs.put("format", "json");
  requestArgs.put("uid", "new-user");

  CloseableHttpResponse response =
   adminApi.execute("PUT", "/admin/user", null, requestArgs);

  System.out.println(response.getStatusLine());
  HttpEntity entity = response.getEntity();

  try {
```

```
 System.out.println("\nResponse Content is: "
  + EntityUtils.toString(entity, "UTF-8") + "\n");
 response.close();
} catch (IOException e){
 System.err.println ("Encountered an I/O exception.");
 e.printStackTrace();
}

/*
 * Get a user
 */
requestArgs = new HashMap();
requestArgs.put("format", "json");
requestArgs.put("uid", "new-user");

response = adminApi.execute("GET", "/admin/user", null, requestArgs);

System.out.println(response.getStatusLine());
entity = response.getEntity();

try {
 System.out.println("\nResponse Content is: "
  + EntityUtils.toString(entity, "UTF-8") + "\n");
 response.close();
} catch (IOException e){
 System.err.println ("Encountered an I/O exception.");
 e.printStackTrace();
}

/*
 * Modify a user
 */
requestArgs = new HashMap();
requestArgs.put("display-name", "John Doe");
requestArgs.put("email", "johndoe@email.com");
requestArgs.put("format", "json");
requestArgs.put("uid", "new-user");
requestArgs.put("max-buckets", "100");

response = adminApi.execute("POST", "/admin/user", null, requestArgs);

System.out.println(response.getStatusLine());
entity = response.getEntity();

try {
 System.out.println("\nResponse Content is: "
  + EntityUtils.toString(entity, "UTF-8") + "\n");
 response.close();
} catch (IOException e){
 System.err.println ("Encountered an I/O exception.");
 e.printStackTrace();
}


/*
 * Create a subuser
```

```
 */
requestArgs = new HashMap();
requestArgs.put("format", "json");
requestArgs.put("uid", "new-user");
requestArgs.put("subuser", "foobar");

response = adminApi.execute("PUT", "/admin/user", "subuser", requestArgs);
System.out.println(response.getStatusLine());
entity = response.getEntity();

try {
 System.out.println("\nResponse Content is: "
  + EntityUtils.toString(entity, "UTF-8") + "\n");
 response.close();
} catch (IOException e){
 System.err.println ("Encountered an I/O exception.");
 e.printStackTrace();
}


/*
 * Delete a user
 */
requestArgs = new HashMap();
requestArgs.put("format", "json");
requestArgs.put("uid", "new-user");

response = adminApi.execute("DELETE", "/admin/user", null, requestArgs);
System.out.println(response.getStatusLine());
entity = response.getEntity();

try {
 System.out.println("\nResponse Content is: "
  + EntityUtils.toString(entity, "UTF-8") + "\n");
 response.close();
} catch (IOException e){
 System.err.println ("Encountered an I/O exception.");
 e.printStackTrace();
}
 }
}
```

Return to the API function list.

1.2. CREATING AN ADMINISTRATIVE USER

IMPORTANT

To run the **radosgw-admin** command from the Ceph Object Gateway node, please ensure the node has the admin key (which can be copied from any monitor node).

Follow these steps to use the Ceph Object Gateway Administrative API:

1. Create an object gateway user:

Syntax

```
radosgw-admin user create --uid="<user_name>" --display-name="<display_name>"
```

Example

```
radosgw-admin user create --uid="admin-api-user" --display-name="Admin API User"
```

The **radosgw-admin** command-line interface will return the user. For example:

```
{
    "user_id": "admin-api-user",
    "display_name": "Admin API User",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "admin-api-user",
            "access_key": "NRWGT19TWMYOB1YDBV1Y",
            "secret_key": "gr1VEGIV7rxcP3xvXDFCo4UDwwl2YoNrmtRlIAty"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

2. Assign administrative capabilities to the user you create:

Syntax

```
radosgw-admin caps add --uid="<user_name>" --caps="users=*"
```

Example

```
radosgw-admin caps add --uid=admin-api-user --caps="users=*"
```

The **radosgw-admin** command-line interface will return the user. The **"caps":** will have the capabilities you assigned to the user:

```
{
    "user_id": "admin-api-user",
    "display_name": "Admin API User",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "admin-api-user",
            "access_key": "NRWGT19TWMYOB1YDBV1Y",
            "secret_key": "gr1VEGIV7rxcP3xvXDFCo4UDwwl2YoNrmtRlIAty"
        }
    ],
    "swift_keys": [],
    "caps": [
        {
            "type": "users",
            "perm": "*"
        }
    ],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

Now you have a user with administrative privileges.

Return to the API function list.

1.3. ADMINISTRATIVE OPERATIONS

An administrative Application Programming Interface (API) request will be done on a URI that starts with the configurable 'admin' resource entry point. Authorization for the administrative API duplicates the S3 authorization mechanism. Some operations require that the user holds special administrative capabilities. The response entity type, either XML or JSON, might be specified as the 'format' option in the request and defaults to JSON if not specified.

1.3.1. Get Usage

Requesting bandwidth usage information.

caps

usage=read

Syntax

```
GET /admin/usage?format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
```

Table 1.1. Request Parameters

Name	Description	Type	Required
uid	The user for which the information is requested.	String.	Yes
start	Date and (optional) time that specifies the start time of the requested data. E.g., **2012-09-25 16:00:00**	String	No
end	Date and (optional) time that specifies the end time of the requested data (non-inclusive). E.g., **2012-09-25 16:00:00**	String	No
show-entries	Specifies whether data entries should be returned.	Boolean	No
show-summary	Specifies whether data summary should be returned.	Boolean	No

Table 1.2. Response Entities

Name	Description	Type
usage	A container for the usage information.	Container
entries	A container for the usage entries information.	Container
user	A container for the user data information.	Container
owner	The name of the user that owns the buckets.	String
bucket	The bucket name.	String
time	Time lower bound for which data is being specified (rounded to the beginning of the first relevant hour).	String
epoch	The time specified in seconds since **1/1/1970**.	String
categories	A container for stats categories.	Container

Name	Description	Type
entry	A container for stats entry.	Container
category	Name of request category for which the stats are provided.	String
bytes_sent	Number of bytes sent by the Ceph Object Gateway.	Integer
bytes_received	Number of bytes received by the Ceph Object Gateway.	Integer
ops	Number of operations.	Integer
successful_ops	Number of successful operations.	Integer
summary	A container for stats summary.	Container
total	A container for stats summary aggregated total.	Container

If successful, the response contains the requested information.

Return to the API function list.

1.3.2. Trim Usage

Remove usage information. With no dates specified, removes all usage information.

caps

usage=write

Syntax

```
DELETE /admin/usage?format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
```

Table 1.3. Request Parameters

Name	Description	Type	Example	Required
uid	The user for which the information is requested.	String	**foo_user**	No
start	Date and (optional) time that specifies the start time of the requested data.	String	**2012-09-25 16:00:00**	No

Name	Description	Type	Example	Required
end	Date and (optional) time that specifies the end time of the requested data (none inclusive).	String	**2012-09-25 16:00:00**	No
remove-all	Required when **uid** is not specified, in order to acknowledge multi-user data removal.	Boolean	True [False]	No

Return to the API function list.

1.3.3. Get User Information

Get the user's information.

caps

users=read

Syntax

```
GET /admin/user?format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
```

Table 1.4. Request Parameters

Name	Description	Type	Example	Required
uid	The user for which the information is requested.	String	**foo_user**	Yes

Table 1.5. Response Entities

Name	Description	Type	Parent
user	A container for the user data information.	Container	N/A
user_id	The user ID.	String	**user**
display_name	Display name for the user.	String	**user**
suspended	True if the user is suspended.	Boolean	**user**
max_buckets	The maximum number of buckets to be owned by the user.	Integer	**user**

Name	Description	Type	Parent
subusers	Subusers associated with this user account.	Container	**user**
keys	S3 keys associated with this user account.	Container	**user**
swift_keys	Swift keys associated with this user account.	Container	**user**
caps	User capabilities.	Container	**user**

If successful, the response contains the user information.

Special Error Responses

None.

Return to the API function list.

1.3.4. Creating a User

Create a new user. By Default, a S3 key pair will be created automatically and returned in the response. If only one of **access-key** or **secret-key** is provided, the omitted key will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If **access-key** is specified and refers to an existing key owned by the user then it will be modified.

caps

users=write

Syntax

```
PUT /admin/user?format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
```

Table 1.6. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to be created.	String	**foo_user**	Yes
display-name	The display name of the user to be created.	String	**foo user**	Yes
email	The email address associated with the user.	String	**foo@bar.com**	No
key-type	Key type to be generated, options are: swift, s3 (default).	String	**s3** [**s3**]	No

Name	Description	Type	Example	Required
access-key	Specify access key.	String	**ABCD0EF12GHIJ2K34LMN**	No
secret-key	Specify secret key.	String	**0AbCDEFg1h2i34JkIM5nop6QrSTUV+WxyzaBC7D8**	No
user-caps	User capabilities.	String	**usage=read, write; users=read**	No
generate-key	Generate a new key pair and add to the existing keyring.	Boolean	True [True]	No
max-buckets	Specify the maximum number of buckets the user can own.	Integer	500 [1000]	No
suspended	Specify whether the user should be suspended.	Boolean	False [False]	No

Table 1.7. Response Entities

Name	Description	Type	Parent
user	A container for the user data information.	Container	N/A
user_id	The user ID.	String	**user**
display_name	Display name for the user.	String	**user**
suspended	True if the user is suspended.	Boolean	**user**
max_buckets	The maximum number of buckets to be owned by the user.	Integer	**user**
subusers	Subusers associated with this user account.	Container	**user**
keys	S3 keys associated with this user account.	Container	**user**
swift_keys	Swift keys associated with this user account.	Container	**user**
caps	User capabilities.	Container	**user**

If successful, the response contains the user information.

Table 1.8. Special Error Responses

Name	Description	Code
UserExists	Attempt to create existing user.	409 Conflict
InvalidAccessKey	Invalid access key specified.	400 Bad Request
InvalidKeyType	Invalid key type specified.	400 Bad Request
InvalidSecretKey	Invalid secret key specified.	400 Bad Request
InvalidKeyType	Invalid key type specified.	400 Bad Request
KeyExists	Provided access key exists and belongs to another user.	409 Conflict
EmailExists	Provided email address exists.	409 Conflict
InvalidCap	Attempt to grant invalid admin capability.	400 Bad Request

See Section 1.3.7, "Creating a Subuser" for creating subusers.

Return to the API function list.

1.3.5. Modifying a User

Modify an existing user.

caps

users=write

Syntax

```
POST /admin/user?format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
```

Table 1.9. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to be modified.	String	**foo_user**	Yes
display-name	The display name of the user to be modified.	String	**foo user**	No

Name	Description	Type	Example	Required
email	The email address to be associated with the user.	String	**foo@bar.com**	No
generate-key	Generate a new key pair and add to the existing keyring.	Boolean	True [False]	No
access-key	Specify access key.	String	**ABCD0EF12GHIJ2K34LMN**	No
secret-key	Specify secret key.	String	**0AbCDEFg1h2i34JkIM5nop6QrSTUV+WxyzaBC7D8**	No
key-type	Key type to be generated, options are: swift, s3 (default).	String	**s3**	No
user-caps	User capabilities.	String	**usage=read, write; users=read**	No
max-buckets	Specify the maximum number of buckets the user can own.	Integer	500 [1000]	No
suspended	Specify whether the user should be suspended.	Boolean	False [False]	No

Table 1.10. Response Entities

Name	Description	Type	Parent
user	A container for the user data information.	Container	N/A
user_id	The user ID.	String	**user**
display_name	Display name for the user.	String	**user**
suspended	True if the user is suspended.	Boolean	**user**
max_buckets	The maximum number of buckets to be owned by the user.	Integer	**user**

Name	Description	Type	Parent
subusers	Subusers associated with this user account.	Container	**user**
keys	S3 keys associated with this user account.	Container	**user**
swift_keys	Swift keys associated with this user account.	Container	**user**
caps	User capabilities.	Container	**user**

If successful, the response contains the user information.

Table 1.11. Special Error Responses

Name	Description	Code
InvalidAccessKey	Invalid access key specified.	400 Bad Request
InvalidKeyType	Invalid key type specified.	400 Bad Request
InvalidSecretKey	Invalid secret key specified.	400 Bad Request
KeyExists	Provided access key exists and belongs to another user.	409 Conflict
EmailExists	Provided email address exists.	409 Conflict
InvalidCap	Attempt to grant invalid admin capability.	400 Bad Request

See Section 1.3.8, "Modifying a Subuser" for modifying subusers.

Return to the API function list.

1.3.6. Removing a User

Remove an existing user.

caps

users=write

Syntax

```
DELETE /admin/user?format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
```

Table 1.12. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to be removed.	String	**foo_user**	Yes.
purge-data	When specified the buckets and objects belonging to the user will also be removed.	Boolean	True	No

Response Entities

None.

Special Error Responses

None.

See Section 1.3.9, "Removing a Subuser" for removing subusers.

Return to the API function list.

1.3.7. Creating a Subuser

Create a new subuser, primarily useful for clients using the Swift API. Note that either **gen-subuser** or **subuser** is required for a valid request. Also, note that in general for a subuser to be useful, it must be granted permissions by specifying **access**. As with user creation if **subuser** is specified without **secret**, then a secret key will be automatically generated.

caps

users=write

Syntax

```
PUT /admin/user?subuser&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.13. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID under which a subuser is to be created.	String	**foo_user**	Yes
subuser	Specify the subuser ID to be created.	String	**sub_foo**	Yes (or **gen-subuser**)
gen-subuser	Specify the subuser ID to be created.	String	**sub_foo**	Yes (or **subuser**)

Name	Description	Type	Example	Required
secret-key	Specify secret key.	String	**0AbCDEFg1h2i34JklM5nop6QrSTUVWxyzaBC7D8**	No
key-type	Key type to be generated, options are: swift (default), s3.	String	**swift** [**swift**]	No
access	Set access permissions for sub-user, should be one of **read, write, readwrite, full**.	String	**read**	No
generate-secret	Generate the secret key.	Boolean	True [False]	No

Table 1.14. Response Entities

Name	Description	Type	Parent
subusers	Subusers associated with the user account.	Container	N/A
id	Subuser ID.	String	**subusers**
permissions	Subuser access to user account.	String	**subusers**

If successful, the response contains the subuser information.

Table 1.15. Special Error Responses

Name	Description	Code
SubuserExists	Specified subuser exists.	409 Conflict
InvalidKeyType	Invalid key type specified.	400 Bad Request

Name	Description	Code
InvalidSecretKey	Invalid secret key specified.	400 Bad Request
InvalidAccess	Invalid subuser access specified.	400 Bad Request

Return to the API function list.

