PUBLIC

2017-10-26

Object Store Service

THE BEST RUN

Content

1 Object Store Service

Object Store service enables storage and management of objects, which involves creation, upload, download, and deletion of objects. This service is specific to the IaaS layer i.e,

Features

- Easy and secure access - Creates S3 buckets and passes secure credentials to the application to access the buckets.
- High availability - Ensures that the storage for solutions is always available, without any interruptions.
- High durability - Ensures durability as the underlying technologies that we use provide storage replication.
- Scalability - Offers highly scalable storage that can be used by cloud foundry applications to store and manage objects.

Cloud Foundry Service Plans

Plan	Description	Feature
s3-standard	Provides an objectstore on AWS S3 that CF applications can use to store & manage objects	AWS-S3 Standard Storage class (Available only in AWS based Landscapes).
swift-standard	Provides scalable cloud blob storage on Openstack Swift that CF applications can use to store & manage objects	Swift service (Available only in SAP based Landscapes).
gcs-standard	Provides an objectstore on Google Cloud Storage that CF applications can use to store & manage objects	GCS regional storage class (Available only in GCP based Landscapes).
azure-standard	Provides an objectstore on Azure BlobStorage account that CF applications can use to store & manage objects	Azure LRS Storage class (Available only in Azure based Landscapes).

Related Information

Configure Object Store to use Azure Blob Storage [page 4]
Configure Object Store to use Amazon Simple Storage Service [page 10]
Configure Object Store to use SAP Infrastructure [page 18]
Configure Object Store to use GCP [page 24]

2 Object Store Service on Azure Blob Storage

2.1 Configure Object Store to use Azure Blob Storage

The Object Store service in the Azure landscape provides Azure BlobStorage containers as object store resource.

Context

The Object Store service supports following operations: Create Azure BlobStorage container (Service instance), delete container, create credentials to access the container via cf bind-service and create-service-key call, and delete credentials via cf unbind-service and delete-service-key call.

i Note

By Azure's subscription structure, a container has to be created within a storage account. In Object Store service, an Azure BlobStorage account is created for each CF space, and for each service instance in that CF space creates a container within that BlobStorage account. Usage billing takes place at the Cloud Foundry (CF) space level (BlobStorage account level) rather than at the service instance (or container) level. To perform object-related operations (upload, download, delete, and so on), you can directly make calls to Azure using the Azure SDKs.

Procedure

1. Look for the service on Cloud Foundry using the command `cf marketplace`.

   ```
   Service: objectstore
   Plans: azure-standard
   Description: Objectstore service
   ```

   You see a service called "objectstore" is available, with plan "azure-standard."
2. Create a service instance using the command `cf create-service objectstore azure-standard <serviceInstanceName>`. This creates a service instance and a container in Azure blob storage. A container is created in the same region as the underlying Cloud Foundry infrastructure.
3. List the services available using the command `cf services`.

   ```
   Name || Service || Plan || Bound apps || Last operation
   <serviceInstanceName> || objectstore || azure-standard
   ```

4. Get credentials.

   For container and blob operations, the application requires certain parameters (container name, container URI, SAS Token). There are two ways a user can get these credentials; either by binding the service instance to an application directly, or by using the Cloud Foundry Service-Key concept.
5. Bind an Object Store service instance to an application. `cf bind-service <yourApplicatioName> <serviceInstanceName>`
6. The application now appears as bound with the service. Use the command `cf services` to obtain the following output.

   Output:

   ```
   Name || Service || Plan || Bound apps || Last operation
   <serviceInstanceName> || objectstore || azure-standard || <ApplicatioName>
   ```
7. Read environment variables using the command `cf env <ApplicatioName>`

   ```
   System-Provided:
   {
    "VCAP_SERVICES": {
     "objectstore": [
      {
       "credentials": {
       "account_name": "<storage_account_name>",
        "container_name": "<container_name_corresponding_to_service_instance>",
        "container_uri": "<container_uri>",
        "sas_token": "<sas_token>",
       "region": "<azure_container_region>"
       },
       "label": "objectstore",
       "name": "<serviceInstanceName>",
       "plan": "azure-standard",
       "provider": null,
       "syslog_drain_url": null,
       "tags": [
        "blobStore",
        "objectStore"
       ],
       "volume_mounts": []
      }
     ]
    }
   }
   ```

   When you bind your application to a service instance, Cloud Foundry sets the environment variable: VCAP_SERVICES. From VCAP_SERVICES, your application should read the following:
   - account_name
   - container_name
   - container_uri
   - sas_token
   - region

2.2 Create Service-Keys for an ObjectStore Service Instance

Procedure

1. Create a service-key using the command: `cf create-service-key <serviceInstanceName> <KeyName>`
2. View the service-key that was created using the command `cf service-key <serviceInstanceName> <Key Name>.` The output would be like:

```
Getting key <KeyName> for service instance <serviceInstanceName> as admin...
{
   "account_name": "<storage_account_name>",
   "container_name": "<container_name_corresponding_to_service_instance>",
   "container_uri": "<container_uri>",
   "sas_token": "<sas_token>",
   "region": "<azure_container_region>"
 }
```

   A json file with following fields appear in the output:
   - account_name
   - container_name
   - container_uri
   - sas_token
   - region

Reference Script to Automate the Process Mentioned Above

Context

The script snippet in the steps below may assist you to automate the process of passing credentials as environment variables to the application. We have a limitation on the number of service instance bindings that can be created per service instance in Cloud Foundry to 5. If user has a use case where one needs more than 5 bindings. In those cases, we recommend users to use `cf service-keys` concept from Cloud Foundry.

Suppose multiple applications have to use a particular service instance and if the `cf bind-service` command is used, then one can exhaust the Binding limits. To overcome that, please make use of `cf service-key` concept.

Also, if one follows blue-green deployment of application to overcome downtime, there would be a time when 2 bindings for a service instance would be needed at a time - leading to exhaust Binding limits if other applications are also bound. In this case also, below steps can be used in the script to overcome binding limitations.

