Red Hat OpenShift Container Storage 4.7

Managing hybrid and multicloud resources

Hybrid and multicloud resource management for cluster and storage administrators

Last Updated: 2022-05-24

Red Hat OpenShift Container Storage 4.7 Managing hybrid and multicloud resources

Hybrid and multicloud resource management for cluster and storage administrators

Legal Notice

Abstract

This document explains how to manage storage resources across a hybrid cloud or multicloud environment.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 4

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 5

CHAPTER 1. ABOUT THE MULTICLOUD OBJECT GATEWAY 6

CHAPTER 2. ACCESSING THE MULTICLOUD OBJECT GATEWAY WITH YOUR APPLICATIONS 7
2.1. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE TERMINAL 7
2.2. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE MCG COMMAND-LINE INTERFACE 10

CHAPTER 3. ALLOWING USER ACCESS TO THE MULTICLOUD OBJECT GATEWAY CONSOLE 13

CHAPTER 4. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD 15
4.1. CREATING A NEW BACKING STORE 15
4.2. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD USING THE MCG COMMAND LINE INTERFACE 16
4.2.1. Creating an AWS-backed backingstore 17
4.2.2. Creating an IBM COS-backed backingstore 18
4.2.3. Creating an Azure-backed backingstore 20
4.2.4. Creating a GCP-backed backingstore 21
4.2.5. Creating a local Persistent Volume-backed backingstore 23
4.3. CREATING AN S3 COMPATIBLE MULTICLOUD OBJECT GATEWAY BACKINGSTORE 24
4.4. ADDING STORAGE RESOURCES FOR HYBRID AND MULTICLOUD USING THE USER INTERFACE 25
4.5. CREATING A NEW BUCKET CLASS 27
4.6. EDITING A BUCKET CLASS 28
4.7. EDITING BACKING STORES FOR BUCKET CLASS 28

CHAPTER 5. MANAGING NAMESPACE BUCKETS 30
5.1. AMAZON S3 API ENDPOINTS FOR OBJECTS IN NAMESPACE BUCKETS 30
5.2. ADDING A NAMESPACE BUCKET USING THE MULTICLOUD OBJECT GATEWAY CLI AND YAML 31
5.2.1. Adding an AWS S3 namespace bucket using YAML 31
5.2.2. Adding an IBM COS namespace bucket using YAML 33
5.2.3. Adding an AWS S3 namespace bucket using the Multicloud Object Gateway CLI 35
5.2.4. Adding an IBM COS namespace bucket using the Multicloud Object Gateway CLI 37

CHAPTER 6. MIRRORING DATA FOR HYBRID AND MULTICLOUD BUCKETS 39
6.1. CREATING BUCKET CLASSES TO MIRROR DATA USING THE MCG COMMAND-LINE-INTERFACE 39
6.2. CREATING BUCKET CLASSES TO MIRROR DATA USING A YAML 39
6.3. CONFIGURING BUCKETS TO MIRROR DATA USING THE USER INTERFACE 40

CHAPTER 7. BUCKET POLICIES IN THE MULTICLOUD OBJECT GATEWAY 42
7.1. ABOUT BUCKET POLICIES 42
7.2. USING BUCKET POLICIES 42
7.3. CREATING AN AWS S3 USER IN THE MULTICLOUD OBJECT GATEWAY 43

CHAPTER 8. OBJECT BUCKET CLAIM 47
8.1. DYNAMIC OBJECT BUCKET CLAIM 47
8.2. CREATING AN OBJECT BUCKET CLAIM USING THE COMMAND LINE INTERFACE 49
8.3. CREATING AN OBJECT BUCKET CLAIM USING THE OPENSHIFT WEB CONSOLE 52
8.4. ATTACHING AN OBJECT BUCKET CLAIM TO A DEPLOYMENT 54
8.5. VIEWING OBJECT BUCKETS USING THE OPENSHIFT WEB CONSOLE 55
8.6. DELETING OBJECT BUCKET CLAIMS 56

CHAPTER 9. CACHING POLICY FOR OBJECT BUCKETS 58

9.1. CREATING AN AWS CACHE BUCKET 58
9.2. CREATING AN IBM COS CACHE BUCKET 60
CHAPTER 10. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE BY ADDING ENDPOINTS . . 63
10.1. S3 ENDPOINTS IN THE MULTICLOUD OBJECT GATEWAY 63
10.2. SCALING WITH STORAGE NODES 63
CHAPTER 11. AUTOMATIC SCALING OF MULTICLOUD OBJECT GATEWAY ENDPOINTS 67
CHAPTER 12. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT 68

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Do let us know how we can make it better. To give feedback:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. ABOUT THE MULTICLOUD OBJECT GATEWAY

The Multicloud Object Gateway (MCG) is a lightweight object storage service for OpenShift, allowing users to start small and then scale as needed on-premise, in multiple clusters, and with cloud-native storage.