1.3.8. Modifying a Subuser

Modify an existing subuser.

caps

users=write

Syntax

```
POST /admin/user?subuser&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.16. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID under which the subuser is to be modified.	String	**foo_user**	Yes
subuser	The subuser ID to be modified.	String	**sub_foo**	Yes
generate-secret	Generate a new secret key for the subuser, replacing the existing key.	Boolean	True [False]	No
secret	Specify secret key.	String	**0AbCDEFg1h2i34JkIM5nop6QrSTUV+WxyzaBC7D8**	No
key-type	Key type to be generated, options are: swift (default), s3.	String	**swift** [**swift**]	No
access	Set access permissions for sub-user, should be one of **read, write, readwrite, full**.	String	**read**	No

Table 1.17. Response Entities

Name	Description	Type	Parent
subusers	Subusers associated with the user account.	Container	N/A
id	Subuser ID.	String	**subusers**
permissions	Subuser access to user account.	String	**subusers**

If successful, the response contains the subuser information.

Table 1.18. Special Error Responses

Name	Description	Code
InvalidKeyType	Invalid key type specified.	400 Bad Request
InvalidSecretKey	Invalid secret key specified.	400 Bad Request
InvalidAccess	Invalid subuser access specified.	400 Bad Request

Return to the API function list.

1.3.9. Removing a Subuser

Remove an existing subuser.

caps

users=write

Syntax

```
DELETE /admin/user?subuser&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.19. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID under which the subuser is to be removed.	String	**foo_user**	Yes
subuser	The subuser ID to be removed.	String	**sub_foo**	Yes

Name	Description	Type	Example	Required
purge-keys	Remove keys belonging to the subuser.	Boolean	True [True]	No

Response Entities

None.

Special Error Responses

None.

Return to the API function list.

1.3.10. Creating a Key

Create a new key. If a **subuser** is specified then by default created keys will be swift type. If only one of **access-key** or **secret-key** is provided the committed key will be automatically generated, that is if only **secret-key** is specified then **access-key** will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If **access-key** is specified and refers to an existing key owned by the user then it will be modified. The response is a container listing all keys of the same type as the key created. Note that when creating a swift key, specifying the option **access-key** will have no effect. Additionally, only one swift key might be held by each user or subuser.

caps

users=write

Syntax

```
PUT /admin/user?key&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.20. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to receive the new key.	String	**foo_user**	Yes
subuser	The subuser ID to receive the new key.	String	**sub_foo**	No
key-type	Key type to be generated, options are: swift, s3 (default).	String	**s3** [**s3**]	No
access-key	Specify the access key.	String	**AB01C2D3EF45G6H7IJ8K**	No

Name	Description	Type	Example	Required
secret-key	Specify the secret key.	String	**0ab/CdeFGhij1klmnopqRSTUv1WxyZabcDEFgHij**	No
generate-key	Generate a new key pair and add to the existing keyring.	Boolean	True [**True**]	No

Table 1.21. Response Entities

Name	Description	Type	Parent
keys	Keys of type created associated with this user account.	Container	N/A
user	The user account associated with the key.	String	**keys**
access-key	The access key.	String	**keys**
secret-key	The secret key	String	**keys**

Table 1.22. Special Error Responses

Name	Description	Code
InvalidAccessKey	Invalid access key specified.	400 Bad Request
InvalidKeyType	Invalid key type specified.	400 Bad Request
InvalidSecretKey	Invalid secret key specified.	400 Bad Request
InvalidKeyType	Invalid key type specified.	400 Bad Request
KeyExists	Provided access key exists and belongs to another user.	409 Conflict

Return to the API function list.

1.3.11. Removing a Key

Remove an existing key.

caps

users=write

Syntax

```
DELETE /admin/user?key&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.23. Request Parameters

Name	Description	Type	Example	Required
access-key	The S3 access key belonging to the S3 key pair to remove.	String	**AB01C2D3EF45G6H7IJ8K**	Yes
uid	The user to remove the key from.	String	**foo_user**	No
subuser	The subuser to remove the key from.	String	**sub_foo**	No
key-type	Key type to be removed, options are: swift, s3. NOTE: Required to remove swift key.	String	**swift**	No

Special Error Responses

None.

Response Entities

None.

Return to the API function list.

1.3.12. Getting Bucket Information

Get information about a subset of the existing buckets. If **uid** is specified without **bucket** then all buckets belonging to the user will be returned. If **bucket** alone is specified, information for that particular bucket will be retrieved.

caps

buckets=read

Syntax

```
GET /admin/bucket?format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.24. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket to return info on.	String	**foo_bucket**	No
uid	The user to retrieve bucket information for.	String	**foo_user**	No
stats	Return bucket statistics.	Boolean	True [False]	No

Table 1.25. Response Entities

Name	Description	Type	Parent
stats	Per bucket information.	Container	N/A
buckets	Contains a list of one or more bucket containers.	Container	**bucket**
Container for single bucket information.	Container	**buckets**	**name**
The name of the bucket.	String	**bucket**	**pool**
The pool the bucket is stored in.	String	**bucket**	**id**
The unique bucket ID.	String	**bucket**	**marker**
Internal bucket tag.	String	**bucket**	**owner**
The user ID of the bucket owner.	String	**bucket**	**usage**
Storage usage information.	Container	**bucket**	**index**

If successful the request returns a buckets container containing the desired bucket information.

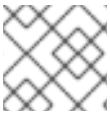
Table 1.26. Special Error Responses

Name	Description	Code
IndexRepairFailed	Bucket index repair failed.	409 Conflict

Return to the API function list.

1.3.13. Checking a Bucket Index

Check the index of an existing bucket.



NOTE

To check multipart object accounting with **check-objects**, **fix** must be set to True.

caps

buckets=write

Syntax

```
GET /admin/bucket?index&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.27. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket to return info on.	String	**foo_bucket**	Yes
check-objects	Check multipart object accounting.	Boolean	True [False]	No
fix	Also fix the bucket index when checking.	Boolean	False [False]	No

Table 1.28. Response Entities

Name	Description	Type
index	Status of bucket index.	String

Table 1.29. Special Error Responses

Name	Description	Code
IndexRepairFailed	Bucket index repair failed.	409 Conflict

Return to the API function list.

1.3.14. Removing a Bucket

Removes an existing bucket.

caps

buckets=write

Syntax

```
DELETE /admin/bucket?format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.30. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket to remove.	String	**foo_bucket**	Yes
purge-objects	Remove a buckets objects before deletion.	Boolean	True [False]	No

Response Entities

None.

Table 1.31. Special Error Responses

Name	Description	Code
BucketNotEmpty	Attempted to delete non-empty bucket.	409 Conflict
ObjectRemovalFailed	Unable to remove objects.	409 Conflict

Return to the API function list.

1.3.15. Linking a Bucket

Link a bucket to a specified user, unlinking the bucket from any previous user.

caps

buckets=write

Syntax

```
PUT /admin/bucket?format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.32. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket to unlink.	String	**foo_bucket**	Yes
uid	The user ID to link the bucket to.	String	**foo_user**	Yes

Table 1.33. Response Entities

Name	Description	Type	Parent
bucket	Container for single bucket information.	Container	N/A
name	The name of the bucket.	String	**bucket**
pool	The pool the bucket is stored in.	String	**bucket**
id	The unique bucket ID.	String	**bucket**
marker	Internal bucket tag.	String	**bucket**
owner	The user ID of the bucket owner.	String	**bucket**
usage	Storage usage information.	Container	**bucket**
index	Status of bucket index.	String	**bucket**

Table 1.34. Special Error Responses

Name	Description	Code
BucketUnlinkFailed	Unable to unlink bucket from specified user.	409 Conflict
BucketLinkFailed	Unable to link bucket to specified user.	409 Conflict

Return to the API function list.

1.3.16. Unlinking a Bucket

Unlink a bucket from a specified user. Primarily useful for changing bucket ownership.

caps

buckets=write

Syntax

```
POST /admin/bucket?format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.35. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket to unlink.	String	**foo_bucket**	Yes
uid	The user ID to unlink the bucket from.	String	**foo_user**	Yes

Response Entities

None.

Table 1.36. Special Error Responses

Name	Description	Code
BucketUnlinkFailed	Unable to unlink bucket from specified user.	409 Conflict

Return to the API function list.

1.3.17. Removing an Object

Remove an existing object.

NOTE

Does not require owner to be non-suspended.

caps

buckets=write

Syntax

```
DELETE /admin/bucket?object&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.37. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket containing the object to be removed.	String	**foo_bucket**	Yes
object	The object to remove.	String	**foo.txt**	Yes

Response Entities

None.

Table 1.38. Special Error Responses

Name	Description	Code
NoSuchObject	Specified object does not exist.	404 Not Found
ObjectRemovalFailed	Unable to remove objects.	409 Conflict

Return to the API function list.

1.3.18. Getting Bucket or Object Policy

Read the policy of an object or bucket.

caps

buckets=read

Syntax

```
GET /admin/bucket?policy&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.39. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket to read the policy from.	String	**foo_bucket**	Yes
object	The object to read the policy from.	String	**foo.txt**	No

Table 1.40. Response Entities

Name	Description	Type	Parent
policy	Access control policy.	Container	N/A

If successful, returns the object or bucket policy

Table 1.41. Special Error Responses

Name	Description	Code
IncompleteBody	Either bucket was not specified for a bucket policy request or bucket and object were not specified for an object policy request.	400 Bad Request

Return to the API function list.

1.3.19. Adding a Capability to an Existing User

Add an administrative capability to a specified user.

caps

users=write

Syntax

```
PUT /admin/user?caps&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.42. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to add an administrative capability to.	String	**foo_user**	Yes

Name	Description	Type	Example	Required
user-caps	The administrative capability to add to the user.	String	**usage=read, write**	Yes

Table 1.43. Response Entities

Name	Description	Type	Parent
user	A container for the user data information.	Container	N/A
user_id	The user ID.	String	**user**
caps	User capabilities.	Container	**user**

If successful, the response contains the user's capabilities.

Table 1.44. Special Error Responses

Name	Description	Code
InvalidCap	Attempt to grant invalid admin capability.	400 Bad Request

Return to the API function list.

1.3.19.1. Example Request

```
PUT /admin/user?caps&format=json HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
Content-Type: text/plain
Authorization: <Authorization_Token>

usage=read
```

1.3.20. Removing a Capability from an Existing User

Remove an administrative capability from a specified user.

caps

users=write

Syntax

```
DELETE /admin/user?caps&format=json HTTP/1.1
Host <Fully_Qualified_Domain_Name>
```

Table 1.45. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to remove an administrative capability from.	String	**foo_user**	Yes
user-caps	The administrative capabilities to remove from the user.	String	**usage=read, write**	Yes

Table 1.46. Response Entities

Name	Description	Type	Parent
user	A container for the user data information.	Container	N/A
user_id	The user ID.	String	**user**
caps	User capabilities.	Container	**user**

If successful, the response contains the user's capabilities.

Table 1.47. Special Error Responses

Name	Description	Code
InvalidCap	Attempt to remove an invalid admin capability.	400 Bad Request
NoSuchCap	User does not possess specified capability.	404 Not Found

Return to the API function list.

1.3.21. Quotas

The administrative Operations API enables you to set quotas on users and on bucket owned by users. See Quota Management for additional details. Quotas include the maximum number of objects in a bucket and the maximum storage size in megabytes.

To view quotas, the user must have a **users=read** capability. To set, modify or disable a quota, the user must have **users=write** capability. See the Administration (CLI) for details.

Valid parameters for quotas include:

- **Bucket:** The **bucket** option allows you to specify a quota for buckets owned by a user.
- **Maximum Objects:** The **max-objects** setting allows you to specify the maximum number of objects. A negative value disables this setting.

- **Maximum Size:** The **max-size** option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
- **Quota Scope:** The **quota-scope** option sets the scope for the quota. The options are **bucket** and **user**.

Return to the API function list.

1.3.21.1. Getting User Quota

To get a quota, the user must have **users** capability set with **read** permission.

Syntax

```
GET /admin/user?quota&uid=<uid>&quota-type=user
```

Return to the API function list.

1.3.21.2. Setting User Quota

To set a quota, the user must have **users** capability set with **write** permission.

Syntax

```
PUT /admin/user?quota&uid=<uid>&quota-type=user
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

Return to the API function list.

1.3.21.3. Getting Bucket Quota

To get a quota, the user must have **users** capability set with **read** permission.

Syntax

```
GET /admin/user?quota&uid=<uid>&quota-type=bucket
```

Return to the API function list.

1.3.21.4. Setting Bucket Quota

To set a quota, the user must have **users** capability set with **write** permission.

Syntax

```
PUT /admin/user?quota&uid=<uid>&quota-type=bucket
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

Return to the API function list.

1.3.22. Standard Error Responses

Name	Description	Code
AccessDenied	Access denied.	403 Forbidden
InternalError	Internal server error.	500 Internal Server Error
NoSuchUser	User does not exist.	404 Not Found
NoSuchBucket	Bucket does not exist.	404 Not Found
NoSuchKey	No such access key.	404 Not Found

CHAPTER 2. OBJECT GATEWAY S3 APPLICATION PROGRAMMING INTERFACE (API)

Red Hat Ceph Object Gateway supports a RESTful API that is compatible with the basic data access model of the Amazon S3 API.

The following table describes the support status for current Amazon S3 functional features.

Table 2.1. Features

Feature	Status	Remarks
List Buckets	Supported	
Create Bucket	Supported	Different set of canned ACLs
Put Bucket Lifecycle	Partially Supported	**Expiration**, **NoncurrentVersionExpiration** and **AbortIncompleteMultipartUpload** supported.
Get Bucket	Supported	
Get Bucket Lifecycle	Partially Supported	**Expiration**, **NoncurrentVersionExpiration** and **AbortIncompleteMultipartUpload** supported.
Get Bucket Location	Supported	
Get Bucket Versioning	Supported	
Delete Bucket	Supported	
Delete Bucket Lifecycle	Supported	
Bucket ACLs (Get, Put)	Supported	Different set of canned ACLs
Bucket cors (Get, Put, Delete)	Supported	
Bucket Object Versions	Supported	
Head Bucket	Supported	
List Bucket Multipart Uploads	Supported	

Feature	Status	Remarks
Bucket Lifecycle	Partially Supported	**Expiration**, **NoncurrentVersionExpiration** and **AbortIncompleteMultipartUpload** supported.
Policy (Buckets)	Partially Supported	
Bucket Website	Supported	
Bucket Request Payment (Get, Put)	Supported	
Put Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Object ACLs (Get, Put)	Supported	
Get Object Info (HEAD)	Supported	
Copy Object	Supported	
Post Object	Supported	
Options Object	Supported	
Delete Multiple Objects	Supported	
Initiate Multipart Upload	Supported	
Initiate Multipart Upload Part	Supported	
List Multipart Upload Parts	Supported	
Complete Multipart Upload	Supported	
Abort Multipart Upload	Supported	
Copy Multipart Upload	Supported	
Multi Tenancy	Supported	

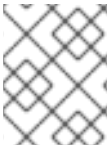
The following table lists the common request header fields that are not supported.

Table 2.2. Unsupported Header Fields

Name	Type
x-amz-security-token	Request
Server	Response
x-amz-delete-marker	Response
x-amz-id-2	Response
x-amz-request-id	Response
x-amz-version-id	Response

2.1. S3 API SERVER-SIDE ENCRYPTION

The Ceph Object Gateway supports server-side encryption of uploaded objects for the S3 API. Server-side encryption means that the S3 client sends data over HTTP in its unencrypted form, and the Ceph Object Gateway stores that data in the Ceph Storage Cluster in encrypted form.