Procedure

1. Create a service key using the command `cf create-service-key< serviceInstanceName> <KeyName>`.
2. Retrieve credentials from the service key and store it in a variable using the command `credentialJSON=`cf service-key <servicenInstanceName> <KeyName> | sed -n '1,2!p'``.
3. Store the credentials into separate variables using the following commands:

```
accountName=`echo "${credentialsJSON}" | jq -r '.account_name'`
containerName=`echo "${credentialsJSON}" | jq -r '.container_name'`
containerUri=`echo "${credentialsJSON}" | jq -r '.container_uri'`
sasToken=`echo "${credentialsJSON}" | jq -r '.sas_token'
region=`echo "${credentialsJSON}" | jq -r '.region'`
```

4. Pass the credentials stored in above variables to your app.

   For CF application, users can pass the above credentials to an application by setting environment variables. To pass the variables as environment variables using application's manifest.yml file, make following entries in your manifest.yml file:

```
env:
    vcap.services.<serviceInstanceName>.credentials.account_name:
    vcap.services.<serviceInstanceName>.credentials.container_name:
    vcap.services.<serviceInstanceName>.credentials.container_uri:
    vcap.services.<serviceInstanceName>.credentials.sas_token:
    vcap.services.<serviceInstanceName>.credentials.region:
```

   And in the script, one can update these environment variables in manifest.yml using following commands, to be done before deployment:

```
sed -i "s/vcap.services.<serviceInstanceName>.credentials.account_name.*/
vcap.services.<serviceInstanceName>.credentials.account_name: $accountName /
g" ./manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.container_name.*/
vcap.services.<serviceInstanceName>.credentials.container_name:
$containerName /g" ./manifest.yml
sed -i
"s~vcap.services.<serviceInstanceName>.credentials.container_uri.*~vcap.servic
es.<serviceInstanceName>.credentials.container_uri: $containerUri ~g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.sas_token.*/
vcap.services.<serviceInstanceName>.credentials.sas_token: $sasToken /g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.region.*/
vcap.services.<serviceInstanceName>.credentials.region: $region /g" ./
manifest.yml
```

2.3 Supported Operations

Operations supported on containers/blobs

Operations

- List (List blobs in the container)
- Delete (Delete blob in the container)
- Write (Create or write content, properties, metadata, or block list)
- Create (Write a new blob, snapshot a blob, or copy a blob to a new blob)
- Add (Add a block to an append blob)
- Read (Read the content, properties, metadata and block list)

! Restriction

- Number of service instance bindings that can be created per service instance in Cloud Foundry: 5
- At each CF Space level, there is a storage size limit of 500TB, this limit is set by Azure. For more information, see Storage Limits .

References

https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

https://docs.microsoft.com/en-us/azure/storage/blobs/storage-dotnet-how-to-use-blobs

2.4 Java Code Snippets

You can use the following sample code snippets as a reference:

- Read VCAP_SERVICES

  Sample Code

  ```
  @Value("${vcap.services.objectstore.credentials.account_name}")
  String accountName;
  @Value("${vcap.services.objectstore.credentials.container_name}")
  String containerName;
  @Value("${vcap.services.objectstore.credentials.container_uri}")
  String containerUri;
  @Value("${vcap.services.objectstore.credentials.sas_token}")
  String sasToken;
  @Value("${vcap.services.objectstore.credentials.region}")
  String region;
  ```

- Get container reference

Sample Code

```
CloudBlobContainer cloudBlobContainer = new CloudBlobContainer(new
URI(containerUri + "?" + sasToken));
```

- Get BlockBlob reference

Sample Code

```
String blobName = "test-blob-1";
CloudBlockBlob cloudBlockBlob = new CloudBlockBlob(new URI(containerUri +
"/" + blobName + "?" + sasToken));
```

- List blobs in the container

Sample Code

```
for (ListBlobItem listBlobItem : cloudBlobContainer.listBlobs()) {
    System.out.println("Storage URI: " + listBlobItem.getStorageUri());
}
```

- Upload blob

Sample Code

```
String blobContent = "This is blob content text.";
InputStream inputStream = new
ByteArrayInputStream(blobContent.getBytes("UTF-8"));
cloudBlockBlob.upload(inputStream, blobContent.length());
```

- Download blob

Sample Code

```
OutputStream outputStream = new ByteArrayOutputStream();
cloudBlockBlob.download(outputStream);
```

- Delete Blob

Sample Code

```
cloudBlockBlob.deleteIfExists();
```

Instead of providing blob content as text, you can provide blobs by providing file path. For more information, see Blob operations .

3 Object Store Service on Amazon Web Service

3.1 Configure Object Store to use Amazon Simple Storage Service

Configuring enables you to bind your application with an Object Store instance and store objects.

Context

Provision using:

- Object Store
  This involves restaging after binding the service to an application.

> **Note**
>
> - Object store service supports STANDARD storage class objects.
>
> - Objectstore service is a paid service, and the plans are sold as units of 100GBs. For more information, refer SAP Cloud Platform Marketplace. When user creates a service instance, an AWS S3 bucket is created, and user can store all the units in a single S3 bucket or in multiple S3 buckets.
>
> - Creating a new service instance for every purchased unit of Object store service is not advised.

Procedure

1. Open the command line interface.
2. List the Object Store service from the marketplace using the command `cf marketplace`. A service called "objectstore" with the plan "s3-standard" is available on Cloud Foundry environment.

Service	Plan	Description
`<objectstore>`	`<s3-standard>`	ObjectStore service with storage space on Amazon s3.

3. Create an instance of the service using the command `cf create-service objectstore s3-standard <serviceInstanceName>`.

i Note

You can create only 5 service instances per org.

4. List the service instance using the command `cf services`. The output lists the service instance details.

Name	Service	Plan	Bound Apps	Last Operation
`<serviceInstance Name>`	`<objectstore>`	`<s3-standard>`	none	none

5. Bind an application to the object store service instance using the command `cf bind-service <yourApplicationName> <serviceInstanceName>`.

   i Note

   You can bind only 5 applications per service instance. For more information, see Use Service Keys to Access Object Store Service Instance [page 12]

   Binding enables Cloud Foundry environment to set the environment variable “VCAP_SERVICES”. The bound application uses the environment variable to fetch the credentials for the service instance.
6. Restage your application using the command `cf restage <yourApplicationName>`.
   Restaging enables the application to use the environment variables at runtime.
7. Verify the binding using the command `cf services`. The output lists the details including the name of the application bounded to the objectstore service instance `<serviceInstanceName>`.