CHAPTER 2. ACCESSING THE MULTICLOUD OBJECT GATEWAY WITH YOUR APPLICATIONS

You can access the object service with any application targeting AWS S3 or code that uses AWS S3 Software Development Kit (SDK). Applications need to specify the MCG endpoint, an access key, and a secret access key. You can use your terminal or the MCG CLI to retrieve this information.

For information on accessing the RADOS Object Gateway S3 endpoint, see Chapter 12, *Accessing the RADOS Object Gateway S3 endpoint*.

Prerequisites

- A running OpenShift Container Storage Platform
- Download the MCG command-line interface for easier management:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

- Alternatively, you can install the **mcg** package from the OpenShift Container Storage RPMs found at Download RedHat OpenShift Container Storage page .

You can access the relevant endpoint, access key, and secret access key two ways:

- Section 2.1, “Accessing the Multicloud Object Gateway from the terminal”
- Section 2.2, “Accessing the Multicloud Object Gateway from the MCG command-line interface”

Accessing the MCG bucket(s) using the virtual-hosted style

> **Example 2.1. Example**
>
> If the client application tries to access https://<bucket-name>.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
>
> where **<bucket-name>** is the name of the MCG bucket
>
> For example, https://mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
>
> A DNS entry is needed for **mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com** to point to the S3 Service.

IMPORTANT

Ensure that you have a DNS entry in order to point the client application to the MCG bucket(s) using the virtual-hosted style.

2.1. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE TERMINAL

Procedure

Run the **describe** command to view information about the MCG endpoint, including its access key (**AWS_ACCESS_KEY_ID** value) and secret access key (**AWS_SECRET_ACCESS_KEY** value):

```
# oc describe noobaa -n openshift-storage
```

The output will look similar to the following:

```
Name:         noobaa
Namespace:    openshift-storage
Labels:       <none>
Annotations:  <none>
API Version:  noobaa.io/v1alpha1
Kind:         NooBaa
Metadata:
  Creation Timestamp:  2019-07-29T16:22:06Z
  Generation:          1
  Resource Version:    6718822
  Self Link:           /apis/noobaa.io/v1alpha1/namespaces/openshift-storage/noobaas/noobaa
  UID:                 019cfb4a-b21d-11e9-9a02-06c8de012f9e
Spec:
Status:
  Accounts:
    Admin:
      Secret Ref:
        Name:           noobaa-admin
        Namespace:      openshift-storage
  Actual Image:         noobaa/noobaa-core:4.0
  Observed Generation:  1
  Phase:                Ready
  Readme:

  Welcome to NooBaa!
  -----------------

  Welcome to NooBaa!
    -----------------
    NooBaa Core Version:
    NooBaa Operator Version:

    Lets get started:

    1. Connect to Management console:

      Read your mgmt console login information (email & password) from secret: "noobaa-admin".

        kubectl get secret noobaa-admin -n openshift-storage -o json | jq '.data|map_values(@base64d)'

      Open the management console service - take External IP/DNS or Node Port or use port
forwarding:

        kubectl port-forward -n openshift-storage service/noobaa-mgmt 11443:443 &
        open https://localhost:11443

    2. Test S3 client:
```

```
   kubectl port-forward -n openshift-storage service/s3 10443:443 &
1
   NOOBAA_ACCESS_KEY=$(kubectl get secret noobaa-admin -n openshift-storage -o json | jq -r
'.data.AWS_ACCESS_KEY_ID|@base64d')
2
   NOOBAA_SECRET_KEY=$(kubectl get secret noobaa-admin -n openshift-storage -o json | jq -r
'.data.AWS_SECRET_ACCESS_KEY|@base64d')
   alias s3='AWS_ACCESS_KEY_ID=$NOOBAA_ACCESS_KEY
AWS_SECRET_ACCESS_KEY=$NOOBAA_SECRET_KEY aws --endpoint https://localhost:10443 --
no-verify-ssl s3'
   s3 ls


  Services:
   Service Mgmt:
    External DNS:
     https://noobaa-mgmt-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
     https://a3406079515be11eaa3b70683061451e-1194613580.us-east-
2.elb.amazonaws.com:443
    Internal DNS:
     https://noobaa-mgmt.openshift-storage.svc:443
    Internal IP:
     https://172.30.235.12:443
    Node Ports:
     https://10.0.142.103:31385
    Pod Ports:
     https://10.131.0.19:8443
   serviceS3:
    External DNS: 3
     https://s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
     https://a340f4e1315be11eaa3b70683061451e-943168195.us-east-2.elb.amazonaws.com:443
    Internal DNS:
     https://s3.openshift-storage.svc:443
    Internal IP:
     https://172.30.86.41:443
    Node Ports:
     https://10.0.142.103:31011
    Pod Ports:
     https://10.131.0.19:6443
```