NOTE

Red Hat does NOT support S3 object encryption of SLO(Static Large Object) and DLO(Dynamic Large Object).

IMPORTANT

To use encryption, client requests **MUST** send requests over an SSL connection. Red Hat does not support S3 encryption from a client unless the Ceph Object Gateway uses SSL. However, for testing purposes, administrators may disable SSL during testing by setting the **rgw_crypt_require_ssl** configuration setting to **false** at runtime, setting it to **false** in the Ceph configuration file and restarting the gateway instance, or setting it to **false** in the Ansible configuration files and replaying the Ansible playbooks for the Ceph Object Gateway.

There are two options for the management of encryption keys:

Customer-Provided Keys

When using customer-provided keys, the S3 client passes an encryption key along with each request to read or write encrypted data. It is the customer's responsibility to manage those keys. Customers must remember which key the Ceph Object Gateway used to encrypt each object.

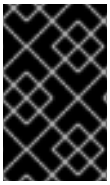
Ceph Object Gateway implements the customer-provided key behavior in the S3 API according to the Amazon SSE-C specification.

Since the customer handles the key management and the S3 client passes keys to the Ceph Object Gateway, the Ceph Object Gateway requires no special configuration to support this encryption mode.

Key Management Service

When using a key management service, the secure key management service stores the keys and the Ceph Object Gateway retrieves them on demand to serve requests to encrypt or decrypt data.

Ceph Object Gateway implements the key management service behavior in the S3 API according to the Amazon SSE-KMS specification.



IMPORTANT

Currently, the only tested key management implementation uses OpenStack Barbican. However, OpenStack Barbican is a Technology Preview and is not supported for use in production systems.

2.2. BUCKET POLICIES

The Ceph Object Gateway supports a subset of the Amazon S3 policy language applied to buckets.

Creation and Removal

Ceph Object Gateway manages S3 Bucket policies through standard S3 operations rather than using the **radosgw-admin** CLI tool.

Administrators may use the **s3cmd** command to set or delete a policy. For example:

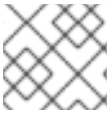
```
$ cat > examplepol
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": ["arn:aws:iam::usfolks:user/fred"]},
    "Action": "s3:PutObjectAcl",
    "Resource": [
      "arn:aws:s3:::happybucket/*"
    ]
  }]
}

$ s3cmd setpolicy examplepol s3://happybucket
$ s3cmd delpolicy s3://happybucket
```

Limitations

Ceph Object Gateway only supports the following S3 actions:

- **s3:AbortMultipartUpload**
- **s3:CreateBucket**
- **s3:DeleteBucketPolicy**
- **s3:DeleteBucket**
- **s3:DeleteBucketWebsite**
- **s3:DeleteObject**
- **s3:DeleteObjectVersion**

- **s3:GetBucketAcl**
- **s3:GetBucketCORS**
- **s3:GetBucketLocation**
- **s3:GetBucketPolicy**
- **s3:GetBucketRequestPayment**
- **s3:GetBucketVersioning**
- **s3:GetBucketWebsite**
- **s3:GetLifecycleConfiguration**
- **s3:GetObjectAcl**
- **s3:GetObject**
- **s3:GetObjectTorrent**
- **s3:GetObjectVersionAcl**
- **s3:GetObjectVersion**
- **s3:GetObjectVersionTorrent**
- **s3:ListAllMyBuckets**
- **s3:ListBucketMultiPartUploads**
- **s3:ListBucket**
- **s3:ListBucketVersions**
- **s3:ListMultipartUploadParts**
- **s3:PutBucketAcl**
- **s3:PutBucketCORS**
- **s3:PutBucketPolicy**
- **s3:PutBucketRequestPayment**
- **s3:PutBucketVersioning**
- **s3:PutBucketWebsite**
- **s3:PutLifecycleConfiguration**
- **s3:PutObjectAcl**
- **s3:PutObject**
- **s3:PutObjectVersionAcl**



NOTE

Ceph Object Gateway does not support setting policies on users, groups, or roles.

The Ceph Object Gateway uses the RGW 'tenant' identifier in place of the Amazon twelve-digit account ID. Ceph Object Gateway administrators who want to use policies between Amazon Web Service (AWS) S3 and Ceph Object Gateway S3 will have to use the Amazon account ID as the tenant ID when creating users.

With AWS S3, all tenants share a single namespace. By contrast, Ceph Object Gateway gives every tenant its own namespace of buckets. At present, Ceph Object Gateway clients trying to access a bucket belonging to another tenant MUST address it as **tenant:bucket** in the S3 request.

In the AWS, a bucket policy can grant access to another account, and that account owner can then grant access to individual users with user permissions. Since Ceph Object Gateway does not yet support user, role, and group permissions, account owners will need to grant access directly to individual users.

IMPORTANT

Granting an entire account access to a bucket grants access to ALL users in that account.

Bucket policies do **NOT** support string interpolation.

Ceph Object Gateway supports the following condition keys:

- **aws:CurrentTime**
- **aws:EpochTime**
- **aws:PrincipalType**
- **aws:Referer**
- **aws:SecureTransport**
- **aws:SourceIp**
- **aws:UserAgent**
- **aws:username**

Ceph Object Gateway **ONLY** supports the following condition keys for the **ListBucket** action:

- **s3:prefix**
- **s3:delimiter**
- **s3:max-keys**

Impact on Swift

Ceph Object Gateway provides no functionality to set bucket policies under the Swift API. However, bucket policies that have been set with the S3 API govern Swift as well as S3 operations.

Ceph Object Gateway matches Swift credentials against Principals specified in a policy.

2.3. AUTHENTICATION AND ACCESS CONTROL LISTS

Requests to the Ceph Object Gateway can be either authenticated or unauthenticated. Ceph Object Gateway assumes unauthenticated requests are sent by an anonymous user. Ceph Object Gateway supports canned ACLs.

2.3.1. Authentication

For most use cases, clients use existing open source libraries like the Amazon SDK's **AmazonS3Client** for Java, and Python Boto, where you simply pass in the access key and secret key, and the library builds the request header and authentication signature for you. However, you can create requests and sign them too.

Authenticating a request requires including an access key and a base 64-encoded Hash-based Message Authentication Code (HMAC) in the request before it is sent to the Ceph Object Gateway server. Ceph Object Gateway uses an S3-compatible authentication approach.

Example

```
HTTP/1.1
PUT /buckets/bucket/object.mpeg
Host: cname.domain.com
Date: Mon, 2 Jan 2012 00:01:01 +0000
Content-Encoding: mpeg
Content-Length: 9999999

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

In the foregoing example, replace **<access_key>** with the value for the access key ID followed by a colon (**:**). Replace **<hash_of_header_and_secret>** with a hash of a canonicalized header string and the secret corresponding to the access key ID.

To generate the hash of the header string and secret, you must:

1. Get the value of the header string.
2. Normalize the request header string into canonical form.
3. Generate an HMAC using a SHA-1 hashing algorithm.
4. Encode the **hmac** result as base-64.

To normalize the header into canonical form:

1. Get all **content-** headers.
2. Remove all **content-** headers except for **content-type** and **content-md5**.
3. Ensure the **content-** header names are lowercase.
4. Sort the **content-** headers lexicographically.
5. Ensure you have a **Date** header AND ensure the specified date uses GMT and not an offset.
6. Get all headers beginning with **x-amz-**.

7. Ensure that the **x-amz-** headers are all lowercase.
8. Sort the **x-amz-** headers lexicographically.
9. Combine multiple instances of the same field name into a single field and separate the field values with a comma.
10. Replace white space and line breaks in header values with a single space.
11. Remove white space before and after colons.
12. Append a new line after each header.
13. Merge the headers back into the request header.

Replace the **<hash_of_header_and_secret>** with the base-64 encoded HMAC string.

For additional details, consult the Signing and Authenticating REST Requests section of Amazon Simple Storage Service documentation.

2.3.2. Access Control Lists (ACLs)

Ceph Object Gateway supports S3-compatible ACL functionality. An ACL is a list of access grants that specify which operations a user can perform on a bucket or on an object. Each grant has a different meaning when applied to a bucket versus applied to an object:

Table 2.3. User Operations

Permission	Bucket	Object
READ	Grantee can list the objects in the bucket.	Grantee can read the object.
WRITE	Grantee can write or delete objects in the bucket.	N/A
READ_ACP	Grantee can read bucket ACL.	Grantee can read the object ACL.
WRITE_ACP	Grantee can write bucket ACL.	Grantee can write to the object ACL.
FULL_CONTROL	Grantee has full permissions for object in the bucket.	Grantee can read or write to the object ACL.

2.3.3. Authentication using the STS Lite API (Technology Preview)

The Ceph Object Gateway provides support for a subset of the Amazon Secure Token Service (STS) REST APIs. STS Lite provides access to a set of temporary credentials for identity and access management.

The STS Lite authentication mechanism is integrated with Keystone in the Ceph Object Gateway. A set of temporary security credentials are returned after authenticating a set of Amazon Web Services (AWS) credentials with Keystone. The STS engine authenticates these temporary security credentials made by subsequent S3 calls, resulting in less load on the Keystone server.

The Ceph Object Gateway implements the following STS Lite REST APIs:

GetSessionToken

Returns a set of temporary security credentials for a set of AWS credentials. Use this API for initial authentication with Keystone and the temporary credentials returned can be used to make subsequent S3 calls. The temporary credentials will have the same permission as that of the AWS credentials.
Parameters:

DurationSeconds (Integer/ Optional)

The duration in seconds for which the credentials should remain valid. The default value is 3600 seconds. The max value is 43200 seconds. You can set this value using the **rgw_sts_max_session** option in the Ceph configuration file, by default the **/etc/ceph/ceph.conf** file.

SerialNumber (String/ Optional)

The identifier number of the Multi-Factor Authentication (MFA) device associated with the user making the GetSessionToken call.

TokenCode (String/ Optional)

The value provided by the MFA device, if MFA is required.

AssumeRole

Returns a set of temporary credentials that can be used for cross-account access. The temporary credentials will have permissions that are allowed by both - permission policies attached with the Role and policy attached with the AssumeRole API.
Parameters:

RoleArn (String/ Required)

Amazon Resource Name (ARN) of the Role to Assume.

RoleSessionName (String/ Required)

An Identifier for the assumed role session.

Policy (String/ Optional):

An IAM Policy in JSON format.

DurationSeconds (Integer/ Optional)

The duration in seconds of the session. The default value is 3600.

ExternalId (String/ Optional)

A unique Id that might be used when a role is assumed in another account.

SerialNumber (String/ Optional)

The identifier number of the MFA device associated with the user making the *AssumeRole* call.

TokenCode (String/ Optional)

The value provided by the MFA device, if the trust policy of the role being assumed requires MFA.

Additional Resources

- See the AWS Security Token Service API Reference documentation for more information.
- See the AWS Identity and Access Management User Guide for more information.

2.4. ACCESSING THE GATEWAY

You can use various programming languages to create a connection with the gateway server and do the bucket management tasks. There are different open source libraries available for these programming languages that are used for authentication with the gateway.

The sections mentioned below will describe the procedure for some of the popular programming languages.

2.4.1. Prerequisites

You must follow some prerequisites on the Ceph Object Gateway node before attempting to access the gateway server. The prerequisites are as follows:

1. Set up the gateway server properly by following the instructions based on the operating system:
   a. For Red Hat Enterprise Linux, see the Ceph Object Gateway Installation chapter in the *Installation Guide for Red hat Enterprise Linux* .
   b. For Ubuntu, see the Ceph Object Gateway Installation chapter in the *Installation Guide for Ubuntu*.
2. **DO NOT** modify the Ceph configuration file to use port **80** and let **Civetweb** use the default Ansible configured port of **8080**.
3. As **root**, open port **8080** on firewall:

```
# firewall-cmd --zone=public --add-port=8080/tcp --permanent
# firewall-cmd --reload
```

4. Add a wildcard to the DNS server that you are using for the gateway as mentioned in the *Object Gateway Guide*.
   You can also set up the gateway node for local DNS caching. To do so, execute the following steps:
   a. As **root**, install and setup **dnsmasq**:

```
# yum install dnsmasq
# echo "address=/.<FQDN_of_gateway_node>/<IP_of_gateway_node>" | tee --append
/etc/dnsmasq.conf
# systemctl start dnsmasq
# systemctl enable dnsmasq
```

   Replace **<IP_of_gateway_node>** and **<FQDN_of_gateway_node>** with the IP address and FQDN of the gateway node.
   b. As **root**, stop NetworkManager:

```
# systemctl stop NetworkManager
# systemctl disable NetworkManager
```

   c. As **root**, set the gateway server's IP as the nameserver:

```
# echo "DNS1=<IP_of_gateway_node>" | tee --append /etc/sysconfig/network-
scripts/ifcfg-eth0
```

```
# echo "<IP_of_gateway_node> <FQDN_of_gateway_node>" | tee --append /etc/hosts
# systemctl restart network
# systemctl enable network
# systemctl restart dnsmasq
```

Replace **<IP_of_gateway_node>** and **<FQDN_of_gateway_node>** with the IP address and FQDN of the gateway node.

d. Verify subdomain requests:

```
$ ping mybucket.<FQDN_of_gateway_node>
```

Replace **<FQDN_of_gateway_node>** with the FQDN of the gateway node.

WARNING

Setting up the gateway server for local DNS caching is for testing purposes only. You won't be able to access outside network after doing this. **It is strongly recommended to use a proper DNS server for the Ceph cluster and gateway node.**

5. Create the **radosgw** user for **S3** access carefully as mentioned in the *Object Gateway Guide for Red Hat Enterprise Linux* or *Object Gateway Guide for Ubuntu* and copy the generated **access_key** and **secret_key**. You will need these keys for **S3** access and subsequent bucket management tasks.

2.4.2. Ruby AWS::S3 Examples (aws-s3 gem)

You can use Ruby programming language along with **aws-s3** gem for **S3** access. Execute the steps mentioned below on the node used for accessing the Ceph Object Gateway server with **Ruby AWS::S3**.

Setup Ruby

Execute the following steps to setup Ruby:

1. As **root**, install **ruby**:

```
# yum install ruby
```

NOTE

The above command will install **ruby** and it's essential dependencies like **rubygems** and **ruby-libs** too. If somehow the command doesn't install all the dependencies, install them separately.

2. As **root**, install **aws-s3**:

```
# gem install aws-s3
```

Creating a connection

1. Create a project directory:

```
$ mkdir ruby_aws_s3
$ cd ruby_aws_s3
```

2. Create the connection file:

```
$ vim conn.rb
```

3. Paste the following contents into the **conn.rb** file:

```
#!/usr/bin/env ruby

require 'aws/s3'
require 'resolv-replace'

AWS::S3::Base.establish_connection!(
        :server            => '<FQDN_of_gateway_node>',
        :port          => '8080',
        :access_key_id     => 'my-access-key',
        :secret_access_key => 'my-secret-key'
)
```

Replace **<FQDN_of_gateway_node>** with the FQDN of your gateway node. Replace **my-access-key** and **my-secret-key** with the **access_key** and **secret_key** that were generated when you created the **radosgw** user for **S3** access as mentioned in the *Object Gateway Guide for Red Hat Enterprise Linux* or the *Object Gateway Guide for Ubuntu* .

An example connection file looks like the following:

```
#!/usr/bin/env ruby

require 'aws/s3'

require 'resolv-replace'

AWS::S3::Base.establish_connection!(
        :server            => 'testclient.englab.pnq.redhat.com',
        :port          => '8080',
        :access_key_id     => '98J4R9P22P5CDL65HKP8',
        :secret_access_key => '6C+jcaP0dp0+FZfrRNgyGA9EzRy25pURldwje049'
)
```

Save the file and exit the editor.

4. Make the file executable:

```
$ chmod +x conn.rb
```

5. Run the file:

```
$ ./conn.rb | echo $?
```

If you have provided the values correctly in the file, the output of the command will be **0**.

Creating a bucket

1. Create a new file:

```
$ vim create_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Bucket.create('my-new-bucket1')
```

Save the file and exit the editor.

2. Make the file executable:

```
$ chmod +x create_bucket.rb
```

3. Run the file:

```
$ ./create_bucket.rb
```

If the output of the command is **true** it would mean that bucket **my-new-bucket1** was created successfully.

Listing owned buckets

1. Create a new file:

```
$ vim list_owned_buckets.rb
```

Paste the following content into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Service.buckets.each do |bucket|
        puts "#{bucket.name}\t#{bucket.creation_date}"
end
```

Save the file and exit the editor.