Name	Service	Plan	Bound Apps	Last Operation
`<serviceInstance Name>`	`<objectstore>`	`<s3-standard>`	`<yourApplication Name>`	none

8. View the environment variable used by the application using the command `cf env <yourApplicationName>`.

   i Note

   Application reads the following details from the “VCAP_SERVICES”:
   - bucket
   - access_key_id
   - secret_access_key
   - host
   - region
   - username

   You can access Amazon Simple Storage Service (Amazon S3) using the credentials mentioned in the environment variable.

   Sample Code

   ```
   System-Provided:
   ```

```
{
 "VCAP_SERVICES": {
  "objectstore": [
   {
    "credentials": {
     "access_key_id": "<some_access_key_id>",
     "bucket": "<some_bucket_name>",
     "host": "<region_specific_s3_endpoint>",
    "region": "<region>",
     "secret_access_key": "<some_secret_access_key>",
     "uri": "s3://
<some_access_key_id>:<some_secret_access_key>@<region_specific_s3_endpoint>
/<some_bucket_name>",
     "username": "<some_username>"
    },
    "label": "objectstore",
    "name": "<serviceInstanceName>",
    "plan": "s3-standard",
    "provider": null,
    "syslog_drain_url": null,
    "tags": [
     "blobStore",
     "objectStore"
    ],
    "volume_mounts": []
   }
  ]
 }
```

3.2 Use Service Keys to Access Object Store Service Instance

Applications use service keys to access Object Store service instance.

Context

Follow the steps below to create a service key:

Procedure

1. Create a service key using the command `cf create-service-key< serviceInstanceName> <KeyName>`.
2. View the generated service key using the command `cf service-key <serviceInstanceName> <KeyName>`.

   The sample output is as shown below:

   ```
   Getting key <KeyName> for service instance <serviceInstanceName> as admin...
   ```

```
{
 "access_key_id": "<some_access_key_id>",
 "bucket": "<some_bucket_name>",
 "host": "<s3_endpoint_depending_on_region>",
 "region": "<region_of_bucket>",
 "secret_access_key": "<some_secret_access_key>",
 "uri": "s3://
<some_access_key_id>:<some_secret_access_key>@"<s3_endpoint_depending_on_regio
n>/<some_bucket_Name>",
 "username": "<some_username>"
}
```

A json file with the following fields appears in the output:

- bucket
- access_key_id
- secret_access_key
- host
- region
- username

Reference Script to Automate the Above Procedure

Context

The script snippet in the steps below may assist you to automate the process of passing credentials as environment variables to the application.

Procedure

1. Create a service key using the command `cf create-service-key< serviceInstanceName> <KeyName>`.
2. Retrieve credentials from the service key and store it in a variable using the command `credentialJSON=`cf service-key <servicenInstanceName> <KeyName> | sed -n '1,2!p'``.
3. Store the credentials into separate variables using the following commands:
   - `accessKey=`echo "${credentialsJSON}" | jq -r '.access_key_id'``
   - `secretKey=`echo "${credentialsJSON}" | jq -r '.secret_access_key'``
   - `host=`echo "${credentialsJSON}" | jq -r '.host'``
   - `region=`echo "${credentialsJSON}" | jq -r '.region'``
   - `bucket=`echo "${credentialsJSON}" | jq -r '.bucket'``
4. Set the environment variable by creating a section named **env** in the “manifest.yml” file.

```
env:
    vcap.services.<serviceInstanceName>.credentials.access_key_id:
    vcap.services.<serviceInstanceName>.credentials.secret_access_key:
    vcap.services.<serviceInstanceName>.credentials.host:
    vcap.services.<serviceInstanceName>.credentials.region:
```

```
    vcap.services.<serviceInstanceName>.credentials.bucket:
```

> **i Note**
>
> For Cloud Foundry application, you can pass the credentials to an application by setting environment variables in the “manifest.yml” file.

5. Update the environment variables in the “manifest.yml” using the following commands :

```
sed -i "s/vcap.services.<serviceInstanceName>.credentials.access_key_id.*/
vcap.services.<serviceInstanceName>.credentials.access_key_id: $accessKey /
g" ./manifest.yml
sed -i
"s~vcap.services.<serviceInstanceName>.credentials.secret_access_key.*~vcap.se
rvices.<serviceInstanceName>.credentials.secret_access_key: $secretKey ~g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.host.*/
vcap.services.<serviceInstanceName>.credentials.host: $host /g" ./manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.region.*/
vcap.services.<serviceInstanceName>.credentials.region: $region /g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.bucket.*/
vcap.services.<serviceInstanceName>.credentials.bucket: $bucket /g" ./
manifest.yml
```

6. Deploy the application to Cloud Foundry using the command `cf push -f manifest.yml`

3.3 Supported Operations

Here are the list of supported operations:

Operation supported on Buckets

List Bucket (List of objects in a bucket)

Operations supported on Objects

- Abort Multipart Upload
- Delete Object
- Delete Object Tagging
- Get Object
- Get Object ACL
- Get Object Torrent
- Get Object Tagging
- List Multipart Upload Parts
- Put Object
- Put Object ACL
- Put Object Version ACL

> **i Note**
>
> To perform object related operations (upload, download, delete etc) users can directly make calls to AWS using the AWS SDKs.

For more information, see the documentation on the Amazon Web Services Web site.