1 access key (**AWS_ACCESS_KEY_ID** value)

2 secret access key (**AWS_SECRET_ACCESS_KEY** value)

3 MCG endpoint

NOTE

The output from the **oc describe noobaa** command lists the internal and external DNS names that are available. When using the internal DNS, the traffic is free. The external DNS uses Load Balancing to process the traffic, and therefore has a cost per hour.

2.2. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE MCG COMMAND-LINE INTERFACE

Prerequisites

- Download the MCG command-line interface:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

Procedure

Run the **status** command to access the endpoint, access key, and secret access key:

```
noobaa status -n openshift-storage
```

The output will look similar to the following:

```
INFO[0000] Namespace: openshift-storage
INFO[0000]
INFO[0000] CRD Status:
INFO[0003]   Exists: CustomResourceDefinition "noobaas.noobaa.io"
INFO[0003]   Exists: CustomResourceDefinition "backingstores.noobaa.io"
INFO[0003]   Exists: CustomResourceDefinition "bucketclasses.noobaa.io"
INFO[0004]   Exists: CustomResourceDefinition "objectbucketclaims.objectbucket.io"
INFO[0004]   Exists: CustomResourceDefinition "objectbuckets.objectbucket.io"
INFO[0004]
INFO[0004] Operator Status:
INFO[0004]   Exists: Namespace "openshift-storage"
INFO[0004]   Exists: ServiceAccount "noobaa"
INFO[0005]   Exists: Role "ocs-operator.v0.0.271-6g45f"
INFO[0005]   Exists: RoleBinding "ocs-operator.v0.0.271-6g45f-noobaa-f9vpj"
INFO[0006]   Exists: ClusterRole "ocs-operator.v0.0.271-fjhgh"
INFO[0006]   Exists: ClusterRoleBinding "ocs-operator.v0.0.271-fjhgh-noobaa-pdxn5"
INFO[0006]   Exists: Deployment "noobaa-operator"
INFO[0006]
INFO[0006] System Status:
INFO[0007]   Exists: NooBaa "noobaa"
INFO[0007]   Exists: StatefulSet "noobaa-core"
INFO[0007]   Exists: Service "noobaa-mgmt"
INFO[0008]   Exists: Service "s3"
INFO[0008]   Exists: Secret "noobaa-server"
INFO[0008]   Exists: Secret "noobaa-operator"
INFO[0008]   Exists: Secret "noobaa-admin"
INFO[0009]   Exists: StorageClass "openshift-storage.noobaa.io"
INFO[0009]   Exists: BucketClass "noobaa-default-bucket-class"
INFO[0009]   (Optional) Exists: BackingStore "noobaa-default-backing-store"
INFO[0010]   (Optional) Exists: CredentialsRequest "noobaa-cloud-creds"
INFO[0010]   (Optional) Exists: PrometheusRule "noobaa-prometheus-rules"
INFO[0010]   (Optional) Exists: ServiceMonitor "noobaa-service-monitor"
INFO[0011]   (Optional) Exists: Route "noobaa-mgmt"
INFO[0011]   (Optional) Exists: Route "s3"
INFO[0011]   Exists: PersistentVolumeClaim "db-noobaa-core-0"
INFO[0011]   System Phase is "Ready"
INFO[0011]   Exists:  "noobaa-admin"
```

```
#------------------#
#- Mgmt Addresses -#
#------------------#

ExternalDNS : [https://noobaa-mgmt-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
https://a3406079515be11eaa3b70683061451e-1194613580.us-east-2.elb.amazonaws.com:443]
ExternalIP  : []
NodePorts   : [https://10.0.142.103:31385]
InternalDNS : [https://noobaa-mgmt.openshift-storage.svc:443]
InternalIP  : [https://172.30.235.12:443]
PodPorts    : [https://10.131.0.19:8443]

#--------------------#
#- Mgmt Credentials -#
#--------------------#

email    : admin@noobaa.io
password : HKLbH1rSuVU0I/souIkSiA==

#----------------#
#- S3 Addresses -#
#----------------#

1
ExternalDNS : [https://s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
https://a340f4e1315be11eaa3b70683061451e-943168195.us-east-2.elb.amazonaws.com:443]
ExternalIP  : []
NodePorts   : [https://10.0.142.103:31011]
InternalDNS : [https://s3.openshift-storage.svc:443]
InternalIP  : [https://172.30.86.41:443]
PodPorts    : [https://10.131.0.19:6443]

#------------------#
#- S3 Credentials -#
#------------------#

2
AWS_ACCESS_KEY_ID     : jVmAsu9FsvRHYmfjTiHV
3
AWS_SECRET_ACCESS_KEY : E//420VNedJfATvVSmDz6FMtsSAzuBv6z180PT5c

#------------------#
#- Backing Stores -#
#------------------#

NAME                        TYPE     TARGET-BUCKET                                        PHASE   AGE
noobaa-default-backing-store   aws-s3   noobaa-backing-store-15dc896d-7fe0-4bed-9349-
5942211b93c9   Ready   141h35m32s

#------------------#
#- Bucket Classes -#
#------------------#

NAME                        PLACEMENT                                                PHASE   AGE
noobaa-default-bucket-class   {Tiers:[{Placement: BackingStores:[noobaa-default-backing-store]}]}
```

```
Ready   141h35m33s

#-----------------#
#- Bucket Claims -#
#-----------------#

No OBC's found.
```