2. Make the file executable:

```
$ chmod +x list_owned_buckets.rb
```

3. Run the file:

```
$ ./list_owned_buckets.rb
```

The output should look something like this:

```
my-new-bucket1 2016-01-21 10:33:19 UTC
```

Creating an object

1. Create a new file:

```
$ vim create_object.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::S3Object.store(
        'hello.txt',
        'Hello World!',
        'my-new-bucket1',
        :content_type => 'text/plain'
)
```

Save the file and exit the editor.

2. Make the file executable:

```
$ chmod +x create_object.rb
```

3. Run the file:

```
$ ./create_object.rb
```

This will create a file **hello.txt** with the string **Hello World!**.

Listing a Bucket's Content

1. Create a new file:

```
$ vim list_bucket_content.rb
```

Paste the following content into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

new_bucket = AWS::S3::Bucket.find('my-new-bucket1')
new_bucket.each do |object|
        puts "#{object.key}\t#{object.about['content-length']}\t#{object.about['last-modified']}"
end
```

-

Save the file and exit the editor.

2. Make the file executable.

```
$ chmod +x list_bucket_content.rb
```

3. Run the file:

```
$ ./list_bucket_content.rb
```

The output will look something like this:

```
hello.txt    12    Fri, 22 Jan 2016 15:54:52 GMT
```

Deleting a empty bucket

1. Create a new file:

```
$ vim del_empty_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Bucket.delete('my-new-bucket1')
```

Save the file and exit the editor.

2. Make the file executable:

```
$ chmod +x del_empty_bucket.rb
```

3. Run the file:

```
$ ./del_empty_bucket.rb | echo $?
```

If the bucket is successfully deleted, the command will return **0** as output.

NOTE

Please edit the **create_bucket.rb** file to create empty buckets like **my-new-bucket9**, **my-new-bucket10** etc and edit the above mentioned **del_empty_bucket.rb** file accordingly before trying to delete empty buckets.

Deleting a non-empty bucket (forcefully)

1. Create a new file:

```
$ vim del_non_empty_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Bucket.delete('my-new-bucket1', :force => true)
```

Save the file and exit the editor.

2. Make the file executable:

```
$ chmod +x del_non_empty_bucket.rb
```

3. Run the file:

```
$ ./del_non_empty_bucket.rb | echo $?
```

If the bucket is successfully deleted, the command will return **0** as output.

Deleting an object

1. Create a new file:

```
$ vim delete_object.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::S3Object.delete('hello.txt', 'my-new-bucket1')
```

Save the file and exit the editor.

2. Make the file executable:

```
$ chmod +x delete_object.rb
```

3. Run the file:

```
$ ./delete_object.rb
```

This will delete the object **hello.txt**.

2.4.3. Ruby AWS::SDK Examples (aws-sdk gem ~>2)

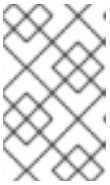
You can use the Ruby programming language along with **aws-sdk** gem for **S3** access. Execute the steps mentioned below on the node used for accessing the Ceph Object Gateway server with **Ruby AWS::SDK**.

Setup Ruby

Execute the following steps to setup Ruby:

1. As **root**, install **ruby**:

```
# yum install ruby
```



NOTE

The above command will install **ruby** and its essential dependencies such as **rubygems** and **ruby-libs**. If somehow the command doesn't install all the dependencies, install them separately.

2. As **root**, install **aws-sdk**:

```
# gem install aws-sdk
```

Creating a connection

1. Create a project directory:

```
$ mkdir ruby_aws_sdk
$ cd ruby_aws_sdk
```

2. Create the connection file:

```
$ vim conn.rb
```

3. Paste the following contents into the **conn.rb** file:

```
#!/usr/bin/env ruby

require 'aws-sdk'
require 'resolv-replace'

Aws.config.update(
        endpoint: 'http://<FQDN_of_gateway_node>:8080',
        access_key_id: 'my-access-key',
        secret_access_key: 'my-secret-key',
        force_path_style: true,
        region: 'us-east-1'
)
```

Replace **<FQDN_of_gateway_node>** with the FQDN of your gateway node. Replace **my-access-key** and **my-secret-key** with the **access_key** and **secret_key** that were generated when you created the **radosgw** user for **S3** access as mentioned in the *Red Hat Ceph Storage Object Gateway Guide*.

An example connection file looks like the following:

```
#!/usr/bin/env ruby

require 'aws-sdk'
require 'resolv-replace'
```

```
Aws.config.update(
        endpoint: 'http://testclient.englab.pnq.redhat.com:8080',
        access_key_id: '98J4R9P22P5CDL65HKP8',
        secret_access_key: '6C+jcaP0dp0+FZfrRNgyGA9EzRy25pURldwje049',
        force_path_style: true,
        region: 'us-east-1'
)
```

Save the file and exit the editor.

4. Make the file executable:

```
chmod +x conn.rb
```

5. Run the file:

```
./conn.rb | echo $?
```

If you have provided the values correctly in the file, the output of the command will be **0**.

Creating a bucket

1. Create a new file:

```
vim create_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

s3_client = Aws::S3::Client.new
s3_client.create_bucket(bucket: 'my-new-bucket2')
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x create_bucket.rb
```

3. Run the file:

```
./create_bucket.rb
```

If the output of the command is **true** it would mean that bucket **my-new-bucket2** was created successfully.

Listing owned buckets

1. Create a new file:

```
vim list_owned_buckets.rb
```

Paste the following content into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

s3_client = Aws::S3::Client.new
s3_client.list_buckets.buckets.each do |bucket|
        puts "#{bucket.name}\t#{bucket.creation_date}"
end
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x list_owned_buckets.rb
```

3. Run the file:

```
./list_owned_buckets.rb
```

The output should look something like this:

```
my-new-bucket2 2016-01-21 10:33:19 UTC
```

Creating an object

1. Create a new file:

```
vim create_object.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

s3_client = Aws::S3::Client.new
s3_client.put_object(
        key: 'hello.txt',
        body: 'Hello World!',
        bucket: 'my-new-bucket2',
        content_type: 'text/plain'
)
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x create_object.rb
```

3. Run the file:

```
./create_object.rb
```

This will create a file **hello.txt** with the string **Hello World!**.

Listing a Bucket's Content

1. Create a new file:

```
vim list_bucket_content.rb
```

Paste the following content into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

s3_client = Aws::S3::Client.new
s3_client.list_objects(bucket: 'my-new-bucket2').contents.each do |object|
        puts "#{object.key}\t#{object.size}"
end
```

Save the file and exit the editor.

2. Make the file executable.

```
chmod +x list_bucket_content.rb
```

3. Run the file:

```
./list_bucket_content.rb
```

The output will look something like this:

```
hello.txt    12    Fri, 22 Jan 2016 15:54:52 GMT
```

Deleting a empty bucket

1. Create a new file:

```
vim del_empty_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

s3_client = Aws::S3::Client.new
s3_client.delete_bucket(bucket: 'my-new-bucket2')
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x del_empty_bucket.rb
```

3. Run the file:

```
./del_empty_bucket.rb | echo $?
```

If the bucket is successfully deleted, the command will return **0** as output.

NOTE

Please edit the **create_bucket.rb** file to create empty buckets like **my-new-bucket6**, **my-new-bucket7** etc and edit the above mentioned **del_empty_bucket.rb** file accordingly before trying to delete empty buckets.

Deleting a non-empty bucket (forcefully)

1. Create a new file:

```
vim del_non_empty_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

s3_client = Aws::S3::Client.new
Aws::S3::Bucket.new('my-new-bucket2', client: s3_client).clear!
s3_client.delete_bucket(bucket: 'my-new-bucket2')
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x del_non_empty_bucket.rb
```

3. Run the file:

```
./del_non_empty_bucket.rb | echo $?
```

If the bucket is successfully deleted, the command will return **0** as output.

Deleting an object

1. Create a new file:

```
vim delete_object.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby
```

```
load 'conn.rb'

s3_client = Aws::S3::Client.new
s3_client.delete_object(key: 'hello.txt', bucket: 'my-new-bucket2')
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x delete_object.rb
```

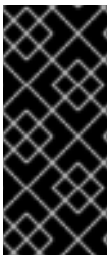
3. Run the file:

```
./delete_object.rb
```

This will delete the object **hello.txt**.

2.4.4. PHP S3 Examples

You can use PHP scripts too for S3 access. Execute the steps mentioned below on the node used for accessing the Ceph Object Gateway server with PHP.



IMPORTANT

The examples given below are tested against **php v5.4.16** and **aws-sdk v2.8.24**. **DO NOT** use the latest version of **aws-sdk** for **php** as it requires **php >= 5.5+**. **php 5.5** is not available in the default repos of **RHEL 7**. If you want to use **php 5.5**, you will have to enable **epel** and other third party repos. Also, the configuration options for **php 5.5** and latest version of **aws-sdk** are different.

Setup PHP/AWS SDK

Execute the following steps to set up PHP:

1. As **root**, install **php**:

```
# yum install php
```

2. Install **aws-sdk** for php:
Download the zip archive of **aws-sdk** for php and extract it.

Creating a connection

1. Create a project directory:

```
$ mkdir php_s3
$ cd php_s3
```

2. Copy the extracted **aws** directory to the project directory. For example:

```
$ cp -r ~/Downloads/aws/ ~/php_s3/
```

3. Create the connection file:

```
$ vim conn.php
```

4. Paste the following contents in the **conn.php** file:

```
<?php
define('AWS_KEY', 'my_access_key');
define('AWS_SECRET_KEY', 'my_secret_key');
define('HOST', '<FQDN_of_gateway_node>');
define('PORT', '8080');

// require the AWS SDK for php library
require '/path_to_aws/aws-autoloader.php';

use Aws\S3\S3Client;

// Establish connection with host using S3 Client
$client = S3Client::factory(array(
    'base_url' => HOST,
    'port' => PORT,
    'key'      => AWS_KEY,
    'secret'   => AWS_SECRET_KEY
));
?>
```

Replace **<FQDN_of_gateway_node>** with the FQDN of the gateway node. Replace **my-access-key** and **my-secret-key** with the **access_key** and **secret_key** that were generated when you created the **radosgw** user for **S3** access as mentioned in the *Red Hat Ceph Storage Object Gateway Guide*. Also, replace **path_to_aws** with absolute path to the extracted **aws** directory that you copied to the **php** project directory.

An example connection file will look like the following:

```
<?php
define('AWS_KEY', '{key}');
define('AWS_SECRET_KEY', '{secret}');
define('HOST', 'http://{hostname}');

// require the AWS SDK for php library
require '/home/ceph/php_s3/aws/aws-autoloader.php';

use Aws\S3\S3Client;

// Establish connection with host using S3 Client
$client = S3Client::factory(array(
    'base_url' => HOST,
    'port' => PORT,
    'key'      => AWS_KEY,
    'secret'   => AWS_SECRET_KEY
));
?>
```

Save the file and exit the editor.

5. Run the file:

```
$ php -f conn.php | echo $?
```

If you have provided the values correctly in the file, the output of the command will be **0**.

Creating a bucket

1. Create a new file:

```
vim create_bucket.php
```

Paste the following contents into the file:

```
<?php

include 'conn.php';

$client->createBucket(array('Bucket' => 'my-new-bucket3'));

?>
```

Save the file and exit the editor.

2. Run the file:

```
php -f create_bucket.php
```

Listing owned buckets

1. Create a new file:

```
vim list_owned_buckets.php
```

Paste the following content into the file:

```
<?php

include 'conn.php';

$blist = $client->listBuckets();
echo "    Buckets belonging to " . $blist['Owner']['ID'] . ":\n";
foreach ($blist['Buckets'] as $b) {
    echo "{$b['Name']}\t{$b['CreationDate']}\n";
}

?>
```

Save the file and exit the editor.

2. Run the file:

```
php -f list_owned_buckets.php
```

The output should look something like this:

```
my-new-bucket3 2016-01-21 10:33:19 UTC
```

Creating an object

1. Create a source file **hello.txt**:

```
echo "Hello World!" > hello.txt
```

2. Create a new php file:

```
vim create_object.php
```

Paste the following contents into the file:

```
<?php

include 'conn.php';

$key         = 'hello.txt';
$source_file = './hello.txt';
$acl         = 'private';
$bucket      = 'my-new-bucket3';
$client->upload($bucket, $key, fopen($source_file, 'r'), $acl);

?>
```

Save the file and exit the editor.

3. Run the file:

```
php -f create_object.php
```

This will create the object **hello.txt** in bucket **my-new-bucket3**.

Listing a Bucket's Content

1. Create a new file:

```
vim list_bucket_content.php
```

Paste the following content into the file:

```
<?php

include 'conn.php';

$o_iter = $client->getIterator('ListObjects', array(
    'Bucket' => 'my-new-bucket3'
));
foreach ($o_iter as $o) {
```

```
    echo "{$o['Key']}\t{$o['Size']}\t{$o['LastModified']}\n";
}
?>
```

Save the file and exit the editor.

2. Run the file:

```
php -f list_bucket_content.php
```

The output will look something like this:

```
hello.txt    12    Fri, 22 Jan 2016 15:54:52 GMT
```

Deleting an empty bucket

1. Create a new file:

```
vim del_empty_bucket.php
```

Paste the following contents into the file:

```
<?php

include 'conn.php';

$client->deleteBucket(array('Bucket' => 'my-new-bucket3'));
?>
```

Save the file and exit the editor.

2. Run the file:

```
php -f del_empty_bucket.php | echo $?
```

If the bucket is successfully deleted, the command will return **0** as output.

NOTE

Edit the **create_bucket.php** file to create empty buckets like **my-new-bucket4**, **my-new-bucket5**, and so on, and then edit the **del_empty_bucket.php** file accordingly before trying to delete empty buckets.

Deleting a non-empty bucket (forcefully)

Deleting a non-empty bucket is not currently supported in **php 2** and newer versions of **aws-sdk**.

Deleting an object

1. Create a new file:

```
vim delete_object.php
```

Paste the following contents into the file:

```
<?php

include 'conn.php';

$client->deleteObject(array(
    'Bucket' => 'my-new-bucket3',
    'Key'    => 'hello.txt',
));
?>
```

Save the file and exit the editor.

2. Run the file:

```
php -f delete_object.php
```

This will delete the object **hello.txt**.

2.4.5. Configuring and using STS Lite with Keystone (Technology Preview)

The Amazon Secure Token Service (STS) and S3 APIs co-exist in the same namespace. The STS options can be configured in conjunction with the Keystone options.

NOTE

Both S3 and STS APIs can be accessed using the same endpoint in Ceph Object Gateway.

Prerequisites

- Red Hat Ceph Storage 3.2 or higher.
- A running Ceph Object Gateway.
- Installation of the Boto Python module, version 3 or higher.

Procedure

1. Open and edit the **group_vars/rgws.yml** file with the following options:

```
rgw_sts_key = $STS_KEY_FOR_ENCRYPTING_SESSION_TOKEN
rgw_s3_auth_use_sts = true
```

2. Rerun the Ansible playbook:

```
[user@admin ceph-ansible]$ ansible-playbook site.yml --limit rgws
```

3. Generate the EC2 credentials:

Example

```
$ openstack ec2 credentials create
+------------+---------------------------------------------------------+
| Field      | Value                                                   |
+------------+---------------------------------------------------------+
| access     | b924dfc87d454d15896691182fdeb0ef                        |
| links      | {u'self': u'http://192.168.0.15/identity/v3/users/      |
|            | 40a7140e424f493d8165abc652dc731c/credentials/           |
|            | OS-EC2/b924dfc87d454d15896691182fdeb0ef'}               |
| project_id | c703801dccaf4a0aaa39bec8c481e25a                        |
| secret     | 6a2142613c504c42a94ba2b82147dc28                        |
| trust_id   | None                                                    |
| user_id    | 40a7140e424f493d8165abc652dc731c                        |
+------------+---------------------------------------------------------+
```

4. Use the generated credentials to get back a set of temporary security credentials using *GetSessionToken* API.

Example

```
import boto3

access_key = b924dfc87d454d15896691182fdeb0ef
secret_key = 6a2142613c504c42a94ba2b82147dc28

client = boto3.client('sts',
aws_access_key_id=access_key,
aws_secret_access_key=secret_key,
endpoint_url=https://www.example.com/rgw,
region_name='',
)

response = client.get_session_token(
    DurationSeconds=43200
)
```

5. Obtaining the temporary credentials can be used for making S3 calls:

Example

```
    s3client = boto3.client('s3',
      aws_access_key_id = response['Credentials']['AccessKeyId'],
      aws_secret_access_key = response['Credentials']['SecretAccessKey'],
      aws_session_token = response['Credentials']['SessionToken'],
      endpoint_url=https://www.example.com/s3,
      region_name='')

    bucket = s3client.create_bucket(Bucket='my-new-shiny-bucket')
    response = s3client.list_buckets()
    for bucket in response["Buckets"]:
        print "{name}\t{created}".format(
                name = bucket['Name'],
                created = bucket['CreationDate'],
    )
```

6. Create a new *S3Access* role and configure a policy.

   a. Assign a user with administrative CAPS:

   ```
   radosgw-admin caps add --uid="$USER" --caps="roles=*"
   ```

   Example

   ```
   [user@client]$ radosgw-admin caps add --uid="gwadmin" --caps="roles=*"
   ```

   b. Create the *S3Access* role:

   ```
   radosgw-admin role create --role-name=$ROLE_NAME --path=$PATH --assume-role-policy-doc=$TRUST_POLICY_DOC
   ```

   Example

   ```
   [user@client]$ radosgw-admin role create --role-name=S3Access --path=/application_abc/component_xyz/ --assume-role-policy-doc=\{\"Version\":\"2012-10-17\",\"Statement\":\[\{\"Effect\":\"Allow\",\"Principal\":\{\"AWS\":\[\"arn:aws:iam:::user/TESTER\"\]\},\"Action\":\[\"sts:AssumeRole\"\]\}\]\}
   ```

   c. Attach a permission policy to the *S3Access* role:

   ```
   radosgw-admin role-policy put --role-name=$ROLE_NAME --policy-name=$POLICY_NAME --policy-doc=$PERMISSION_POLICY_DOC
   ```

   Example

   ```
   [user@client]$ radosgw-admin role-policy put --role-name=S3Access --policy-name=Policy --policy-doc=\{\"Version\":\"2012-10-17\",\"Statement\":\[\{\"Effect\":\"Allow\",\"Action\":\[\"s3:*\"\],\"Resource\":\"arn:aws:s3:::example_bucket\"\}\]\}
   ```

   d. Now another user can assume the role of the **gwadmin** user. For example, the **gwuser** user can assume the permissions of the **gwadmin** user.

   e. Make a note of the assuming user's **access_key** and **secret_key** values.

   Example

   ```
   [user@client]$ radosgw-admin user info --uid=gwuser | grep -A1 access_key
   ```

7. Use the *AssumeRole* API call, providing the **access_key** and **secret_key** values from the assuming user:

   Example

   ```
   import boto3

   access_key = 11BS02LGFB6AL6H1ADMW
   secret_key = vzCEkuryfn060dfee4fgQPqFrncKEIkh3ZcdOANY
   ```

```
client = boto3.client('sts',
aws_access_key_id=access_key,
aws_secret_access_key=secret_key,
endpoint_url=https://www.example.com/rgw,
region_name='',
)

response = client.assume_role(
RoleArn='arn:aws:iam:::role/application_abc/component_xyz/S3Access',
RoleSessionName='Bob',
DurationSeconds=3600
)
```

IMPORTANT

The *AssumeRole* API requires the *S3Access* role.

Additional Resources

- See the *Test S3 Access* section in the Red Hat Ceph Storage *Object Gateway Guide* for more information on installing the Boto Python module.
- See the *Create a User* section in the Red Hat Ceph Storage *Object Gateway Guide* for more information.

2.4.6. Working around the limitations of using STS Lite with Keystone (Technology Preview)

A limitation with Keystone is that it does not supports STS requests. Another limitation is the payload hash is not included with the request. To work around these two limitations the Boto authentication code must be modified.

Prerequisites

- Red Hat Ceph Storage 3.2 or higher.
- A running Ceph Object Gateway.
- Installation of Boto Python module, version 3 or higher.

Procedure

1. Open and edit Boto's **auth.py** file.
   a. Add the following four lines to the code block:

```
class SigV4Auth(BaseSigner):
  """
  Sign a request with Signature V4.
  """
  REQUIRES_REGION = True

  def __init__(self, credentials, service_name, region_name):
     self.credentials = credentials
     # We initialize these value here so the unit tests can have
```

```
        # valid values.  But these will get overriden in ``add_auth``
        # later for real requests.
        self._region_name = region_name
        if service_name == 'sts': (1)
            self._service_name = 's3' (2)
        else: (3)
            self._service_name = service_name (4)
```

b. Add the following two lines to the code block:

```
def _modify_request_before_signing(self, request):
        if 'Authorization' in request.headers:
            del request.headers['Authorization']
        self._set_necessary_date_headers(request)
        if self.credentials.token:
            if 'X-Amz-Security-Token' in request.headers:
                del request.headers['X-Amz-Security-Token']
            request.headers['X-Amz-Security-Token'] = self.credentials.token

        if not request.context.get('payload_signing_enabled', True):
            if 'X-Amz-Content-SHA256' in request.headers:
                del request.headers['X-Amz-Content-SHA256']
            request.headers['X-Amz-Content-SHA256'] = UNSIGNED_PAYLOAD (1)
        else: (2)
            request.headers['X-Amz-Content-SHA256'] = self.payload(request)
```

Additional Resources

- See the *Test S3 Access* section in the Red Hat Ceph Storage *Object Gateway Guide* for more information on installing the Boto Python module.

2.5. COMMON OPERATIONS

2.5.1. Bucket and Host Name

There are two different modes of accessing the buckets. The first, and preferred method identifies the bucket as the top-level directory in the URI.

Example

```
GET /mybucket HTTP/1.1
Host: cname.domain.com
```

The second method identifies the bucket via a virtual bucket host name.

Example

```
GET / HTTP/1.1
Host: mybucket.cname.domain.com
```

TIP

Red Hat prefers the first method, because the second method requires expensive domain certification and DNS wild cards.

2.5.2. Common Request Headers

The following table lists the valid common request headers and their descriptions.

Table 2.4. Request Headers

Request Header	Description
CONTENT_LENGTH	Length of the request body.
DATE	Request time and date (in UTC).
HOST	The name of the host server.
AUTHORIZATION	Authorization token.

2.5.3. Common Response Status

The following table lists the valid common HTTP response status and its corresponding code.

Table 2.5. Response Status

HTTP Status	Response Code
100	Continue
200	Success
201	Created
202	Accepted
204	NoContent
206	Partial content
304	NotModified
400	InvalidArgument
400	InvalidDigest
400	BadDigest

HTTP Status	Response Code
400	InvalidBucketName
400	InvalidObjectName
400	UnresolvableGrantByEmailAddress
400	InvalidPart
400	InvalidPartOrder
400	RequestTimeout
400	EntityTooLarge
403	AccessDenied
403	UserSuspended
403	RequestTimeTooSkewed
404	NoSuchKey
404	NoSuchBucket
404	NoSuchUpload
405	MethodNotAllowed
408	RequestTimeout
409	BucketAlreadyExists
409	BucketNotEmpty
411	MissingContentLength
412	PreconditionFailed
416	InvalidRange
422	UnprocessableEntity
500	InternalError

2.6. SERVICE OPERATIONS

2.6.1. List Buckets

GET / returns a list of buckets created by the user making the request. **GET** / only returns buckets created by an authenticated user. You cannot make an anonymous request.

Syntax

```
GET / HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Table 2.6. Response Entities

Name	Type	Description
Buckets	Container	Container for list of buckets.
Bucket	Container	Container for bucket information.
Name	String	Bucket name.
CreationDate	Date	UTC time when the bucket was created.
ListAllMyBucketsResult	Container	A container for the result.
Owner	Container	A container for the bucket owner's **ID** and **DisplayName**.
ID	String	The bucket owner's ID.
DisplayName	String	The bucket owner's display name.

Return to the features table.

2.7. BUCKET OPERATIONS

2.7.1. Bucket Operations with Multi Tenancy

When a client application accesses buckets, it always operates with credentials of a particular user. In Red Hat Ceph Storage 3, every user belongs to a tenant. See Multi Tenancy for additional details. Consequently, every bucket operation has an implicit tenant in its context if no tenant is specified explicitly. Thus multi tenancy is completely backward compatible with previous releases, as long as the referred buckets and referring user belong to the same tenant.

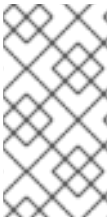
Extensions employed to specify an explicit tenant differ according to the protocol and authentication system used.

In the following example, a colon character separates tenant and bucket. Thus a sample URL would be:

```
https://rgw.domain.com/tenant:bucket
```

By contrast, a simple Python example separates the tenant and bucket in the bucket method itself:

```
from boto.s3.connection import S3Connection, OrdinaryCallingFormat
 c = S3Connection(
  aws_access_key_id="TESTER",
  aws_secret_access_key="test123",
  host="rgw.domain.com",
  calling_format = OrdinaryCallingFormat()
 )
 bucket = c.get_bucket("tenant:bucket")
```



NOTE

It's not possible to use S3-style subdomains using multi-tenancy, since host names cannot contain colons or any other separators that are not already valid in bucket names. Using a period creates an ambiguous syntax. Therefore, the **bucket-in-URL-path** format has to be used with multi-tenancy.

2.7.2. Bucket Lifecycle

You can use a bucket lifecycle configuration to manage your objects so they are stored effectively throughout their lifetime. The S3 API in the Ceph Object Gateway supports a subset of the AWS bucket lifecycle actions:

- **Expiration**: This defines the lifespan of objects within a bucket. It takes the number of days the object should live or an expiration date, at which point Ceph Object Gateway will delete the object. If the bucket doesn't enable versioning, Ceph Object Gateway will delete the object permanently. If the bucket enables versioning, Ceph Object Gateway will create a delete marker for the current version, and then delete the current version.

- **NoncurrentVersionExpiration**: This defines the lifespan of non-current object versions within a bucket. To use this feature, the bucket must enable versioning. It takes the number of days a non-current object should live, at which point Ceph Object Gateway will delete the non-current object.

- **AbortIncompleteMultipartUpload**: This defines the number of days an incomplete multipart upload should live before it is aborted.

The lifecycle configuration contains one or more rules using the **<Rule>** element. For example:

```
<LifecycleConfiguration>
   <Rule>
     <Prefix/>
     <Status>Enabled</Status>
     <Expiration>
       <Days>10</Days>
     </Expiration>
   </Rule>
</LifecycleConfiguration>
```

A lifecycle rule can apply to all or a subset of objects in a bucket based on the **<Filter>** element that you specify in the lifecycle rule. You can specify a filter several ways:

- Key prefixes
- Object tags
- Both key prefix and one or more object tags

Key prefixes

You can apply a lifecycle rule to a subset of objects based on the key name prefix. For example, specifying **<keypre/>** would apply to objects that begin with **keypre/**:

```
<LifecycleConfiguration>
   <Rule>
   <Status>Enabled</Status>
      <Filter>
         <Prefix>keypre/</Prefix>
      </Filter>
   </Rule>
</LifecycleConfiguration>
```

You can also apply different lifecycle rules to objects with different key prefixes:

```
<LifecycleConfiguration>
   <Rule>
   <Status>Enabled</Status>
      <Filter>
         <Prefix>keypre/</Prefix>
      </Filter>
   </Rule>
   <Rule>
      <Filter>
         <Prefix>mypre/</Prefix>
      </Filter>
   </Rule>
</LifecycleConfiguration>
```

Object tags

You can apply a lifecycle rule to only objects with a specific tag using the **<Key>** and **<Value>** elements:

```
<LifecycleConfiguration>
   <Rule>
   <Status>Enabled</Status>
      <Filter>
         <Tag>
            <Key>key</Key>
            <Value>value</Value>
         </Tag>
      </Filter>
   </Rule>
</LifecycleConfiguration>
```

Both prefix and one or more tags

In a lifecycle rule, you can specify a filter based on both the key prefix and one or more tags. They must be wrapped in the **<And>** element. A filter can have only one prefix, and zero or more tags:

```
<LifecycleConfiguration>
   <Rule>
   <Status>Enabled</Status>
      <Filter>
       <And>
         <Prefix>key-prefix</Prefix>
         <Tag>
           <Key>key1</Key>
           <Value>value1</Value>
         </Tag>
         <Tag>
           <Key>key2</Key>
           <Value>value2</Value>
         </Tag>
          ...
       </And>
      </Filter>
      <Status>Enabled</Status>
   </Rule>
</LifecycleConfiguration>
```

For additional details, see:

- Put Bucket Lifecycle
- Get Bucket Lifecycle
- Delete Bucket Lifecycle

2.7.3. Head Bucket

Calls HEAD on a bucket to determine if it exists and if the caller has access permissions. Returns **200 OK** if the bucket exists and the caller has permissions; **404 Not Found** if the bucket does not exist; and, **403 Forbidden** if the bucket exists but the caller does not have access permissions.