3.4 Java Code Snippets

You can use the following sample code snippets as a reference:

- Read VCAP_SERVICES

Sample Code

```
@Value("${vcap.services.objectstore.credentials.access_key_id}")
String accessKeyId;
@Value("${vcap.services.objectstore.credentials.secret_access_key}")
String secretAccessKey;
@Value("${vcap.services.objectstore.credentials.bucket}")
String bucketName;
@Value("${vcap.services.objectstore.credentials.host}")
String endPoint;
@Value("${vcap.services.objectstore.credentials.region}")
String bucketRegion;
```

- Get an Amazon S3 client

Sample Code

```
//Using setEndPoint method
AWSCredentials credentials = new BasicAWSCredentials(accessKeyId,
secretAccessKey);
AmazonS3 s3client = null;
s3client = new AmazonS3Client(credentials);
s3client.setEndPoint(endPoint);
s3client.setRegion(bucketRegion);
//Using region method
AWSCredentials credentials = new BasicAWSCredentials(accessKeyId,
secretAccessKey);
AmazonS3 s3client = null;
s3client = new AmazonS3Client(credentials);
s3client.setRegion(bucketRegion);
```

- Upload an object

Sample Code

```
s3client.putObject(new PutObjectRequest(bucketName, objectName, file));
```

- Download an object

Sample Code

```
s3client.getObject(new GetObjectRequest(bucketName, objectName));
```

- Delete an object

Sample Code

```
s3client.deleteObject(bucketName, objectName);
```

3.5 Sample Script Snippets to Overcome Service Binding Limitations

There is a limitation on the number of service instance bindings that can be created per service instance in Cloud Foundry to five. If you have a use case where you need more than five bindings, you should use `cf service-keys` concept from Cloud Foundry.

Suppose multiple applications have to use a particular service instance and if the cf bind-service command is used, then you can exhaust the Binding limits. To overcome that, make use of cf service-key concept.

Also, if you follow blue-green deployment of application to overcome downtime, there is a time when two bindings for a service instance is needed at a time - leading to exhaust binding limits if other applications are also bound. In this case also, below steps can be used in the script to overcome binding limitations. Here is how you can make use of service-key functionality in a programmatic way to overcome binding limitations:

1. Create a service key.
2. Retrieve credentials from the service key and store it in a variable using the command

   ```
   credentialJSON=`cf service-key <servicenInstanceName> <KeyName> | sed -n '1,2!
   p'`
   ```
3. Store the credentials obtained in above step in variables.

   ```
   accessKey=`echo "${credentialsJSON}" | jq -r '.access_key_id'`
   secretKey=`echo "${credentialsJSON}" | jq -r '.secret_access_key'`
   host=`echo "${credentialsJSON}" | jq -r '.host'`
   region=`echo "${credentialsJSON}" | jq -r '.region'`
   bucket=`echo "${credentialsJSON}" | jq -r '.bucket'`
   ```
4. Pass the credentials stored in above variables to your application.
   For CF application, you can pass the above credentials to an application by setting environment variables. To pass the variables as environment variables using application's manifest.yml file, make following entries in your manifest.yml file:

   ```
   env:
       vcap.services.<serviceInstanceName>.credentials.access_key_id:
       vcap.services.<serviceInstanceName>.credentials.secret_access_key:
       vcap.services.<serviceInstanceName>.credentials.host:
       vcap.services.<serviceInstanceName>.credentials.region:
       vcap.services.<serviceInstanceName>.credentials.bucket:
   ```

   And in the script, you can update these environment variables in manifest.yml using the following commands before deployment:

   ```
   sed -i "s/vcap.services.<serviceInstanceName>.credentials.access_key_id.*/
   vcap.services.<serviceInstanceName>.credentials.access_key_id: $accessKey /
   g" ./manifest.yml
   sed -i
   "s~vcap.services.<serviceInstanceName>.credentials.secret_access_key.*~vcap.se
   ```

```
rvices.<serviceInstanceName>.credentials.secret_access_key: $secretKey ~g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.host.*/
vcap.services.<serviceInstanceName>.credentials.host: $host /g" ./manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.region.*/
vcap.services.<serviceInstanceName>.credentials.region: $region /g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.bucket.*/
vcap.services.<serviceInstanceName>.credentials.bucket: $bucket /g" ./
manifest.yml
```

4 Object Store Service on SAP Infrastructure

4.1 Configure Object Store to use SAP Infrastructure

Object store service on SAP infrastructure provides swift containers as object store resource.

Prerequisites

Ensure that you have space developer privileges in an org and space.

Context

The Objectstore service supports the following operations:

- Create Swift container (service instance) - containers are managed in the object store service tenant.
- Delete swift container.
- Create credentials to access the container using the `cf bind-service` call.
- Delete credentials using the `cf unbind-service` call.

> i Note
>
> - The Objectstore service is a paid service; plans are sold as units of 100GBs.
> - When a user creates a service instance, a swift container is created and the user can store all the units in a single or multiple Swift containers.
> - We recommend that you do not create a new service instance for every purchased unit of Objectstore service.

Procedure

1. Look for available services on Cloud Foundry using the command `cf marketplace`.

   ```
   Service: objectstore
   Plans: swift-standard
   Description: Highly available, distributed, eventually consistent objectstore
   ```

2. Create a service instance of the above service and plan using the command `cf create-service objectstore swift-standard <serviceInstanceName>`.

3. List the services available using the command `cf services`.

```
Name || Service || Plan || Bound apps || Last operation
```

```
<serviceInstanceName> || objectstore || swift-standard
```

4. Get credentials.

   To use the Object Store-as-a-Service APIs for object upload, download, delete, and so on, the application requires certain parameters (container_name, domain, password, tenant, url and user name). The following steps fetch these parameters:
5. Bind an Object Store service instance to an application using the following commands: `cf bind-service <ApplicationName> <serviceInstanceName>` and `cf restage <ApplicationName>`
6. The application is shown as bound with the service. Use the command `cf services` to obtain the following output.

   Output:

```
Name || Service || Plan || Bound apps || Last operation
<serviceInstanceName> || objectstore || swift-standard || <ApplicatioName>
```

7. Read the environment variables using the command `cf env <ApplicatioName>`

```
{
    "VCAP_SERVICES": {
        "objectstore": [{
            "credentials": {
                "container_name": "<some_container_name>",
                "user_domain": "<some_user_domain_name>",
                "project_domain": "<some_project_domain_name>",
                "password": "<some_password>",
                "project": "<some_project_name>",
                "auth_url": "some_URL",
                "username": "some_username"
            },
            "label": "objectstore",
            "name": "<serviceInstanceName>",
            "plan": "swift-standard",
            "provider": null,
            "syslog_drain_url": null,
            "tags": ["blobStore", "objectStore"],
            "volume_mounts": []
        }]
    }
}
```

   When binding your application with a service instance, cloud foundry sets the environment variableVCAP_SERVICES. From VCAP_SERVICES, your application should read the following:
   - container_name: To get the unique name of container.
   - user_domain: User domain to be used in the request body for authentication.
   - project_domain: Project domain to be used in the request body for authentication.
   - password: Password to be used in the request body for authentication.
   - project: Project name to beused in the request body for authentication.
   - auth_url: Base URL for authentication API.
   - username: User name to be used in the request body for authentication.

