# Cloudian Kubernetes S3 Operator Quick Start

The Cloudian Kubernetes S3 Operator makes it easy for Kubernetes-orchestrated applications to use Cloudian HyperStore for S3-compatible object storage. With the Cloudian S3 Operator, you can use familiar Kubernetes resources and tooling to make HyperStore object storage buckets dynamically available to containerized application that have S3 client capabilities. Such applications can then read and write to those object storage buckets using standard S3 API calls.

This document describes how to deploy the Cloudian S3 Operator, create the relevant Kubernetes resources, and have Kubernetes applications use Cloudian S3 object storage. The document assumes a working knowledge of Kubernetes and its tooling.



## 1. Requirements

- HyperStore version 7.2 or newer
- A Kubernetes cluster
- A containerized application with S3 client capabilities, that you will run in your Kubernetes cluster

## 2. Preparing Your HyperStore System for Kubernetes Access

If you have not already done so, complete the following basic setup tasks in your HyperStore system before deploying the Cloudian S3 Operator in your Kubernetes cluster:

- Create **at least one storage policy**.
- Create **a HyperStore user account**, to be used by the Cloudian S3 Operator. This needs to be a regular user (what Amazon Web Services calls an "account root user"), not an IAM user.
    - **Optionally, create a bucket** under this account if you will want to support "brownfield" object storage for your Kubernetes cluster (where multiple object bucket claims share the same pre-existing bucket). If you intend to support only "greenfield" object storage (where each object bucket claim gets its own newly created bucket), you do not need to manually create a bucket -- instead, for each object bucket claim the Cloudian S3 Operator will dynamically create a new bucket under this user account. You will be able to configure brownfield and/or greenfield object storage classes for your Kubernetes users as described later in this document.
    - **Optionally, create an IAM user** under this account if you will want multiple object bucket claims to access buckets as the same shared pre-existing IAM user. If you prefer to always have the Cloudian S3 Operator dynamically create a new IAM user under this user account for each object bucket claim (and automatically delete the IAM user when the claim ends), you do not need to manually create an IAM user. You will be able to control whether or not the Cloudian S3 Operator creates IAM users per claim when you configure object storage classes for your Kubernetes users as described later in this document. If you do manually create an IAM user for claims to share, be sure to give the IAM user permissions to access the bucket that you've manually created under the user root account (if you want to only support brownfield object storage) or permissions to access **all** buckets created under the user root account (if you want to support greenfield object storage in addition to or instead of brownfield storage).

For detail about performing these HyperStore tasks see your HyperStore product documentation.

When setting up the Cloudian S3 Operator in your Kubernetes environment you will need the following **information from your HyperStore system**:

- The name of the HyperStore service region in which you want the Cloudian S3 Operator to create buckets (or to use a pre-existing bucket).
- The S3 Service endpoint for that HyperStore service region.
- The IAM Service endpoint for the HyperStore system.
- The S3 credentials (access key ID and secret access key) for the HyperStore user account that the Cloudian S3 Operator will use to provision buckets.
- If you created an IAM user for object bucket claims to share, the S3 credentials for that IAM user.
- If you created a bucket for object bucket claims to share, the name of that bucket.
- If you want Cloudian S3 Operator generated buckets to use a HyperStore storage policy other than the default storage policy in your HyperStore system, you will need the storage policy ID (not the policy name but rather the system-generated policy ID). You can see a storage policy's ID in the CMC: **Cluster -> Storage Policies -> View/Edit**.

Note that your **Kubernetes cluster must be configured so that it can resolve HyperStore service endpoints**. Specifically it must be able to resolve the S3 Service endpoint (typically *s3-<regionname>.<domain>*), S3 Service endpoint wildcard (**.s3-<regionname>.<domain>*), and IAM Service endpoint (*iam.<domain>*).

# 3. Deploying the Cloudian S3 Operator in Your Kubernetes Cluster

Deploying the Cloudian Kubernetes S3 Operator in your Kubernetes cluster will enable you to use a combination of standard Kubernetes resources (Secret and StorageClass) and custom Kubernetes resources (ObjectBucket and ObjectBucketClaim) to automate the provisioning of Cloudian HyperStore object storage for your containerized applications. The ObjectBucket/ObjectBucketClaim provisioning model for Kubernetes is analogous to how the standard PersistentVolume and PersistentVolumeClaim (PV/PVC) resources work for block and file storage, except with custom resources tailored to the requirements of object storage.