Syntax

```
HEAD /<bucket> HTTP/1.1
Host: cname.domain.com
Date: date
Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

2.7.4. PUT Bucket

Creates a new bucket. To create a bucket, you must have a user ID and a valid AWS Access Key ID to authenticate requests. You can not create buckets as an anonymous user.

Constraints

In general, bucket names should follow domain name constraints.

- Bucket names must be unique.

- Bucket names must begin and end with a lowercase letter.
- Bucket names can contain a dash (-).

Syntax

```
PUT /<bucket> HTTP/1.1
Host: cname.domain.com
x-amz-acl: public-read-write

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Table 2.7. Parameters

Name	Description	Valid Values	Required
x-amz-acl	Canned ACLs.	**private**, **public-read**,**public-read-write**, **authenticated-read**	No

HTTP Response

If the bucket name is unique, within constraints and unused, the operation will succeed. If a bucket with the same name already exists and the user is the bucket owner, the operation will succeed. If the bucket name is already in use, the operation will fail.

HTTP Status	Status Code	Description
409	BucketAlreadyExists	Bucket already exists under different user's ownership.

Return to the features table.

2.7.5. Put Bucket Lifecycle

To create or replace a bucket lifecycle, use **PUT** and specify a destination bucket and a lifecycle configuration. The Ceph Object Gateway only supports a subset of the S3 lifecycle functionality. See S3 API Bucket Lifecycle for details.

Syntax

```
PUT /<bucket>?lifecycle HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
<LifecycleConfiguration>
  <Rule>
    <Expiration>
      <Days>10</Days>
    </Expiration>
  </Rule>
    ...
```

```
  <Rule>
  </Rule>
</LifecycleConfiguration>
```

Table 2.8. Request Headers

Name	Description	Valid Values	Required
content-md5	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No

See also Common Request Headers

Return to the features table.

2.7.6. DELETE Bucket

Deletes a bucket. You can reuse bucket names following a successful bucket removal.

Syntax

```
DELETE /<bucket> HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Table 2.9. HTTP Response

HTTP Status	Status Code	Description
204	No Content	Bucket removed.

Return to the features table.

2.7.7. Delete Bucket Lifecycle

To delete a bucket lifecycle, use **DELETE** and specify a destination bucket.

Syntax

```
DELETE /<bucket>?lifecycle HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Request Headers

The request does not contain any special elements.

See Common Request Headers

Response

The response returns common response status.

See Common Response Status for details.

Return to the features table.

2.7.8. GET Bucket

Returns a list of bucket objects.

Syntax

```
GET /<bucket>?max-keys=25 HTTP/1.1
Host: cname.domain.com
```

Table 2.10. Parameters

Name	Type	Description
prefix	String	Only returns objects that contain the specified prefix.
delimiter	String	The delimiter between the prefix and the rest of the object name.
marker	String	A beginning index for the list of objects returned.
max-keys	Integer	The maximum number of keys to return. Default is 1000.

Table 2.11. HTTP Response

HTTP Status	Status Code	Description
200	OK	Buckets retrieved

GET /<bucket> returns a container for buckets with the following fields:

Table 2.12. Bucket Response Entities

Name	Type	Description
ListBucketResult	Entity	The container for the list of objects.
Name	String	The name of the bucket whose contents will be returned.
Prefix	String	A prefix for the object keys.
Marker	String	A beginning index for the list of objects returned.

Name	Type	Description
MaxKeys	Integer	The maximum number of keys returned.
Delimiter	String	If set, objects with the same prefix will appear in the **CommonPrefixes** list.
IsTruncated	Boolean	If **true**, only a subset of the bucket's contents were returned.
CommonPrefixes	Container	If multiple objects contain the same prefix, they will appear in this list.

The **ListBucketResult** contains objects, where each object is within a **Contents** container.

Table 2.13. Object Response Entities

Name	Type	Description
Contents	Object	A container for the object.
Key	String	The object's key.
LastModified	Date	The object's last-modified date/time.
ETag	String	An MD-5 hash of the object. (entity tag)
Size	Integer	The object's size.
StorageClass	String	Should always return **STANDARD**.

Return to the features table.

2.7.9. Get Bucket Lifecycle

To get a bucket lifecycle, use **GET** and specify a destination bucket.

Syntax

```
GET /<bucket>?lifecycle HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Request Headers

See Common Request Headers

Response

The response contains the bucket lifecycle and its elements.

Return to the features table.

2.7.10. Get Bucket Location

Retrieves the bucket's zone group. The user needs to be the bucket owner to call this. A bucket can be constrained to a zone group by providing **LocationConstraint** during a PUT request.

Add the **location** subresource to bucket resource as shown below.

Syntax

```
GET /<bucket>?location HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Table 2.14. Response Entities

Name	Type	Description
LocationConstraint	String	The zone group where bucket resides, empty string for defult zone group

Return to the features table.

2.7.11. Get Bucket Versioning

Retrieves the versioning state of a bucket. The user needs to be the bucket owner to call this.

Add the **versioning** subresource to bucket resource as shown below.

Syntax

```
GET /<bucket>?versioning HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Return to the features table.

2.7.12. PUT Bucket Versioning

This subresource set the versioning state of an existing bucket. The user needs to be the bucket owner to set the versioning state. If the versioning state has never been set on a bucket, then it has no versioning state. Doing a GET versioning request does not return a versioning state value.

Setting the bucket versioning state:

Enabled : Enables versioning for the objects in the bucket. All objects added to the bucket receive a unique version ID. **Suspended** : Disables versioning for the objects in the bucket. All objects added to the bucket receive the version ID null.

Syntax

```
PUT /<bucket>?versioning HTTP/1.1
```

Table 2.15. Bucket Request Entities

Name	Type	Description
VersioningConfiguration	container	A container for the request.
Status	String	Sets the versioning state of the bucket. Valid Values: Suspended/Enabled

Return to the features table.

2.7.13. Get Bucket ACLs

Retrieves the bucket access control list. The user needs to be the bucket owner or to have been granted **READ_ACP** permission on the bucket.

Add the **acl** subresource to the bucket request as shown below.

Syntax

```
GET /<bucket>?acl HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Table 2.16. Response Entities

Name	Type	Description
AccessControlPolicy	Container	A container for the response.
AccessControlList	Container	A container for the ACL information.
Owner	Container	A container for the bucket owner's **ID** and **DisplayName**.
ID	String	The bucket owner's ID.
DisplayName	String	The bucket owner's display name.
Grant	Container	A container for **Grantee** and **Permission**.

Name	Type	Description
Grantee	Container	A container for the **DisplayName** and **ID** of the user receiving a grant of permission.
Permission	String	The permission given to the **Grantee** bucket.

Return to the features table.

2.7.14. PUT Bucket ACLs

Sets an access control to an existing bucket. The user needs to be the bucket owner or to have been granted **WRITE_ACP** permission on the bucket.

Add the **acl** subresource to the bucket request as shown below.

Syntax

```
PUT /<bucket>?acl HTTP/1.1
```

Table 2.17. Request Entities

Name	Type	Description
AccessControlPolicy	Container	A container for the request.
AccessControlList	Container	A container for the ACL information.
Owner	Container	A container for the bucket owner's **ID** and **DisplayName**.
ID	String	The bucket owner's ID.
DisplayName	String	The bucket owner's display name.
Grant	Container	A container for **Grantee** and **Permission**.
Grantee	Container	A container for the **DisplayName** and **ID** of the user receiving a grant of permission.
Permission	String	The permission given to the **Grantee** bucket.

2.7.15. GET Bucket cors

Retrieves the cors configuration information set for the bucket. The user needs to be the bucket owner or to have been granted **READ_ACP** permission on the bucket.

Add the **cors** subresource to the bucket request as shown below.

Syntax

```
GET /<bucket>?cors HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Return to the features table.

2.7.16. PUT Bucket cors

Sets the cors configuration for the bucket. The user needs to be the bucket owner or to have been granted **READ_ACP** permission on the bucket.

Add the **cors** subresource to the bucket request as shown below.

Syntax

```
PUT /<bucket>?cors HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

2.7.17. DELETE Bucket cors

Deletes the cors configuration information set for the bucket. The user needs to be the bucket owner or to have been granted **READ_ACP** permission on the bucket.

Add the **cors** subresource to the bucket request as shown below.

Syntax

```
DELETE /<bucket>?cors HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

2.7.18. List Bucket Object Versions

Returns a list of metadata about all the version of objects within a bucket. Requires READ access to the bucket.

Add the **versions** subresource to the bucket request as shown below.

Syntax

```
GET /<bucket>?versions HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

You can specify parameters for **GET /<bucket>?versions**, but none of them are required.

Table 2.18. Parameters

Name	Type	Description
prefix	String	Returns in-progress uploads whose keys contains the specified prefix.
delimiter	String	The delimiter between the prefix and the rest of the object name.
key-marker	String	The beginning marker for the list of uploads.
max-keys	Integer	The maximum number of in-progress uploads. The default is 1000.
version-id-marker	String	Specifies the object version to begin the list.

Table 2.19. Response Entities

Name	Type	Description
KeyMarker	String	The key marker specified by the **key-marker** request parameter (if any).
NextKeyMarker	String	The key marker to use in a subsequent request if **IsTruncated** is **true**.
NextUploadIdMarker	String	The upload ID marker to use in a subsequent request if **IsTruncated** is **true**.
IsTruncated	Boolean	If **true**, only a subset of the bucket's upload contents were returned.
Size	Integer	The size of the uploaded part.
DisplayName	String	The owners's display name.
ID	String	The owners's ID.
Owner	Container	A container for the **ID** and **DisplayName** of the user who owns the object.
StorageClass	String	The method used to store the resulting object. **STANDARD** or **REDUCED_REDUNDANCY**

Name	Type	Description
Version	Container	Container for the version information.
versionId	String	Version ID of an object.
versionIdMarker	String	The last version of the key in a truncated response.

Return to the features table.

2.7.19. List Bucket Multipart Uploads

GET /?uploads returns a list of the current in-progress multipart uploads, that is, the application initiates a multipart upload, but the service hasn't completed all the uploads yet.

Syntax

```
GET /<bucket>?uploads HTTP/1.1
```

You can specify parameters for **GET /<bucket>?uploads**, but none of them are required.

Table 2.20. Parameters

Name	Type	Description
prefix	String	Returns in-progress uploads whose keys contains the specified prefix.
delimiter	String	The delimiter between the prefix and the rest of the object name.
key-marker	String	The beginning marker for the list of uploads.
max-keys	Integer	The maximum number of in-progress uploads. The default is 1000.
max-uploads	Integer	The maximum number of multipart uploads. The range from 1-1000. The default is 1000.
version-id-marker	String	Ignored if **key-marker** isn't specified. Specifies the **ID** of first upload to list in lexicographical order at or following the **ID**.

Table 2.21. Response Entities

Name	Type	Description
ListMultipartUploadsResult	Container	A container for the results.
ListMultipartUploadsResult.Prefix	String	The prefix specified by the **prefix** request parameter (if any).
Bucket	String	The bucket that will receive the bucket contents.
KeyMarker	String	The key marker specified by the **key-marker** request parameter (if any).
UploadIdMarker	String	The marker specified by the **upload-id-marker** request parameter (if any).
NextKeyMarker	String	The key marker to use in a subsequent request if **IsTruncated** is **true**.
NextUploadIdMarker	String	The upload ID marker to use in a subsequent request if **IsTruncated** is **true**.
MaxUploads	Integer	The max uploads specified by the **max-uploads** request parameter.
Delimiter	String	If set, objects with the same prefix will appear in the **CommonPrefixes** list.
IsTruncated	Boolean	If **true**, only a subset of the bucket's upload contents were returned.
Upload	Container	A container for **Key**, **UploadId**, **InitiatorOwner**, **StorageClass**, and **Initiated** elements.
Key	String	The key of the object once the multipart upload is complete.
UploadId	String	The **ID** that identifies the multipart upload.
Initiator	Container	Contains the **ID** and **DisplayName** of the user who initiated the upload.
DisplayName	String	The initiator's display name.
ID	String	The initiator's ID.

Name	Type	Description
Owner	Container	A container for the **ID** and **DisplayName** of the user who owns the uploaded object.
StorageClass	String	The method used to store the resulting object. **STANDARD** or **REDUCED_REDUNDANCY**
Initiated	Date	The date and time the user initiated the upload.
CommonPrefixes	Container	If multiple objects contain the same prefix, they will appear in this list.
CommonPrefixes.Prefix	String	The substring of the key after the prefix as defined by the **prefix** request parameter.

Return to the features table.

2.7.20. PUT Bucket Request Payment

Uses the **requestPayment** subresource to set the request payment configuration of a bucket. By default, the bucket owner pays for downloads from the bucket. This configuration parameter enables the bucket owner to specify that the person requesting the download will be charged for the request and the data download from the bucket.

Add the **requestPayment** subresource to the bucket request as shown below.

Syntax

```
PUT /<bucket>?requestPayment HTTP/1.1
Host: cname.domain.com
```

Table 2.22. Request Entities

Name	Type	Description
Payer	Enum	Specifies who pays for the download and request fees.
RequestPaymentConfiguration	Container	A container for **Payer**.

Return to the features table.

2.7.21. GET Bucket Request Payment

Uses the **requestPayment** subresource to return the request payment configuration of a bucket. The user needs to be the bucket owner or to have been granted **READ_ACP** permission on the bucket.

Add the **requestPayment** subresource to the bucket request as shown below.