4.2 Use Object Store Resource

This section helps you to use objectstore resource (Swift container). Before any operations on the container, user will have to generation the Authentication Token and then can make object related calls, details explained below

Procedure

1. Get authentication token to make object-related calls.

   In exchange for a set of authentication credentials, the identity service generate tokens. A token represents the authenticated identity of a user, and grants authorization on a specific project or domain. Currently, password-based authentication with scoped authorization is supported.
2. API specifications.

   Request Method: POST

   URL: {auth_url} + /v3/auth/tokens

   Description: The request body must include a payload that specifies the password authentication method, the credentials, and the project or domain authorization scope. The payload values are from the environment variables obtained in Configure Object Store to use SAP Infrastructure [page 18].

   Sample Code

   ```
   {
       "auth": {
           "identity": {
               "password": {
                   "user": {
                       "id": "<username >",
                       "password": "<password >",
                       "domain": {
                           "name": "<user domain >"
                       }
                   }
               },
               "methods": ["password"]
           },
           "scope": {
               "project": {
                   "name": "<project name>",
                   "domain": {
                       "name": "<project domain >"
                   }
               }
           }
       }
   }
   ```

   The response of the above request includes the following:
   - X-Subject-Token in Response Header: The value of X-Subject-Token must be set in X-Auth-Token header while making object/container API calls.
   - Endpoint URL in Response body: This URL is used as the base URL to make an object or container-related call, and may also be referred to as the <endpoint_url>.

4.2.1 Supported Endpoints for Making Object or Container Related API Calls

Object Endpoints

- **Upload object**
  Method: PUT
  URI: `<endpoint_url> + /{container}/{object}`
  Description: Creates an object with data content and metadata, or replaces an existing object with data content and metadata.
- **Download object**
  Method: GET
  URI: `<endpoint_url> + /{container}/{object}`
  Description: Downloads the object content and gets the object metadata.
- **Delete object**
  Method: DELETE
  URI: `<endpoint_url> + /{container}/{object}`
  Description: Permanently deletes an object from the object store.
- **Show object metadata**
  Method: HEAD
  URI: `<endpoint_url> + /{container}/{object}`
  Description: Shows object metadata.
- **Create or update object metadata**
  Method: POST
  URI: `<endpoint_url> + /{container}/{object}`
  Description: Creates or updates object metadata.

Container Endpoints

- **Show container details**
  Method: GET
  URI: `<endpoint_url> + /{container}`
  Description: Shows details of a container and lists objects, sorted by name in the container.
- **Show container metedata**
  Method: HEAD
  URI: `<endpoint_url> + /{container}`
  Description: Shows container metadata, including the number of objects and the total bytes of all objects stored in the container.

i Note

For the detailed documentation of the above mentioned APIs, refer the open stack documentation at `http://developer.opens tact.org/api-ref-objectstorage-v1.htm`. We support most of the request header and query parameters for each of the APIs listed above. However, there are a few that we do not support.

4.2.2 Implementation Using Open Source Libraries

There are a few open source libraries that can be used by user applications for making API calls. For example OpenStack4j, JClouds, and so on.

The following sample code snippets are for each of the APIs using the Openstack4j library:

- Get authentication token:

```
OSClientV3 osClient = OSFactory.builderV3().endpoint(<base_url> + "/v3").
credentials(username, password,
Identifier.byName(domain)).scopeToProject(Identifier.byName(tenant),
Identifier.byName(domain)).authenticate();
```

- Object upload:

```
osClient.objectStorage().objects().put(containerName, objectName,
Payloads.create(fileToBeUploaded));
```

- Object download:

```
SwiftObject swiftObject = os.objectStorage().objects().get(containerName,
objectName);
```

- Object delete:

```
ActionResponse actionResponse =
os.objectStorage().objects().delete(containerName, objectName);
```

- Object metadata:

```
Map<String, String> map = os.objectStorage().objects().get(containerName,
objectName).getMetadata();
```

- Object list in container:

```
final List<? extends SwiftObject> list =
osClient.objectStorage().objects().list(containerName);
```

- Container metadata:

```
Map<String, String> map =
os.objectStorage().containers().getMetadata(containerName);
```

- Create or update object metadata:

```
boolean updated = os.objectStorage().objects().updateMetadata(objectLocation,
metadata);
```

4.3 Delete Container

Follow the steps below to delete a container.

Procedure

1. Unbind the application from the service.

   Before deleting the container, you must delete all the credentials that are relevant to it, using the following command:

   ```
   cf unbind-service <applicationName> <serviceInstanceName>
   ```

   Command: `cf services`

   Output:

   ```
   Name || Service || Plan || Bound apps || Last operation
   <serviceInstanceName> || objectstore || swift-standard
   ```

   The application name does not appear in the bound apps section.

   Command: `cf env appname`

   Output: The credentials disappear from VCAP_SERVICES.
2. Delete the service instance.
   - Command: `cf delete-service <serviceInstanceName>`
     Output: Container will be permanently deleted.
   - Command: `cf services`
     Output: The service instance entry will not be found in the output.

5 Object Store Service on GCP

5.1 Configure Object Store to use GCP

Object store service on GCP provides GCS bucket as object store resource.

Prerequisites

Ensure that you have space developer privileges in an org and space.

Context

The Objectstore service supports the following operations:

- Create GCS bucket (service instance) - GCS buckets are managed in the Object store service's GCP project.
- Delete GCS bucket.
- Create credentials to access GCS bucket using the `cf bind-service` and `create-service-key` call.
- Delete credentials using the `cf unbind-service` and `delete-service-key` call.

i Note

- Number of service instances that can be created per Cloud Foundry org: 100
- Number of service instance bindings that can be created per service instance in Cloud Foundry: 5
- To perform object related operations (upload, download, delete etc) users can directly make calls to GCP using the GCP SDKs.
- Objectstore service supports only REGIONAL storage class objects.
- Objectstore service is a paid service, and the plans are sold as units of 100GBs. For more information, see SAP Cloud Platform Market Place. When user creates a service instance, a GCP GCS bucket will be created, and user can store all the units in a single GCS bucket or in multiple GCS buckets (limits on buckets per org apply as mentioned).
- Creating a new service instance for every purchased unit of Objectstore service is not advised.