1. endpoint
2. access key
3. secret access key

You now have the relevant endpoint, access key, and secret access key in order to connect to your applications.

> **Example 2.2. Example**
>
> If AWS S3 CLI is the application, the following command will list buckets in OpenShift Container Storage:
>
> ```
> AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY_ID>
> AWS_SECRET_ACCESS_KEY=<AWS_SECRET_ACCESS_KEY>
> aws --endpoint <ENDPOINT> --no-verify-ssl s3 ls
> ```

CHAPTER 3. ALLOWING USER ACCESS TO THE MULTICLOUD OBJECT GATEWAY CONSOLE

To allow access to the Multicloud Object Gateway Console to a user, ensure that the user meets the following conditions:

- User is in **cluster-admins** group.
- User is in **system:cluster-admins** virtual group.

Prerequisites

- A running OpenShift Container Storage Platform.

Procedure

1. Enable access to the Multicloud Object Gateway console.
   Perform the following steps once on the cluster :

   a. Create a **cluster-admins** group.

   ```
   # oc adm groups new cluster-admins
   ```

   b. Bind the group to the **cluster-admin** role.

   ```
   # oc adm policy add-cluster-role-to-group cluster-admin cluster-admins
   ```

2. Add or remove users from the **cluster-admins** group to control access to the Multicloud Object Gateway console.

   - To add a set of users to the **cluster-admins** group :

     ```
     # oc adm groups add-users cluster-admins <user-name> <user-name> <user-name>...
     ```

     where **<user-name>** is the name of the user to be added.

     > **NOTE**
     >
     > If you are adding a set of users to the **cluster-admins** group, you do not need to bind the newly added users to the cluster-admin role to allow access to the OpenShift Container Storage dashboard.

   - To remove a set of users from the **cluster-admins** group :

     ```
     # oc adm groups remove-users cluster-admins <user-name> <user-name> <user-name>...
     ```

     where **<user-name>** is the name of the user to be removed.

Verification steps

1. On the OpenShift Web Console, login as a user with access permission to Multicloud Object Gateway Console.

2. Navigate to **Home → Overview → Object Service** tab → select the **Multicloud Object Gateway** link .

3. On the Multicloud Object Gateway Console, login as the same user with access permission.

4. Click **Allow selected permissions**.

CHAPTER 4. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD

4.1. CREATING A NEW BACKING STORE

Use this procedure to create a new backing store in OpenShift Container Storage.