Syntax

```
GET /<bucket>?requestPayment HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

2.8. OBJECT OPERATIONS

2.8.1. PUT Object

Adds an object to a bucket. You must have write permissions on the bucket to perform this operation.

Syntax

```
PUT /<bucket>/<object> HTTP/1.1
```

Table 2.23. Request Headers

Name	Description	Valid Values	Required
content-md5	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No
content-type	A standard MIME type.	Any MIME type. Default: **binary/octet-stream**	No
x-amz-meta-<...>	User metadata. Stored with the object.	A string up to 8kb. No defaults.	No
x-amz-acl	A canned ACL.	**private**, **public-read**, **public-read-write**, **authenticated-read**	No

Table 2.24. Response Headers

Name	Description
x-amz-version-id	Returns the version ID or null.

Return to the features table.

2.8.2. Copy Object

To copy an object, use **PUT** and specify a destination bucket and the object name.

Syntax

```
PUT /<dest_bucket>/<dest_object> HTTP/1.1
x-amz-copy-source: <source_bucket>/<source_object>
```

Table 2.25. Request Headers

Name	Description	Valid Values	Required
x-amz-copy-source	The source bucket name + object name.	<bucket>/<object>	Yes
x-amz-acl	A canned ACL.	**private**, **public-read**, **public-read-write**, **authenticated-read**	No
x-amz-copy-if-modified-since	Copies only if modified since the timestamp.	Timestamp	No
x-amz-copy-if-unmodified-since	Copies only if unmodified since the timestamp.	Timestamp	No
x-amz-copy-if-match	Copies only if object ETag matches ETag.	Entity Tag	No
x-amz-copy-if-none-match	Copies only if object ETag doesn't match.	Entity Tag	No

Table 2.26. Response Entities

Name	Type	Description
CopyObjectResult	Container	A container for the response elements.
LastModified	Date	The last modified date of the source object.
Etag	String	The ETag of the new object.

Return to the features table.

2.8.3. POST Object

Adds an object to a bucket using HTML forms. You must have write permissions on the bucket to perform this operation.

Syntax

```
POST /<bucket>/<object> HTTP/1.1
```

Return to the features table.

2.8.4. OPTIONS Object

A preflight request to determine if an actual request can be sent with the specific origin, HTTP method, and headers.

Syntax

```
OPTIONS /<object> HTTP/1.1
```

Return to the features table.

2.8.5. Delete Multiple Objects

Deletes multiple objects from a bucket.

Syntax

```
POST /<bucket>/<object>?delete HTTP/1.1
```

Return to the features table.

2.8.6. Remove Object

Removes an object. Requires WRITE permission set on the containing bucket.

Deletes an object. If object versioning is on, it creates a marker.

Syntax

```
DELETE /<bucket>/<object> HTTP/1.1
```

To delete an object when versioning is on, you must specify the **versionId** subresource and the version of the object to delete.

```
DELETE /<bucket>/<object>?versionId=<versionID> HTTP/1.1
```

Return to the features table.

2.8.7. Get Object

Retrieves an object from a bucket:

Syntax

```
GET /<bucket>/<object> HTTP/1.1
```

Add the **versionId** subresource to retrieve a particular version of the object:

Syntax

```
GET /<bucket>/<object>?versionId=<versionID> HTTP/1.1
```

Table 2.27. Request Headers

Name	Description	Valid Values	Required
range	The range of the object to retrieve.	Range: bytes=beginbyte-endbyte	No
if-modified-since	Gets only if modified since the timestamp.	Timestamp	No
if-unmodified-since	Gets only if not modified since the timestamp.	Timestamp	No
if-match	Gets only if object ETag matches ETag.	Entity Tag	No
if-none-match	Gets only if object ETag matches ETag.	Entity Tag	No

Table 2.28. Response Headers

Name	Description
Content-Range	Data range, will only be returned if the range header field was specified in the request
x-amz-version-id	Returns the version ID or null.

Return to the features table.

2.8.8. Get Object Information

Returns information about an object. This request will return the same header information as with the Get Object request, but will include the metadata only, not the object data payload.

Retrieves the current version of the object:

Syntax

```
HEAD /<bucket>/<object> HTTP/1.1
```

Add the **versionId** subresource to retrieve info for a particular version:

Syntax

```
HEAD /<bucket>/<object>?versionId=<versionID> HTTP/1.1
```

Table 2.29. Request Headers

Name	Description	Valid Values	Required
range	The range of the object to retrieve.	Range: bytes=beginbyte-endbyte	No
if-modified-since	Gets only if modified since the timestamp.	Timestamp	No
if-unmodified-since	Gets only if not modified since the timestamp.	Timestamp	No
if-match	Gets only if object ETag matches ETag.	Entity Tag	No
if-none-match	Gets only if object ETag matches ETag.	Entity Tag	No

Table 2.30. Response Headers

Name	Description
x-amz-version-id	Returns the version ID or null.

Return to the features table.

2.8.9. Get Object ACL

Returns the ACL for the current version of the object:

Syntax

```
GET /<bucket>/<object>?acl HTTP/1.1
```

Add the **versionId** subresource to retrieve the ACL for a particular version:

Syntax

```
GET /<bucket>/<object>versionId=<versionID>&acl HTTP/1.1
```

Table 2.31. Response Headers

Name	Description
x-amz-version-id	Returns the version ID or null.

Table 2.32. Response Entities

Name	Type	Description
AccessControlPolicy	Container	A container for the response.
AccessControlList	Container	A container for the ACL information.
Owner	Container	A container for the object owner's **ID** and **DisplayName**.
ID	String	The object owner's ID.
DisplayName	String	The object owner's display name.
Grant	Container	A container for **Grantee** and **Permission**.
Grantee	Container	A container for the **DisplayName** and **ID** of the user receiving a grant of permission.
Permission	String	The permission given to the **Grantee** object.

Return to the features table.

2.8.10. Set Object ACL

Sets an object ACL for the current version of the object.

Syntax

```
PUT /<bucket>/<object>?acl
```

Table 2.33. Request Entities

Name	Type	Description
AccessControlPolicy	Container	A container for the response.
AccessControlList	Container	A container for the ACL information.
Owner	Container	A container for the object owner's **ID** and **DisplayName**.
ID	String	The object owner's ID.

Name	Type	Description
DisplayName	String	The object owner's display name.
Grant	Container	A container for **Grantee** and **Permission**.
Grantee	Container	A container for the **DisplayName** and **ID** of the user receiving a grant of permission.
Permission	String	The permission given to the **Grantee** object.

2.8.11. Initiate Multipart Upload

Initiates a multi-part upload process. Returns a **UploadId**, which you can specify when adding additional parts, listing parts, and completing or abandoning a multi-part upload.

Syntax

```
POST /<bucket>/<object>?uploads
```

Table 2.34. Request Headers

Name	Description	Valid Values	Required
content-md5	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No
content-type	A standard MIME type.	Any MIME type. Default: **binary/octet-stream**	No
x-amz-meta-<...>	User metadata. Stored with the object.	A string up to 8kb. No defaults.	No
x-amz-acl	A canned ACL.	**private**, **public-read**, **public-read-write**, **authenticated-read**	No

Table 2.35. Response Entities

Name	Type	Description
InitiatedMultipartUploadsResult	Container	A container for the results.
Bucket	String	The bucket that will receive the object contents.

Name	Type	Description
Key	String	The key specified by the **key** request parameter (if any).
UploadId	String	The ID specified by the **upload-id** request parameter identifying the multipart upload (if any).

Return to the features table.

2.8.12. Multipart Upload Part

Adds a part to a multi-part upload.

Specify the **uploadId** subresource and the upload ID to add a part to a multi-part upload:

Syntax

```
PUT /<bucket>/<object>?partNumber=&uploadId=<upload_id> HTTP/1.1
```

The following HTTP response might be returned:

Table 2.36. HTTP Response

HTTP Status	Status Code	Description
404	NoSuchUpload	Specified upload-id does not match any initiated upload on this object

Return to the features table.

2.8.13. List Multipart Upload Parts

Specify the **uploadId** subresource and the upload ID to list the parts of a multi-part upload:

Syntax

```
GET /<bucket>/<object>?uploadId=<upload-id> HTTP/1.1
```

Table 2.37. Response Entities

Name	Type	Description
InitiatedMultipartUploadsResult	Container	A container for the results.
Bucket	String	The bucket that will receive the object contents.

Name	Type	Description
Key	String	The key specified by the **key** request parameter (if any).
UploadId	String	The ID specified by the **upload-id** request parameter identifying the multipart upload (if any).
Initiator	Container	Contains the **ID** and **DisplayName** of the user who initiated the upload.
ID	String	The initiator's ID.
DisplayName	String	The initiator's display name.
Owner	Container	A container for the **ID** and **DisplayName** of the user who owns the uploaded object.
StorageClass	String	The method used to store the resulting object. **STANDARD** or **REDUCED_REDUNDANCY**
PartNumberMarker	String	The part marker to use in a subsequent request if **IsTruncated** is **true**. Precedes the list.
NextPartNumberMarker	String	The next part marker to use in a subsequent request if **IsTruncated** is **true**. The end of the list.
MaxParts	Integer	The max parts allowed in the response as specified by the **max-parts** request parameter.
IsTruncated	Boolean	If **true**, only a subset of the object's upload contents were returned.
Part	Container	A container for **Key**, **Part**, **InitiatorOwner**, **StorageClass**, and **Initiated** elements.
PartNumber	Integer	The identification number of the part.
ETag	String	The part's entity tag.
Size	Integer	The size of the uploaded part.

Return to the features table.

2.8.14. Complete Multipart Upload

Assembles uploaded parts and creates a new object, thereby completing a multipart upload.

Specify the **uploadId** subresource and the upload ID to complete a multi-part upload:

Syntax

```
POST /<bucket>/<object>?uploadId= HTTP/1.1
```

Table 2.38. Request Entities

Name	Type	Description	Required
CompleteMultipartUpload	Container	A container consisting of one or more parts.	Yes
Part	Container	A container for the **PartNumber** and **ETag**.	Yes
PartNumber	Integer	The identifier of the part.	Yes
ETag	String	The part's entity tag.	Yes

Table 2.39. Response Entities

Name	Type	Description
CompleteMultipartUploadResult	Container	A container for the response.
Location	URI	The resource identifier (path) of the new object.
Bucket	String	The name of the bucket that contains the new object.
Key	String	The object's key.
ETag	String	The entity tag of the new object.

Return to the features table.

2.8.15. Abort Multipart Upload

Aborts a multipart upload.

Specify the **uploadId** subresource and the upload ID to abort a multi-part upload:

Syntax

```
DELETE /<bucket>/<object>?uploadId=<upload_id> HTTP/1.1
```

Return to the features table.

2.8.16. Copy Multipart Upload

Uploads a part by copying data from an existing object as data source.

Specify the **uploadId** subresource and the upload ID to perform a multi-part upload copy:

Syntax

```
PUT /<bucket>/<object>?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
Host: cname.domain.com

Authorization: AWS <access_key>:<hash_of_header_and_secret>
```

Table 2.40. Request Headers

Name	Description	Valid Values	Required
x-amz-copy-source	The source bucket name and object name.	<bucket>/<object>	Yes
x-amz-copy-source-range	The range of bytes to copy from the source object.	Range: **bytes=first-last**, where the first and last are the zero-based byte offsets to copy. For example, **bytes=0-9** indicates that you want to copy the first ten bytes of the source.	No

Table 2.41. Response Entities

Name	Type	Description
CopyPartResult	Container	A container for all response elements.
ETag	String	Returns the ETag of the new part.
LastModified	String	Returns the date the part was last modified.

For more information about this feature, see the *Amazon S3 site*.

Return to the features table.

2.9. HADOOP S3A INTEROPERABILITY

For data analytics applications that require Hadoop Distributed File System (HDFS) access, the Ceph Object Gateway can be accessed using the Apache S3A connector for Hadoop. The S3A connector is an open source tool that presents S3 compatible object storage as an HDFS file system with HDFS file system read and write semantics to the applications while data is stored in the Ceph Object Gateway.

Ceph Object Gateway is fully compatible with the S3A connector that ships with Hadoop 2.7.3.

2.10. S3 LIMITATIONS

IMPORTANT

The following limitations should be used with caution. There are implications related to your hardware selections, so you should always discuss these requirements with your Red Hat account team.

- **Maximum object size when using Amazon S3:** Individual Amazon S3 objects can range in size from a minimum of 0B to a maximum of 5TB. The largest object that can be uploaded in a single **PUT** is 5GB. For objects larger than 100MB, you should consider using the Multipart Upload capability.
- **Maximum metadata size when using Amazon S3:** There is no defined limit on the total size of user metadata that can be applied to an object, but a single HTTP request is limited to 16,000.
- **The amount of data overhead Red Hat Ceph Storage produces to store S3 objects and metadata:** The estimate here is 200-300 bytes plus the length of the object name. Versioned objects consume additional space proportional to the number of versions. Also, transient overhead is produced during multi-part upload and other transactional updates, but these overheads are recovered during garbage collection.

CHAPTER 3. OBJECT GATEWAY SWIFT APPLICATION PROGRAMMING INTERFACE (API)

Ceph supports a RESTful API that is compatible with the basic data access model of the Swift API.

The following table describes the support status for current Swift functional features:

Table 3.1. Features

Feature	Status	Remarks
Authentication	Supported	
Get Account Metadata	Supported	No custom metadata
Swift ACLs	Supported	Supports a subset of Swift ACLs
List Containers	Supported	
Delete Container	Supported	
Create Container	Supported	
Get Container Metadata	Supported	
Update Container Metadata	Supported	
Delete Container Metadata	Supported	
List Objects	Supported	
Static Website	Not Supported	
Multi Tenancy	Supported	
Create/Update an Object	Supported	
Create Large Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Copy Object	Supported	
Get Object Metadata	Supported	
Add/Update Object Metadata	Supported	

Feature	Status	Remarks
Temp URL Operations	Supported	
Expiring Objects	Supported	
Object Versioning	Not Supported	
CORS	Not Supported	

3.1. AUTHENTICATION

Swift API requests that require authentication must contain an **X-Storage-Token** authentication token in the request header. The token can be retrieved from Ceph Object Gateway, or from another authenticator. To obtain a token from Ceph Object Gateway, you must create a user.

Syntax

```
# radosgw-admin user create --uid="<user_name>" --display-name="<display_name>"
```

Example

```
# radosgw-admin user create --uid="swift1" --display-name="First Swift User"
```

Return to the features table.

3.1.1. Authentication GET

To authenticate a user, make a request containing an **X-Auth-User** and a **X-Auth-Key** in the header.

Syntax

```
GET /auth HTTP/1.1
Host: swift.radosgwhost.com
X-Auth-User: johndoe
X-Auth-Key: R7UUOLFDI2ZI9PRCQ53K
```

Table 3.2. Request Headers

Name	Description	Type	Required
X-Auth-User	The key Ceph Object Gateway username to authenticate.	String	Yes
X-Auth-Key	The key associated to a Ceph Object Gateway username.	String	Yes

The response from the server should include an **X-Auth-Token** value. The response might also contain a **X-Storage-Url** that provides the **<api_version>/<account>** prefix that is specified in other requests throughout the API documentation.

Table 3.3. Response Headers

Name	Description	Type
X-Storage-Token	The authorization token for the **X-Auth-User** specified in the request.	String
X-Storage-Url	The URL and **<api_version>/<account>** path for the user.	String

Example Response

```
HTTP/1.1 204 No Content
Date: Mon, 16 Jul 2012 11:05:33 GMT
Server: swift
X-Storage-Url: https://swift.radosgwhost.com/v1/ACCT-12345
X-Storage-Token: UOlCCC8TahFKlWuv9DB09TWHF0nDjpPElha0kAa
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

3.2. SERVICE OPERATIONS

To retrieve data about our Swift-compatible service, you can execute **GET** requests using the **X-Storage-Url** value retrieved during authentication.

3.2.1. List Containers

A **GET** request that specifies the API version and the account will return a list of containers for a particular user account. Since the request returns a particular user's containers, the request requires an authentication token. The request cannot be made anonymously.