Procedure

1. Look for available services on Cloud Foundry using the command `cf marketplace`.

   ```
   Service: objectstore
   Plans: gcs-standard
   Description: Objectstore service
   ```

   A service called ‘objectstore’ with plan ‘gcs-standard’ is available on cloud foundry.
2. Create a service instance of the above service and plan using the command `cf create-service objectstore gcs-standard <serviceInstanceName>`.
3. List the services available using the command `cf services`.

   ```
   Name || Service || Plan || Bound apps || Last operation
   <serviceInstanceName> || objectstore || gcs-standard
   ```

4. Get credentials.

   For bucket and object operations, the application requires certain parameters (bucket, base64EncodedPrivateKeyData, projectId, keyAlgo and region). There are two ways user can get the credentials. One is by Binding the service instance to an application directly. Other way is to use Service-Key concept from Cloud Foundry.
5. Bind an Object Store service instance to an application: `cf bind-service <ApplicatioName> <serviceInstanceName>`
6. The application binds to the service instance. Then execute the following command. `cf restage <ApplicationName>`
7. The application is shown as bound with the service. Use the command `cf services` to obtain the following output.

   Output:

   ```
   Name || Service || Plan || Bound apps || Last operation
   <serviceInstanceName> || objectstore || gcs-standard || <ApplicatioName>
   ```

8. Read the environment variables using the command `cf env <ApplicatioName>`

   ```
   System-Provided:
   {
    "VCAP_SERVICES": {
     "objectstore": [
      {
       "credentials": {
        "base64EncodedPrivateKeyData":
   "<base64_encoded_service_account_credentials>",
        "projectId": "<project_id>",
        "keyAlgo": "<service_account_key_generation_algo>",
        "region": "<region_of_bucket>",
        "bucket": "<some_bucket_name>"
       },
       "label": "objectstore",
       "name": "<serviceInstanceName>",
       "plan": "gcs-standard",
       "provider": null,
       "syslog_drain_url": null,
       "tags": [
        "blobStore",
        "objectStore"
       ],
   ```

```
     "volume_mounts": []
    }
   ]
  }
 }
```

When binding your application with a service instance, cloud foundry sets the environment variable VCAP_SERVICES. From VCAP_SERVICES, your application can read the following:

- base64EncodedPrivateKeyData
- projectId
- keyAlgo
- region
- bucket

5.2 Use Service Keys to Access Object Store Service Instance

Applications use service keys to access Object Store service instance.

Context

Follow the steps below to create a service key:

Procedure

1. Create a service key using the command `cf create-service-key< serviceInstanceName> <KeyName>`.
2. View the generated service key using the command `cf service-key <serviceInstanceName> <KeyName>`.

   The sample output is as shown below:

   ```
   Getting key <KeyName> for service instance <serviceInstanceName> as admin...
   {
    "base64EncodedPrivateKeyData":
   "<base64_encoded_service_account_credentials>",
    "projectId": "<project_id>",
    "keyAlgo": "<service_account_key_generation_algo>",
    "region": "<region_of_bucket>",
    "bucket": "<some_bucket_name>"
   }
   ```

   A json file with the following fields appears in the output:
   - base64EncodedPrivateKeyData

    - projectId
    - keyAlgo
    - region
    - bucket

5.3 Supported Operations

Here are the list of supported operations:

Operation supported on Buckets

List Bucket (List of objects in a bucket)

Operations supported on Objects

- Get Object
- Get Object ACL
- Put Object
- Update Object's metadata
- Copy Object from one bucket to another
- Delete Object

Users can make use of SDKs provided by GCP to access the GCS bucket.

Related Information

Buckets
Objects

5.4 Java Code Snippets

You can use the following sample code snippets as a reference:

- Read VCAP_SERVICES

  Sample Code

  ```
  @Value("$
  {vcap.services.objectstore.credentials.base64EncodedPrivateKeyData}")
  String base64EncodedPrivateKeyData;
  @Value("${vcap.services.objectstore.credentials.projectId}")
  String projectId;
  @Value("${vcap.services.objectstore.credentials.keyAlgo}")
  String keyAlgo;
  ```

```
@Value("${vcap.services.objectstore.credentials.region}")
String bucketRegion;
@Value("${vcap.services.objectstore.credentials.bucket}")
String bucketName;
```

- Get Storage Client

Sample Code

```
String credentials =
Base64.decodeBase64(base64EncodedPrivateKeyData).toString();
InputStream is = new
ByteArrayInputStream(credentials.getBytes(Charset.forName("UTF-8")));
ServiceAccountCredentials sac = ServiceAccountCredentials.fromStream(is);
Storage storageClient = StorageOptions.newBuilder()
        .setProjectId(projectId)
        .setCredentials(sac)
        .build()
        .getService();
```

- Upload an object

Sample Code

```
BlobId blobId = BlobId.of(bucketName, <path_to_object>);
// Here, <path_to_object> = <path1>/<path2>/objectName;
BlobInfo blobInfo = null;
blobInfo = BlobInfo.newBuilder(blobId)
    .setContentType("text/plain")
    .build();
String content = <content to upload>;
storageClient.create(blobInfo, content.getBytes(Charset.forName("UTF-8")));
```

- Download an object

Sample Code

```
Blob blob = null;
BlobId blobId = BlobId.of(bucketName, <path_to_object>);
// Here, <path_to_object> = <path1>/<path2>/objectName;
String blobContent = null;
blob = storageClient.get(blobId);
blobContent = new String(blob.getContent(), Charset.forName("UTF-8"));
```

- Delete an object

Sample Code

```
BlobId blobId = BlobId.of(bucketName, <path_to_object>);
// Here, <path_to_object> = <path1>/<path2>/objectName;
storageClient.delete(blobId);
```

5.5 Sample Script Snippets to Overcome Service Binding Limitations

There is a limitation on the number of service instance bindings that can be created per service instance in Cloud Foundry to five. If you have a use case where you need more than five bindings, you should use `cf service-keys` concept from Cloud Foundry.