Prerequisites

- Administrator access to OpenShift.

Procedure

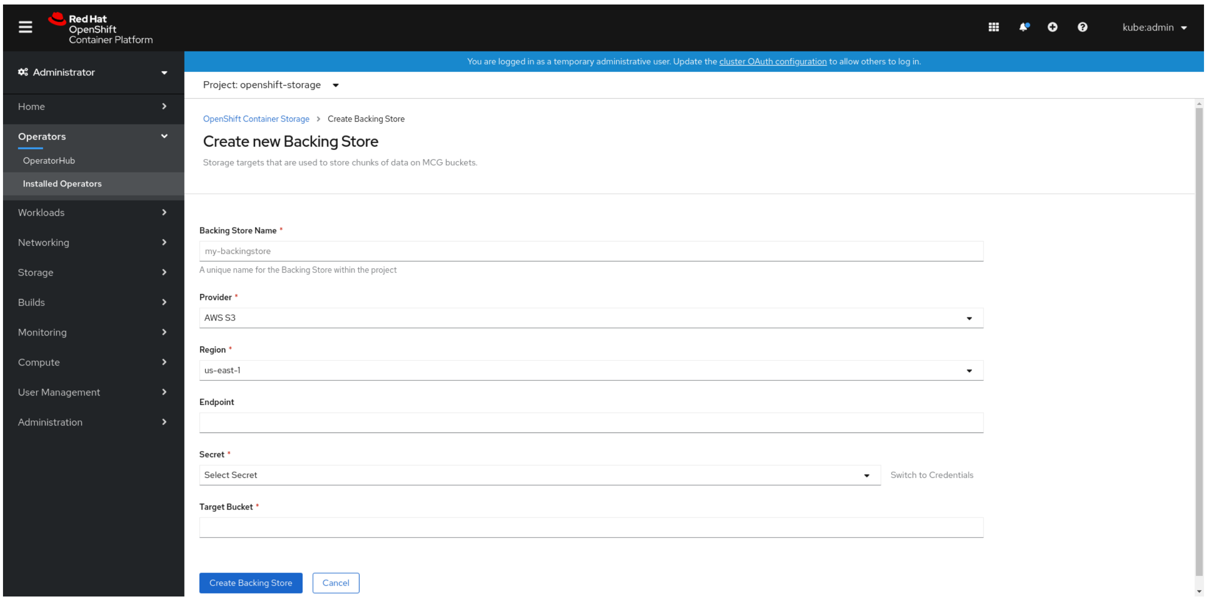
1. Click **Operators → Installed Operators** from the left pane of the OpenShift Web Console to view the installed operators.

2. Click **OpenShift Container Storage** Operator.

3. On the OpenShift Container Storage Operator page, scroll right and click the **Backing Store** tab.

4. Click **Create Backing Store**.

   Figure 4.1. Create Backing Store page



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Project: openshift-storage

OpenShift Container Storage > Create Backing Store

Create new Backing Store

Storage targets that are used to store chunks of data on MCG buckets.

Backing Store Name *
my-backingstore
A unique name for the Backing Store within the project

Provider *
AWS S3

Region *
us-east-1

Endpoint

Secret *
Select Secret

Target Bucket *

5. On the Create New Backing Store page, perform the following:

   a. Enter a **Backing Store Name**.

   b. Select a **Provider**.

   c. Select a **Region**.

   d. Enter an **Endpoint**. This is optional.

   e. Select a **Secret** from drop down list, or create your own secret. Optionally, you can **Switch to Credentials** view which lets you fill in the required secrets.

For more information on creating an OCP secret, see the section Creating the secret in the Openshift Container Platform documentation.

Each backingstore requires a different secret. For more information on creating the secret for a particular backingstore, see the Section 4.2, "Adding storage resources for hybrid or Multicloud using the MCG command line interface" and follow the procedure for the addition of storage resources using a YAML.

> NOTE
>
> This menu is relevant for all providers except Google Cloud and local PVC.

f. Enter **Target bucket**. The target bucket is a container storage that is hosted on the remote cloud service. It allows you to create a connection that tells MCG that it can use this bucket for the system.

6. Click **Create Backing Store**.

Verification steps

1. Click **Operators → Installed Operators**.
2. Click **OpenShift Container Storage** Operator.
3. Search for the new backing store or click **Backing Store** tab to view all the backing stores.