In the steps below it is assumed that you are on a machine with internet access. Also, here and throughout the remainder of the instructions in this document it is assumed that the Kubernetes command line tool *kubectl* is in your path and that you have permissions to run it.

First, run these commands to create the ObjectBucket and ObjectBucketClaim custom resource definitions in your Kubernetes cluster:

```
# kubectl apply -f https://raw.githubusercontent.com/kube-object-storage/lib-bucket-provisioner/master/deploy/crds/objectbucket_v1alpha1_objectbucket_crd.yaml

# kubectl apply -f https://raw.githubusercontent.com/kube-object-storage/lib-bucket-provisioner/master/deploy/crds/objectbucket_v1alpha1_objectbucketclaim_crd.yaml
```

These custom resource definitions (CRDs) are not specific to Cloudian HyperStore. They are general purpose CRDs for provisioning object storage for Kubernetes clusters. The Cloudian S3 Operator utilizes these CRDs.

Next, run this command to deploy the Cloudian S3 Operator in your Kubernetes cluster:

```
# kubectl apply -f https://raw.githubusercontent.com/cloudian/cloudian-s3-operator/hyperstore/examples/cloudian-s3-provisioner.yaml
```

This deploys a provisioner in the *cloudian-s3-operator* namespace.

# 4. Creating a Kubernetes Secret and a Storage Class for Cloudian Object Storage

## Create the Bucket Owner Secret

Next you will create a Kubernetes Secret that encapsulates the S3 security credentials of the HyperStore user account that the Cloudian S3 Operator will use when it provisions buckets (the account noted in the **"Requirements"** (page 1) section of

this document). When creating the Secret you will need the S3 access key ID and the secret access key to be base64 encoded, without any newline. You can create the base64 encodings by running the following commands:

```
# echo -n <raw access key id> | base64

# echo -n <raw secret access key> | base64
```

With the base64 encoded S3 credentials at hand, create a manifest file named *bucket-owner-secret.yaml* with the following content:

```
apiVersion: v1
kind: Secret
metadata:
  name: s3-bucket-owner
  namespace: cloudian-s3-operator
type: Opaque
data:
  AWS_ACCESS_KEY_ID: "<base64_encoded_access_key_id>"
  AWS_SECRET_ACCESS_KEY: "<base64_encoded_secret_access_key>"
```

Then create the resource in your cluster with this command:

```
# kubectl apply -f bucket-owner-secret.yaml
```

**Optionally** you can create a second Secret, in the same manner as described above, that encapsulates the S3 credentials of an IAM user that you have created in HyperStore under the bucket owner account. This is applicable only if you intend to have multiple object bucket claims use the same pre-existing IAM user when accessing HyperStore buckets -- rather than the Cloudian S3 Operator default behavior which is to dynamically create a new IAM user (under the bucket owner account) for each object bucket claim, and then automatically delete that IAM user when the claim ends.

## Create One or More Storage Classes

Next, create at least one Storage Class for Cloudian object storage. A Storage Class is a standard Kubernetes resource that you have likely used before for other types of storage, but in this case the specified provisioner will be the Cloudian S3 Operator and the parameters will be specific to the needs of object storage provisioning. If you wish you can create multiple Storage Classes for Cloudian object storage -- for example, one Storage Class for "greenfield" storage (each application pod deployment gets access to a newly created, empty bucket in Cloudian HyperStore) and one Storage Class for "brownfield" storage (each application pod deployment gets access to an existing bucket in Cloudian HyperStore). Greenfield and brownfield Storage Class configurations are both described below.

**IMPORTANT !** For greenfield storage, when Object Bucket Claims (as described later in this document) use a greenfield Storage Class, the Cloudian S3 Operator dynamically creates new buckets in HyperStore. **The HyperStore administrator must not make any configuration changes to the greenfield buckets** that the Cloudian S3 Operator creates in HyperStore. For example the HyperStore administrator must not enable versioning on greenfield buckets, or auto-tiering, or any other bucket level configuration change.