Suppose multiple applications have to use a particular service instance and if the cf bind-service command is used, then you can exhaust the Binding limits. To overcome that, make use of cf service-key concept.

Also, if you follow blue-green deployment of application to overcome downtime, there is a time when two bindings for a service instance is needed at a time - leading to exhaust binding limits if other applications are also bound. In this case also, below steps can be used in the script to overcome binding limitations. Here is how you can make use of service-key functionality in a programmatic way to overcome binding limitations:

1. Create a service key.
2. Retrieve credentials from the service key and store it in a variable using the command

```
credentialJSON=`cf service-key <servicenInstanceName> <KeyName> | sed -n '1,2!
p'`
```

3. Store the credentials obtained in above step in variables.

```
base64EncodedPrivateKeyData=`echo "${credentialsJSON}" | jq -r
'.base64EncodedPrivateKeyData'`
projectId=`echo "${credentialsJSON}" | jq -r '.projectId'`
keyAlgo=`echo "${credentialsJSON}" | jq -r '.keyAlgo'`
region=`echo "${credentialsJSON}" | jq -r '.region'`
bucket=`echo "${credentialsJSON}" | jq -r '.bucket'`
```

4. Pass the credentials stored in above variables to your application.
   For CF application, you can pass the above credentials to an application by setting environment variables. To pass the variables as environment variables using application's manifest.yml file, make following entries in your manifest.yml file:

```
env:

vcap.services.<serviceInstanceName>.credentials.base64EncodedPrivateKeyData:
    vcap.services.<serviceInstanceName>.credentials.projectId:
    vcap.services.<serviceInstanceName>.credentials.keyAlgo:
    vcap.services.<serviceInstanceName>.credentials.region:
    vcap.services.<serviceInstanceName>.credentials.bucket:
```

   And in the script, you can update these environment variables in manifest.yml using the following commands before deployment:

```
sed -i "s/
vcap.services.<serviceInstanceName>.credentials.base64EncodedPrivateKeyData.*/
vcap.services.<serviceInstanceName>.credentials.base64EncodedPrivateKeyData:
$base64EncodedPrivateKeyData /g" ./manifest.yml
sed -i
"s~vcap.services.<serviceInstanceName>.credentials.projectId.*~vcap.services.<
serviceInstanceName>.credentials.projectId: $projectId ~g" ./manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.keyAlgo.*/
vcap.services.<serviceInstanceName>.credentials.keyAlgo: $keyAlgo /g" ./
manifest.yml
sed -i "s/vcap.services.<serviceInstanceName>.credentials.region.*/
vcap.services.<serviceInstanceName>.credentials.region: $region /g" ./
manifest.yml
```

```
sed -i "s/vcap.services.<serviceInstanceName>.credentials.bucket.*/
vcap.services.<serviceInstanceName>.credentials.bucket: $bucket /g" ./
manifest.yml
```

6 Data Protection and Privacy

Governments place legal requirements on industry to protect data and privacy. We provide features and functions to help you meet these requirements.

i Note

SAP does not provide legal advice in any form. SAP software supports data protection compliance by providing security features and specific data protection-relevant functions, such as simplified blocking and deletion of personal data. In many cases, compliance with applicable data protection and privacy laws will not be covered by a product feature. Definitions and other terms used in this document are not taken from a particular legal source.

The following sections provide information about the Object Store service. For the central data protection and privacy statement for SAP Cloud Platform, see Data Protection and Privacy

User Consent

We assume that software operators, such as SAP customers, collect and store the consent of data subjects, before collecting personal data from data subjects. A data privacy specialist can later determine whether data subjects have granted, withdrawn, or denied consent.

The Object Store service does not provide any support for collecting and storing the consent of data subjects for applications built on SAP Cloud Platform. It is the responsibility of your developers to provide such support.

Read-Access Logging and Change Log

Audit logs are available for all broker related operations of Object Store (like create service, bind service, unbind service, update service, create service-key, delete service) on all SAP supported infrastructures.

In case of AWS, Microsoft Azure and GCP, a CF application that is bound to an ObjectStore instance, gets a direct URL to the underlying Object tStore (S3/Azure/GCS), along with relevant access to perform object related operations (like upload, download, delete objects) directly. Therefore, the ObjectStore-as-a-Service cannot audit log any of the object related operations. However on SAP DC, the service intercepts all object related calls to Swift. Therefore, each of the object related calls are audit logged in addition to the service broker operations. For the object related operations, please note that the audit logs include the technical user details and not the actual Cloud Foundry user.

The Object Store service doesn’t deal with any personal data. However, the service has no control over the kind of data that is written to an Object Store (S3/Azure/GCS/Swift).

To extract the audit logs, you need to create a BCP ticket against Object store in the component BC-NEO-CF-OSAAS. The audit log data stored for your account will be retained for 30 days, after which it will be deleted.

Erasure

When handling personal data, consider the legislation in the different countries where your organization operates. After the data has passed the end of purpose, regulations may require you to delete the data. However, additional regulations may require you to keep the data longer. During this period you must block access to the data by unauthorized persons until the end of the retention period, when the data is finally deleted.

You can delete your Object Store service, and therefore, all the data stored in your databases. To do so, navigate to your sub account using the procedure Navigate to Global Accounts, Subaccounts, Orgs, and Spaces in the Cockpit and delete the service from the *Overview* page.

If you want to delete the objects from S3 buckets or containers, you can do so by making object delete calls. If you want to delete the bucket or container, then you can make a `cf delete-service` instance call. The time taken to delete the container/S3 bucket depends on the number of objects present in the container/S3 bucket.

There is no data backup in Objectstore, as one of the common use case of objectstore is to store backup of other data. However there is data replication that happen at the IaaS layer (AWS infrastructure, Openstack infrastructure). The replicas are also deleted when the object delete or the container/S3 bucket delete call is performed.