4.2. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD USING THE MCG COMMAND LINE INTERFACE

The Multicloud Object Gateway (MCG) simplifies the process of spanning data across cloud provider and clusters.

You must add a backing storage that can be used by the MCG.

Depending on the type of your deployment, you can choose one of the following procedures to create a backing storage:

- For creating an AWS-backed backingstore, see Section 4.2.1, "Creating an AWS-backed backingstore"
- For creating an IBM COS-backed backingstore, see Section 4.2.2, "Creating an IBM COS-backed backingstore"
- For creating an Azure-backed backingstore, see Section 4.2.3, "Creating an Azure-backed backingstore"
- For creating a GCP-backed backingstore, see Section 4.2.4, "Creating a GCP-backed backingstore"
- For creating a local Persistent Volume-backed backingstore, see Section 4.2.5, "Creating a local Persistent Volume-backed backingstore"

For VMware deployments, skip to Section 4.3, "Creating an s3 compatible Multicloud Object Gateway backingstore" for further instructions.

4.2.1. Creating an AWS-backed backingstore

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

  ```
  # subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

- Alternatively, you can install the **mcg** package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages

Procedure

1. From the MCG command-line interface, run the following command:

   ```
   noobaa backingstore create aws-s3 <backingstore_name> --access-key=<AWS ACCESS KEY> --secret-key=<AWS SECRET ACCESS KEY> --target-bucket <bucket-name> -n openshift-storage
   ```

   a. Replace **<backingstore_name>** with the name of the backingstore.

   b. Replace **<AWS ACCESS KEY>** and **<AWS SECRET ACCESS KEY>** with an AWS access key ID and secret access key you created for this purpose.

   c. Replace **<bucket-name>** with an existing AWS bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.
   The output will be similar to the following:

   ```
   INFO[0001]   Exists: NooBaa "noobaa"
   INFO[0002]   Created: BackingStore "aws-resource"
   INFO[0002]   Created: Secret "backing-store-secret-aws-resource"
   ```

You can also add storage resources using a YAML:

1. Create a secret with the credentials:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: <backingstore-secret-name>
     namespace: openshift-storage
   type: Opaque
   data:
     AWS_ACCESS_KEY_ID: <AWS ACCESS KEY ID ENCODED IN BASE64>
     AWS_SECRET_ACCESS_KEY: <AWS SECRET ACCESS KEY ENCODED IN BASE64>
   ```

   a. You must supply and encode your own AWS access key ID and secret access key using Base64, and use the results in place of **<AWS ACCESS KEY ID ENCODED IN BASE64>** and **<AWS SECRET ACCESS KEY ENCODED IN BASE64>**.

   b. Replace **<backingstore-secret-name>** with a unique name.

2. Apply the following YAML for a specific backing store:

   ```
   apiVersion: noobaa.io/v1alpha1
   kind: BackingStore
   metadata:
     finalizers:
     - noobaa.io/finalizer
     labels:
       app: noobaa
     name: bs
     namespace: openshift-storage
   spec:
     awsS3:
       secret:
         name: <backingstore-secret-name>
         namespace: openshift-storage
       targetBucket: <bucket-name>
     type: aws-s3
   ```

   a. Replace **<bucket-name>** with an existing AWS bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

   b. Replace **<backingstore-secret-name>** with the name of the secret created in the previous step.

4.2.2. Creating an IBM COS-backed backingstore

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

  ```
  # subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

- Alternatively, you can install the **mcg** package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages

Procedure

1. From the MCG command-line interface, run the following command:

   ```
   noobaa backingstore create ibm-cos <backingstore_name> --access-key=<IBM ACCESS
   KEY> --secret-key=<IBM SECRET ACCESS KEY> --endpoint=<IBM COS ENDPOINT> --
   target-bucket <bucket-name> -n openshift-storage
   ```

   a. Replace **<backingstore_name>** with the name of the backingstore.

   b. Replace **<IBM ACCESS KEY>**, **<IBM SECRET ACCESS KEY>**, **<IBM COS ENDPOINT>** with an IBM access key ID, secret access key and the appropriate regional endpoint that corresponds to the location of the existing IBM bucket.
   To generate the above keys on IBM cloud, you must include HMAC credentials while creating the service credentials for your target bucket.

c. Replace **<bucket-name>** with an existing IBM bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.
   The output will be similar to the following:

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "ibm-resource"
INFO[0002]   Created: Secret "backing-store-secret-ibm-resource"
```