**To create a Storage Class for greenfield Cloudian object storage**, create a manifest file with a suggestive name such as *cloudian-greenfield-storageclass.yaml* with the following content:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage class name>   [1]
provisioner: cloudian-s3.io/bucket
parameters:
  region: <cloudian hyperstore service region>   [2]
  secretName: s3-bucket-owner   [3]
  secretNamespace: cloudian-s3-operator
```

```
  s3Endpoint: <S3 service endpoint>   [4]
  iamEndpoint: <IAM service endpoint>   [5]
  storagePolicyId: <optional storage policy ID>   [6]
  createBucketUser: <optional override of creating an IAM user per bucket claim>   [7]
  bucketClaimUserSecretName: <optional dedicated Secret for bucket access>   [7]
  bucketClaimUserSecretNamespace: <only if bucketClaimUserSecretName is set>   [7]
  iamPolicy: <optional IAM policy document>   [8]
reclaimPolicy: Delete   [9]
```

1. Name to give to this Storage Class, for example *cloudian-greenfield*. This will by referenced object bucket claims that use this Storage Class.
2. Name of the Cloudian HyperStore service region in which you want greenfield object storage buckets to be created.
3. This references the name of the bucket owner Secret that you created previously.
4. The S3 Service endpoint of the HyperStore service region named by the region parameter. Specify the endpoint in the form of a URL -- for example *"http://s3-region1.mycloudianhyperstore.com"*. You must include the port number at the end of the URL (*:<#>*) if the port number is anything other than 80 (in the example the port is 80 and so is not included in the URL). Important: Be sure your Kubernetes cluster can resolve this endpoint.
5. The IAM Service endpoint of your HyperStore system. Specify the endpoint in the form of a URL -- for example *"http://iam.mycloudianhyperstore.com:16080"*. You must include the port number at the end of the URL if the port number is anything other than 80 (in the example the port is 16080 and so is included in the URL). Important: Be sure your Kubernetes cluster can resolve this endpoint.
6. If you want newly created buckets to use a HyperStore storage policy other than the default storage policy, specify the storage policy ID here (for background information see the **"Requirements"** (page 1) section). If you want newly created buckets to use the default HyperStore storage policy, omit the *storagePolicyId* parameter.
7. By default the Cloudian S3 Operator creates a new IAM user (with its own unique S3 credentials) for each object bucket claim. To use this default behavior, omit the *createBucketUser*, *bucketClaimUserSecretName*, and *bucketClaimUserSecretNamespace* parameters. If instead you want all object bucket claims to access HyperStore buckets as the same shared user, using the same shared S3 credentials, set the *createBucketUser* parameter to *"no"*(including the quote marks). If you want that one shared user for bucket access to be the same account root user used by the Cloudian S3 Operator to provision buckets, omit the *bucketClaimUserSecretName* and *bucketClaimUserSecretNamespace* parameters. If you want that one shared user for bucket access to be a pre-existing IAM user that you have created under the bucket owner root account, and you have created a Secret that encapsulates that IAM user's S3 credentials (as described in in **"Create the Bucket Owner Secret"** (page 2)), use the *bucketClaimUserSecretName* and *bucketClaimUserSecretNamespace* parameters to reference that Secret.
8. Include the *iamPolicy* parameter only if you are having the Cloudian S3 Operator automatically create a new IAM user for each object bucket claim (i.e. in the Storage Class the *createBucketUser* parameter is omitted or explicitly set to "yes") and if you want those dynamically created IAM users to have something less than full read and write permissions on the bucket (which is the default permission that the Cloudian S3 Operator grants to IAM users that it creates). If you do include the *iamPolicy* parameter, set it as a JSON-formatted IAM policy document. You do not need to include a "Resource" field in the document, since the policy will automatically apply only to the provisioned bucket. For example, to grant read-only access to the bucket:

```
iamPolicy: |
  {
    "Version": "2012-10-17",
    "Statement": [{
      "Sid": "ReadOnly",
      "Effect": "Allow",
      "Action": ["s3:HeadObject", "s3:ListBucket", "s3:GetObject"]
  }]
}
```

9. With *reclaimPolicy* set to *Delete*, when the object bucket claim is deleted the bucket (and data within the bucket) will be automatically deleted from HyperStore. Alternatively you could set this to *Retain* in which case when the object bucket claim is deleted, the bucket (and data within the bucket) will **not** be deleted.

Then create the resource in your cluster with this command:

```
# kubectl apply -f <manifest filename>
```

**To create a Storage Class for brownfield Cloudian object storage**, create a manifest file with a suggestive name such as *cloudian-brownfield-storageclass.yaml* with the following content:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage class name>   [1]
provisioner: cloudian-s3.io/bucket
parameters:
  region: <cloudian hyperstore service region>   [2]
  secretName: s3-bucket-owner   [3]
  secretNamespace: cloudian-s3-operator
  bucketName: <existing bucket in HyperStore>   [4]
  s3Endpoint: <S3 service endpoint>   [5]
  iamEndpoint: <IAM service endpoint>   [5]
  createBucketUser: <optional override of creating an IAM user per pod deployment>   [6]
  bucketClaimUserSecretName: <optional dedicated Secret for bucket access>   [6]
  bucketClaimUserSecretNamespace: <only if bucketClaimUserSecretName is set>   [6]
  iamPolicy: <optional IAM policy document>   [6]
```

1. Name to give to this Storage Class, for example *cloudian-brownfield*. This will by referenced object bucket claims that use this Storage Class.
2. The Cloudian HyperStore service region in which the bucket named by the *bucketName* parameter is located.
3. This references the name of the bucket owner Secret that you created previously.
4. This is the name of an existing bucket in HyperStore, which will be used for all bucket claims associated with this Storage Class. This must be a bucket that is owned by the HyperStore account root user whose S3 credentials are encapsulated in the bucket owner Secret.
5. For information about formatting the S3 Service endpoint and IAM Service endpoint, see the preceding description of a greenfield Cloudian object storage class.
6. For information about the *createBucketUser*, *bucketClaimUserSecretName*, *bucketClaimUserSecretNamespace*, and *iamPolicy* parameters, see the preceding description of a greenfield Cloudian object storage class.

Then create the resource in your cluster with this command:

```
# kubectl apply -f <manifest filename>
```

## 5. Having a Containerized Application Use Cloudian Object Storage

Once the Cloudian S3 Operator has been deployed in the cluster and a bucket owner Secret and one or more Storage Classes have been created, then DevOps users can easily utilize Cloudian S3-compatible object storage on demand by creating an Object Bucket Claim (OBC) for their containerized application to use. This is very much like a Persistent Volume Claim (PVC) -- a mechanism with which most DevOps users will be familiar -- except an OBC is simpler in its contents than a PVC.

When the Cloudian S3 Operator detects a new OBC it will either provision a new bucket in HyperStore (in the case of a greenfield OBC) or gain access to an existing HyperStore bucket (in the case of a brownfield OBC). It also generates Kubernetes resources corresponding to the OBC -- an Object Bucket (OB), a Config Map, and a Secret -- that can then be consumed by application pods.

## Create an Object Bucket Claim

A greenfield OBC -- for which a new HyperStore bucket will be created -- references the greenfield object storage class, and provides information about how to name the newly created bucket. A brownfield OBC -- for which an existing HyperStore bucket will be used -- references the brownfield object storage class, and does not specify any bucket naming information.

**To create an Object Bucket Claim for greenfield Cloudian object storage**, create a manifest file with a suggestive name such as *greenfield-bucket.yaml* with the following content:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <OBC name>   [1]
  namespace: <optional OBC namespace>   [2]
spec:
  generateBucketName: <greenfield bucket name prefix>   [3]
  bucketName: <greenfield bucket full name>   [3]
  storageClassName: <name of the greenfield object storage class>   [4]
```

1. Name to give to this Object Bucket Claim, for example *greenfield-bucket*. This same name will be applied to the ConfigMap and Secret that the Cloudian S3 Operator will generate for this OBC and which will be referenced by application pods using this OBC.
2. Optionally, the namespace for this OBC. If this is specified it should be the same namespace as will be used by application pods that use this OBC.
3. For a greenfield OBC, you can do either of the following:

   - Use the *generateBucketName* parameter to specify a bucket name prefix to which the Cloudian S3 Operator will append a random string to automatically create the full bucket name, and omit the *bucketName* parameter (a good approach for ensuring that the bucket has a unique name within HyperStore). Your *generateBucketName* value must be limited to lower case letters, numbers, dashes, or periods, and must not exceed 26 characters.
   - Omit the *generateBucketName* parameter and use the *bucketName* parameter to specify a full bucket name to use for the bucket when it is created. If you mistakenly include both parameters and assign values to both, the *bucketName* parameter will override the *generateBucketName* parameter. Your *bucketName* value must be limited to lower case letters, numbers, dashes, or periods; must be at least 3 and not more than 63 characters long; and must be unique within the HyperStore system.

4. Name of the greenfield storage class that this OBC will use, for example *cloudian-greenfield*.

Then create the resource with this command:

```
# kubectl apply -f <manifest filename>
```

**To create an Object Bucket Claim for brownfield Cloudian object storage**, create a manifest file with a suggestive name such as *brownfield-bucket.yaml* with the following content:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <OBC name>   [1]
  namespace: <optional OBC namespace>   [2]
spec:
  storageClassName: <name of the brownfield object storage class>   [2]
```

1. Name to give to this Object Bucket Claim, for example *brownfield-bucket*. This same name will be applied to the ConfigMap and Secret that the Cloudian S3 Operator will generate for this OBC and which will be referenced by application pods using this OBC.

2. Optionally, the namespace for this OBC. If this is specified it should be the same namespace as will be used by application pods that use this OBC.
3. Name of the storage class that this OBC will use, for example *cloudian-brownfield*. Note that there is no need to specify a bucket name in the OBC -- instead the bucket name is specified within the configuration of the brownfield storage class that the OBC is using.

Then create the resource with this command:

```
# kubectl apply -f <manifest filename>
```

### Creating an Application Pod That Uses the Object Bucket Claim

To have an application pod use an Object Bucket Claim, when you create the application pod include the following lines within the *containers* section of the pod's manifest file:

```
    envFrom:
    - configMapRef:
        name: <name of config map -- same as the name of the object bucket claim>
    - secretRef:
        name: <name of secret -- same as the name of the object bucket claim>
```

For example, if the OBC is named *cloudian-greenfield* the application pod's manifest would look something like this:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app-pod
  namespace: namespace-of-my-app-pod
spec:
  containers:
  - name: my-container
    image: image-for-my-container
    envFrom:
    - configMapRef:
        name: cloudian-greenfield
    - secretRef:
        name: cloudian-greenfield
```

## 6. Verifying Your Setup

If you wish you can run this simple test to verify that your setup works properly. To perform this test you will need the name of the greenfield object Storage Class that was created during the setup procedure; and you will need to be able to log into the Cloudian Management Console (CMC) as the user whose S3 security credentials were encapsulated in the bucket owner Secret during the setup procedure.

Create a manifest file named *test.yaml* that creates both an object bucket claim and a pod that uses that object bucket claim:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: test-setup-check
spec:
  generateBucketName: test-setup-check
  storageClassName: <name of your greenfield object storage class, e.g. cloudian-greenfield>
---
apiVersion: v1
kind: Pod
metadata:
  name: test-setup-check
spec:
```

```
  containers:
  - name: test-setup-check
    image: k8s.gcr.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    envFrom:
    - configMapRef:
        name: test-setup-check
    - secretRef:
        name: test-setup-check
  restartPolicy: Never
```

Create the resources in your cluster:

```
kubectl apply -f test.yaml
```

Wait a moment and then run this command:

```
kubectl get pods
```

In the command response, verify that the *test-setup-check* pod's status is Completed. Next view the log for the pod:

```
kubectl logs test-setup-check
```

You should see the environmental variables that will be used for accessing the Cloudian object storage system, including the name of the bucket.

Next, log into the CMC using the bucket owner account. In the **Buckets & Objects** section of the CMC you should see that the bucket has been created. In the **IAM** section you should see that an IAM user has been created with a name similar to the bucket name (if your greenfield object Storage Class uses the default behavior of creating a new IAM user for each OBC).

After confirming the presence of the bucket and the IAM user, delete the test OBC and pod:

```
kubectl delete -f test.yaml
```

Wait for a moment, then in the CMC you should see that the bucket and IAM user have been deleted.