Syntax

```
GET /<api_version>/<account> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
```

Table 3.4. Request Parameters

Name	Description	Type	Required	Valid Values
limit	Limits the number of results to the specified value.	Integer	No	N/A
format	Defines the format of the result.	String	No	**json** or **xml**
marker	Returns a list of results greater than the marker value.	String	No	N/A

The response contains a list of containers, or returns with an HTTP 204 response code

Table 3.5. Response Entities

Name	Description	Type
account	A list for account information.	Container
container	The list of containers.	Container
name	The name of a container.	String
bytes	The size of the container.	Integer

Return to the features table.

3.3. CONTAINER OPERATIONS

A container is a mechanism for storing data objects. An account can have many containers, but container names must be unique. This API enables a client to create a container, set access controls and metadata, retrieve a container's contents, and delete a container. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless a container's access control is deliberately made publicly accessible, that is, allows anonymous requests.

NOTE

The Amazon S3 API uses the term 'bucket' to describe a data container. When you hear someone refer to a 'bucket' within the Swift API, the term 'bucket' might be construed as the equivalent of the term 'container.'

One facet of object storage is that it does not support hierarchical paths or directories. Instead, it supports one level consisting of one or more containers, where each container might have objects. The RADOS Gateway's Swift-compatible API supports the notion of 'pseudo-hierarchical containers', which is a means of using object naming to emulate a container, or directory hierarchy without actually implementing one in the storage system. You can name objects with pseudo-hierarchical names, for example, photos/buildings/empire-state.jpg, but container names cannot contain a forward slash (/) character.

IMPORTANT

When uploading large objects to versioned Swift containers, use the **--leave-segments** option with the **python-swiftclient** utility. Not using **--leave-segments** overwrites the manifest file. Consequently, an existing object is overwritten, which leads to data loss.

3.3.1. Container Operations with Multi Tenancy

When a client application accesses containers, it always operates with credentials of a particular user. In Red Hat Ceph Storage 3, every user belongs to a tenant. See Multi Tenancy for additional details. Consequently, every container operation has an implicit tenant in its context if no tenant is specified

explicitly. Thus multi tenancy is completely backward compatible with previous releases, as long as the referred containers and referring user belong to the same tenant.

Extensions employed to specify an explicit tenant differ according to the protocol and authentication system used.

A colon character separates tenant and container, thus a sample URL would be:

Example

```
https://rgw.domain.com/tenant:container
```

By contrast, in a **create_container()** method, simply separate the tenant and container in the container method itself:

Example

```
create_container("tenant:container")
```

3.3.2. Create a Container

To create a new container, make a **PUT** request with the API version, account, and the name of the new container. The container name must be unique, must not contain a forward-slash (/) character, and should be less than 256 bytes. You can include access control headers and metadata headers in the request. You can also include a storage policy identifying a key for a set of placement pools, for example, execute **radosgw-admin zone get** to see a list of available keys under **placement_pools**. A storage policy enables you to specify a special set of pools for the container, for example, SSD-based storage. The operation is idempotent; that is, if you make a request to create a container that already exists, it will return with a HTTP 202 return code, but will not create another container.

Syntax

```
PUT /<api_version>/<account>/<tenant>:<container> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
X-Container-Read: <comma_separated_uids>
X-Container-Write: <comma_separated_uids>
X-Container-Meta-<key>: <value>
X-Storage-Policy: <placement_pools_key>
```

Table 3.6. Headers

Name	Description	Type	Required
X-Container-Read	The user IDs with read permissions for the container.	Comma-separated string values of user IDs.	No

Name	Description	Type	Required
X-Container-Write	The user IDs with write permissions for the container.	Comma-separated string values of user IDs.	No
X-Container-Meta-{key}	A user-defined meta data key that takes an arbitrary string value.	String	No
X-Storage-Policy	The key that identifies the storage policy under **placement_pools** for the Ceph Object Gateway. Execute **radosgw-admin zone get** for available keys.	String	No

If a container with the same name already exists, and the user is the container owner then the operation will succeed. Otherwise the operation will fail.

Table 3.7. HTTP Response

Name	Description	Status Code
409	The container already exists under a different user's ownership.	**BucketAlreadyExists**

Return to the features table.

3.3.3. List a Container's Objects

To list the objects within a container, make a **GET** request with the with the API version, account, and the name of the container. You can specify query parameters to filter the full list, or leave out the parameters to return a list of the first 10,000 object names stored in the container.

Syntax

```
GET /<api_version>/<tenant>:<container> HTTP/1.1
 Host: <Fully_Qualified_Domain_Name>
 X-Auth-Token: <auth_token>
```

Table 3.8. Parameters

Name	Description	Type	Valid Values	Required
format	Defines the format of the result.	String	**json** or **xml**	No
prefix	Limits the result set to objects beginning with the specified prefix.	String	N/A	No
marker	Returns a list of results greater than the marker value.	String	N/A	No
limit	Limits the number of results to the specified value.	Integer	0 - 10,000	No
delimiter	The delimiter between the prefix and the rest of the object name.	String	N/A	No
path	The pseudo-hierarchical path of the objects.	String	N/A	No

Table 3.9. Response Entities

Name	Description	Type
container	The container.	Container
object	An object within the container.	Container
name	The name of an object within the container.	String
hash	A hash code of the object's contents.	String
last_modified	The last time the object's contents were modified.	Date
content_type	The type of content within the object.	String

Return to the features table.

3.3.4. Update a Container's Access Control Lists (ACLs)

When a user creates a container, the user has read and write access to the container by default. To allow other users to read a container's contents or write to a container, you must specifically enable the user. You can also specify * in the **X-Container-Read** or **X-Container-Write** settings, which effectively enables all users to either read from or write to the container. Setting * makes the container public. That is it enables anonymous users to either read from or write to the container.

Syntax

```
POST /<api_version>/<account>/<tenant>:<container> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
 X-Auth-Token: <auth_token>
 X-Container-Read: *
 X-Container-Write: <uid1>, <uid2>, <uid3>
```

Table 3.10. Request Headers

Name	Description	Type	Required
X-Container-Read	The user IDs with read permissions for the container.	Comma-separated string values of user IDs.	No
X-Container-Write	The user IDs with write permissions for the container.	Comma-separated string values of user IDs.	No

Return to the features table.

3.3.5. Add/Update Container Metadata

To add metadata to a container, make a **POST** request with the API version, account, and container name. You must have write permissions on the container to add or update metadata.

Syntax

```
POST /<api_version>/<account>/<tenant>:<container> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
 X-Auth-Token: <auth_token>
 X-Container-Meta-Color: red
 X-Container-Meta-Taste: salty
```

Table 3.11. Request Headers

Name	Description	Type	Required
X-Container-Meta-<key>	A user-defined meta data key that takes an arbitrary string value.	String	No

Return to the features table.

3.3.6. Delete a Container

To delete a container, make a **DELETE** request with the API version, account, and the name of the container. The container must be empty. If you'd like to check if the container is empty, execute a **HEAD** request against the container. Once you've successfully removed the container, you'll be able to reuse the container name.

Syntax

```
DELETE /<api version>/<account>/<tenant>:<container> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth-token>
```

Table 3.12. HTTP Response

Name	Description	Status Code
204	The container was removed.	**NoContent**

Return to the features table.

3.4. OBJECT OPERATIONS

An object is a container for storing data and metadata. A container might have many objects, but the object names must be unique. This API enables a client to create an object, set access controls and metadata, retrieve an object's data and metadata, and delete an object. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless the container or object's access control is deliberately made publicly accessible, that is, allows anonymous requests.

3.4.1. Create/Update an Object

To create a new object, make a **PUT** request with the API version, account, container name and the name of the new object. You must have write permission on the container to create or update an object. The object name must be unique within the container. The **PUT** request is not idempotent, so if you do not use a unique name, the request will update the object. However, you can use pseudo-hierarchical syntax in the object name to distinguish it from another object of the same name if it is under a different pseudo-hierarchical directory. You can include access control headers and metadata headers in the request.

Syntax

```
PUT /<api_version>/<account>/<tenant>:<container>/<object> HTTP/1.1
 Host: <Fully_Qualified_Domain_Name>
 X-Auth-Token: <auth_token>
```

Table 3.13. Request Headers

Name	Description	Type	Required	Valid Values
ETag	An MD5 hash of the object's contents. Recommended.	String	No	N/A
Content-Type	The type of content the object contains.	String	No	N/A
Transfer-Encoding	Indicates whether the object is part of a larger aggregate object.	String	No	**chunked**

Return to the features table.

3.4.2. Copy an Object

Copying an object allows you to make a server-side copy of an object, so that you don't have to download it and upload it under another container/name. To copy the contents of one object to another object, you can make either a **PUT** request or a **COPY** request with the API version, account, and the container name. For a **PUT** request, use the destination container and object name in the request, and the source container and object in the request header. For a **Copy** request, use the source container and object in the request, and the destination container and object in the request header. You must have write permission on the container to copy an object. The destination object name must be unique within the container. The request is not idempotent, so if you do not use a unique name, the request will update the destination object. However, you can use pseudo-hierarchical syntax in the object name to distinguish the destination object from the source object of the same name if it is under a different pseudo-hierarchical directory. You can include access control headers and metadata headers in the request.

Syntax

```
PUT /<api_version>/<account>/<tenant>:<dest_container>/<dest_object> HTTP/1.1
X-Copy-From: <tenant>:<source_container>/<source_object>
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
```

or alternatively:

Syntax

```
COPY /<api_version>/<account>/<tenant>:<source_container>/<source_object> HTTP/1.1
Destination: <tenant>:<dest_container>/<dest_object>
```

Table 3.14. Request Headers

Name	Description	Type	Required
X-Copy-From	Used with a **PUT** request to define the source container/object path.	String	Yes, if using **PUT**
Destination	Used with a **COPY** request to define the destination container/object path.	String	Yes, if using **COPY**
If-Modified-Since	Only copies if modified since the date/time of the source object's **last_modified** attribute.	Date	No
If-Unmodified-Since	Only copies if not modified since the date/time of the source object's **last_modified** attribute.	Date	No
Copy-If-Match	Copies only if the ETag in the request matches the source object's ETag.	ETag.	No

Name	Description	Type	Required
Copy-If-None-Match	Copies only if the ETag in the request does not match the source object's ETag.	ETag.	No

Return to the features table.

3.4.3. Delete an Object

To delete an object, make a **DELETE** request with the API version, account, container and object name. You must have write permissions on the container to delete an object within it. Once you've successfully deleted the object, you'll be able to reuse the object name.

Syntax

```
DELETE /<api_version>/<account>/<tenant>:<container>/<object> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
```

Return to the features table.

3.4.4. Get an Object

To retrieve an object, make a **GET** request with the API version, account, container and object name. You must have read permissions on the container to retrieve an object within it.

Syntax

```
GET /<api version>/<account>/<tenant>:<container>/<object> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth-token>
```

Table 3.15. Request Headers

Name	Description	Type	Required
range	To retrieve a subset of an object's contents, you can specify a byte range.	Date	No
If-Modified-Since	Only copies if modified since the date/time of the source object's **last_modified** attribute.	Date	No
If-Unmodified-Since	Only copies if not modified since the date/time of the source object's **last_modified** attribute.	Date	No
Copy-If-Match	Copies only if the ETag in the request matches the source object's ETag.	ETag.	No

Name	Description	Type	Required
Copy-If-None-Match	Copies only if the ETag in the request does not match the source object's ETag.	ETag.	No

Table 3.16. Response Headers

Name	Description
Content-Range	The range of the subset of object contents. Returned only if the range header field was specified in the request.

Return to the features table.

3.4.5. Get Object Metadata

To retrieve an object's metadata, make a **HEAD** request with the API version, account, container and object name. You must have read permissions on the container to retrieve metadata from an object within the container. This request returns the same header information as the request for the object itself, but it does not return the object's data.

Syntax

```
HEAD /<api_version>/<account>/<tenant>:<container>/<object> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
```

Return to the features table.

3.4.6. Add/Update Object Metadata

To add metadata to an object, make a **POST** request with the API version, account, container and object name. You must have write permissions on the parent container to add or update metadata.

Syntax

```
POST /<api_version>/<account>/<tenant>:<container>/<object> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
```

Table 3.17. Request Headers

Name	Description	Type	Required
X-Object-Meta-<key>	A user-defined meta data key that takes an arbitrary string value.	String	No

Return to the features table.

3.5. TEMP URL OPERATIONS

To allow temporary access, for example GET requests, to objects without the need to share credentials, temp url functionality is supported by swift endpoint of **radosgw**. For this functionality, initially the value of **X-Account-Meta-Temp-URL-Key** and optionally **X-Account-Meta-Temp-URL-Key-2** should be set. The Temp URL functionality relies on a HMAC-SHA1 signature against these secret keys.

3.5.1. POST Temp-URL Keys

A **POST** request to the swift account with the required Key will set the secret temp url key for the account against which temporary url access can be provided to accounts. Up to two keys are supported, and signatures are checked against both the keys, if present, so that keys can be rotated without invalidating the temporary urls.

Syntax

```
POST /<api_version>/<account> HTTP/1.1
Host: <Fully_Qualified_Domain_Name>
X-Auth-Token: <auth_token>
```

Table 3.18. Request Headers

Name	Description	Type	Required
X-Account-Meta-Temp-URL-Key	A user-defined key that takes an arbitrary string value.	String	Yes
X-Account-Meta-Temp-URL-Key-2	A user-defined key that takes an arbitrary string value.	String	No

3.5.2. GET Temp-URL Objects

Temporary URL uses a cryptographic HMAC-SHA1 signature, which includes the following elements:

- The value of the Request method, "GET" for instance
- The expiry time, in format of seconds since the epoch, that is, Unix time
- The request path starting from "v1" onwards

The above items are normalized with newlines appended between them, and a HMAC is generated using the SHA-1 hashing algorithm against one of the Temp URL Keys posted earlier.

A sample python script to demonstrate the above is given below:

Example

```
import hmac
from hashlib import sha1
from time import time
```

```
method = 'GET'
host = 'https://objectstore.example.com'
duration_in_seconds = 300  # Duration for which the url is valid
expires = int(time() + duration_in_seconds)
path = '/v1/your-bucket/your-object'
key = 'secret'
hmac_body = '%s\n%s\n%s' % (method, expires, path)
hmac_body = hmac.new(key, hmac_body, sha1).hexdigest()
sig = hmac.new(key, hmac_body, sha1).hexdigest()
rest_uri = "{host}{path}?temp_url_sig={sig}&temp_url_expires={expires}".format(
    host=host, path=path, sig=sig, expires=expires)
print rest_uri
```

Example Output

```
https://objectstore.example.com/v1/your-bucket/your-object?
temp_url_sig=ff4657876227fc6025f04fcf1e82818266d022c6&temp_url_expires=1423200992
```

3.6. SWIFT API LIMITATIONS

IMPORTANT

The following limitations should be used with caution. There are implications related to your hardware selections, so you should always discuss these requirements with your Red Hat account team.

- **Maximum object size when using Swift API:** 5GB
- **Maximum metadata size when using Swift API:** There is no defined limit on the total size of user metadata that can be applied to an object, but a single HTTP request is limited to 16,000.