You can also add storage resources using a YAML:

1. Create a secret with the credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
type: Opaque
data:
  IBM_COS_ACCESS_KEY_ID: <IBM COS ACCESS KEY ID ENCODED IN BASE64>
  IBM_COS_SECRET_ACCESS_KEY: <IBM COS SECRET ACCESS KEY ENCODED IN
BASE64>
```

   a. You must supply and encode your own IBM COS access key ID and secret access key using Base64, and use the results in place of **<IBM COS ACCESS KEY ID ENCODED IN BASE64>** and **<IBM COS SECRET ACCESS KEY ENCODED IN BASE64>**.

   b. Replace **<backingstore-secret-name>** with a unique name.

2. Apply the following YAML for a specific backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  ibmCos:
    endpoint: <endpoint>
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBucket: <bucket-name>
  type: ibm-cos
```

   a. Replace **<bucket-name>** with an existing IBM COS bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

b. Replace **<endpoint>** with a regional endpoint that corresponds to the location of the existing IBM bucket name. This argument tells Multicloud Object Gateway which endpoint to use for its backing store, and subsequently, data storage and administration.

c. Replace **<backingstore-secret-name>** with the name of the secret created in the previous step.

4.2.3. Creating an Azure-backed backingstore

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

- Alternatively, you can install the **mcg** package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages

Procedure

1. From the MCG command-line interface, run the following command:

```
noobaa backingstore create azure-blob <backingstore_name> --account-key=<AZURE
ACCOUNT KEY> --account-name=<AZURE ACCOUNT NAME> --target-blob-container
<blob container name>
```

a. Replace **<backingstore_name>** with the name of the backingstore.

b. Replace **<AZURE ACCOUNT KEY>** and **<AZURE ACCOUNT NAME>** with an AZURE account key and account name you created for this purpose.

c. Replace **<blob container name>** with an existing Azure blob container name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.
The output will be similar to the following:

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "azure-resource"
INFO[0002]   Created: Secret "backing-store-secret-azure-resource"
```

You can also add storage resources using a YAML:

1. Create a secret with the credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
type: Opaque
data:
  AccountName: <AZURE ACCOUNT NAME ENCODED IN BASE64>
  AccountKey: <AZURE ACCOUNT KEY ENCODED IN BASE64>
```

   a. You must supply and encode your own Azure Account Name and Account Key using Base64, and use the results in place of **<AZURE ACCOUNT NAME ENCODED IN BASE64>** and **<AZURE ACCOUNT KEY ENCODED IN BASE64>**.

   b. Replace **<backingstore-secret-name>** with a unique name.

2. Apply the following YAML for a specific backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  azureBlob:
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBlobContainer: <blob-container-name>
  type: azure-blob
```

   a. Replace **<blob-container-name>** with an existing Azure blob container name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

   b. Replace **<backingstore-secret-name>** with the name of the secret created in the previous step.

4.2.4. Creating a GCP-backed backingstore

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

- Alternatively, you can install the **mcg** package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages

Procedure

1. From the MCG command-line interface, run the following command:

```
noobaa backingstore create google-cloud-storage <backingstore_name> --private-key-json-file=<PATH TO GCP PRIVATE KEY JSON FILE> --target-bucket <GCP bucket name>
```

   a. Replace **<backingstore_name>** with the name of the backingstore.

b. Replace **<PATH TO GCP PRIVATE KEY JSON FILE>** with a path to your GCP private key created for this purpose.

c. Replace **<GCP bucket name>** with an existing GCP object storage bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.
The output will be similar to the following:

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "google-gcp"
INFO[0002]   Created: Secret "backing-store-google-cloud-storage-gcp"
```

You can also add storage resources using a YAML:

1. Create a secret with the credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
type: Opaque
data:
  GoogleServiceAccountPrivateKeyJson: <GCP PRIVATE KEY ENCODED IN BASE64>
```

a. You must supply and encode your own GCP service account private key using Base64, and use the results in place of **<GCP PRIVATE KEY ENCODED IN BASE64>**.

b. Replace <backingstore-secret-name> with a unique name.

2. Apply the following YAML for a specific backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  googleCloudStorage:
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBucket: <target bucket>
  type: google-cloud-storage
```

a. Replace **<target bucket>** with an existing Google storage bucket. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

b. Replace **<backingstore-secret-name>** with the name of the secret created in the previous step.