Glossary

Term	Definition
Consent	The action of the data subject confirming that the usage of his or her personal data shall be allowed for a given purpose. A consent functionality allows the storage of a consent record in relation to a specific purpose and shows if a data subject has granted, withdrawn, or denied consent.
Deletion	Deletion of **personal data** so that the data is no longer available.
Personal data	Any information relating to an identified or identifiable natural person ("data subject"). An identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural, or social identity of that natural person

Term	Definition
Retention period	The period of time between the end of the last business activity involving a specific object (for example, a business partner) and the deletion of the corresponding data, subject to applicable laws. The retention period is a combination of the residence period and the blocking period.
Sensitive personal data	A category of personal data that usually includes the following type of information: • Special categories of personal data, such as data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, trade union membership, genetic data, biometric data, data concerning health or sex life or sexual orientation, or personal data concerning bank and credit accounts. • Personal data subject to professional secrecy • Personal data relating to criminal or administrative offenses • Personal data concerning insurances and bank or credit card accounts

7 Data Encryption Strategy

Your crucial information stored in the database is maintained in a highly secure manner as we at SAP use the encryption capabilities provided by the underlying IaaS providers AWS, Azure, GCP and SAP DC. Encryption details for each of the IaaS provides are described below:

Data Center	Data stored on persistent disk	Backup data	Reference
Amazon Web Service	Data is stored on encrypted Elastic Block Store (EBS) volumes. EBS uses Amazon Key Management Service (AWS KMS) customer master keys (CMKs) to encrypt volumes/disks. AWS manages the key per account and this key is used for all encryptions in that account.	Encrypted EBS volume snapshots stored on AWS S3. EBS uses Amazon Key Management Service (AWS KMS) customer master keys (CMKs) to encrypt snapshots.	EBS Encryption
Microsoft Azure	Data is stored on encrypted Managed Disks. Azure SSE (Storage Service Encryption) provides encryption-at-rest for managed disks.	Encrypted managed disk snapshots stored on Azure Zone Redundant Storage (ZRS).	Azure Manage Disks
GCP	Data is stored on encrypted Persistent Disks. Persistent disk is encrypted with system-defined keys (managed by GCP).	Encrypted persistent disk snapshots are taken.	Create Snapshots
SAP DC	Data is stored on persistent volume/disks.	Encrypted backups are uploaded to SAP Swift storage. SAP controls the keys for SAP Cloud Platform in SAP data centers.	

8 Frequently Asked Questions

In this section you will find answers to all your basic queries about Object Store on SAP Cloud Platform in Cloud Foundry environment.

Why do we need object store against file store / block storage?

	Object Store	Block Store
Performance	Performs best for big content and high stream throughput	Strong performance with database and transactional data
Geography	Data can be stored across multiple regions	The further the distance between storage and application, the higher the latency
Scalability	Can scale infinitely to petabytes and beyond	Addressing requirements limit scalability
Analytics	Customizable metadata allows data to be easily organized and retrieved	No metadata

Can object store be my primary database?

Object storage can be used either as primary storage or as external storage. When it is used as primary storage, the object store only store the file content by using a unique identifier. No metadata are stored in the object store: no files names, no directory structures, etc.

What are the regions in which Object Store is available?

Infrastructure	Region
AWS	Europe (Frankfurt) US East (VA) Brazil (São Paulo) Japan (Tokyo)

Infrastructure	Region
GCP	US Central (IA)
Azure	Europe (Netherlands)

Is my data encrypted?

Your crucial information stored in the database is maintained in a highly secure manner as we at SAP use the encryption capabilities provided by the underlying IaaS providers AWS, Azure, and GCP. Encryption details for each of the IaaS provides are described Data Protection and Privacy [page 31].

How to use Postman to make object related calls on AWS?

To perform REST calls (GET, PUT DELETE) on the AWS S3 bucket using Postman, following information needs to be provided:

- **HTTP Method**: GET/PUT/DELETE
- **URL**: https://<host>/<bucketName>/<objectName>
- **AWS AccessKey**: access_key_id - The key that get from the credentials json (on binding or creating service key).
- **AWS SecretKey**: secret_access_key - The key that get from the credentials json (on binding or creating service key).
- **AWS Region**: region - The region name that get from the credentials json (on binding or creating service key).
- **Service Name**: s3 - This key is not passed in the credentials json, since Objectstore uses only S3, value to be used will be a constant : "s3"
  For GET Objects in a bucket (List objects API), you can use following URL: https://<host>/bucketName>

Could you also guide if there is a way to view the uploaded images(once done) on the SAP cloud cockpit?

We do not have a UI console for Objectstore service instance today, so sorry SAP Cloud cockpit will not help you to view the uploaded images. However you can make a List Objects API (GET call mentioned above) from Postman or from code, to view the uploaded objects.

To successfully create/upload blob on Azure, follow the below mentioned steps:

1. Use the Request Method : PUT

2. Blob end point : <container_uri>/<objectname>?<sass-token> <container_uri> as obtained from the credentials after binding to object store service instance.

3. Headers : x-ms-blob-type:BlockBlob

Note : The http header "x-ms-blob-type" with value "BlockBlob" is mandatory to successfully upload blob using this request.

How to enable a particular plan in a particular region

ObjectStore as a Service is available on Europe (Frankfurt) region.

It might be the case that you are trying to look for the service in Europe (Rot) region instead of Europe (Frankfurt) region.

The sub-account is in Europe (Frankfurt) region. Hence, you aren't able to find the service.

Request you to please try looking into the correct region. Kindly follow the steps below:

1. Click on the global account
2. Click on the sub-account tile with name : TCS Innovation IoT AE
3. Click on *Spaces* on the left menu bar
4. Create a space if it doesn't exist already
5. Click on *Service Marketplace* under *Services* on the left menu bar.
6. You should be able to see a tile named *ObjectStore*.

If the above mentioned steps don't help, then it could be possible that the service plan isn't enabled in a particular sub-account.

This can be verified by doing the following:

1. Click on the global account
2. Click on *Entitlements* tab on the left menu bar
3. Look for *Object Store*
4. Verify if following entry is there - ["Your sub-account name", Europe (Frankfurt), s3-standard].

If the entry isn't there, you may have to Edit the page to add the plan in the sub-account.

This would allow you to see the service in the sub-account.

I'm facing some technical problem. How do I seek support?

For any technical issues, please create a ticket in the component **BC-NEO-BS-OBJECTSTORE**.

www.sap.com/contactsap

THE BEST RUN