4.2.5. Creating a local Persistent Volume-backed backingstore

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

- Alternatively, you can install the **mcg** package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages

Procedure

1. From the MCG command-line interface, run the following command:

```
noobaa backingstore create  pv-pool <backingstore_name> --num-volumes=<NUMBER OF
VOLUMES>  --pv-size-gb=<VOLUME SIZE> --storage-class=<LOCAL STORAGE CLASS>
```

   a. Replace **<backingstore_name>** with the name of the backingstore.

   b. Replace **<NUMBER OF VOLUMES>** with the number of volumes you would like to create. Note that increasing the number of volumes scales up the storage.

   c. Replace **<VOLUME SIZE>** with the required size, in GB, of each volume

   d. Replace **<LOCAL STORAGE CLASS>** with the local storage class, recommended to use ocs-storagecluster-ceph-rbd
   The output will be similar to the following:

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Exists: BackingStore "local-mcg-storage"
```

You can also add storage resources using a YAML:

1. Apply the following YAML for a specific backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <backingstore_name>
  namespace: openshift-storage
spec:
   pvPool:
    numVolumes: <NUMBER OF VOLUMES>
    resources:
      requests:
        storage: <VOLUME SIZE>
    storageClass: <LOCAL STORAGE CLASS>
  type: pv-pool
```

a. Replace **<backingstore_name>** with the name of the backingstore.

b. Replace **<NUMBER OF VOLUMES>** with the number of volumes you would like to create. Note that increasing the number of volumes scales up the storage.

c. Replace **<VOLUME SIZE>** with the required size, in GB, of each volume. Note that the letter G should remain.

d. Replace **<LOCAL STORAGE CLASS>** with the local storage class, recommended to use ocs-storagecluster-ceph-rbd

4.3. CREATING AN S3 COMPATIBLE MULTICLOUD OBJECT GATEWAY BACKINGSTORE

The Multicloud Object Gateway can use any S3 compatible object storage as a backing store, for example, Red Hat Ceph Storage's RADOS Gateway (RGW). The following procedure shows how to create an S3 compatible Multicloud Object Gateway backing store for Red Hat Ceph Storage's RADOS Gateway. Note that when RGW is deployed, Openshift Container Storage operator creates an S3 compatible backingstore for Multicloud Object Gateway automatically.

Procedure

1. From the Multicloud Object Gateway (MCG) command-line interface, run the following NooBaa command:

```
noobaa backingstore create s3-compatible rgw-resource --access-key=<RGW ACCESS
KEY> --secret-key=<RGW SECRET KEY> --target-bucket=<bucket-name> --endpoint=
<RGW endpoint>
```

a. To get the **<RGW ACCESS KEY>** and **<RGW SECRET KEY>**, run the following command using your RGW user secret name:

```
oc get secret <RGW USER SECRET NAME> -o yaml -n openshift-storage
```

b. Decode the access key ID and the access key from Base64 and keep them.

c. Replace **<RGW USER ACCESS KEY>** and **<RGW USER SECRET ACCESS KEY>** with the appropriate, decoded data from the previous step.

d. Replace **<bucket-name>** with an existing RGW bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

e. To get the **<RGW endpoint>**, see Accessing the RADOS Object Gateway S3 endpoint . The output will be similar to the following:

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "rgw-resource"
INFO[0002]   Created: Secret "backing-store-secret-rgw-resource"
```

You can also create the backingstore using a YAML:

1. Create a **CephObjectStore** user. This also creates a secret containing the RGW credentials:

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: <RGW-Username>
  namespace: openshift-storage
spec:
  store: ocs-storagecluster-cephobjectstore
  displayName: "<Display-name>"
```

   a. Replace **<RGW-Username>** and **<Display-name>** with a unique username and display name.

2. Apply the following YAML for an S3-Compatible backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <backingstore-name>
  namespace: openshift-storage
spec:
  s3Compatible:
    endpoint: <RGW endpoint>
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    signatureVersion: v4
    targetBucket: <RGW-bucket-name>
  type: s3-compatible
```

   a. Replace **<backingstore-secret-name>** with the name of the secret that was created with **CephObjectStore** in the previous step.

   b. Replace **<bucket-name>** with an existing RGW bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

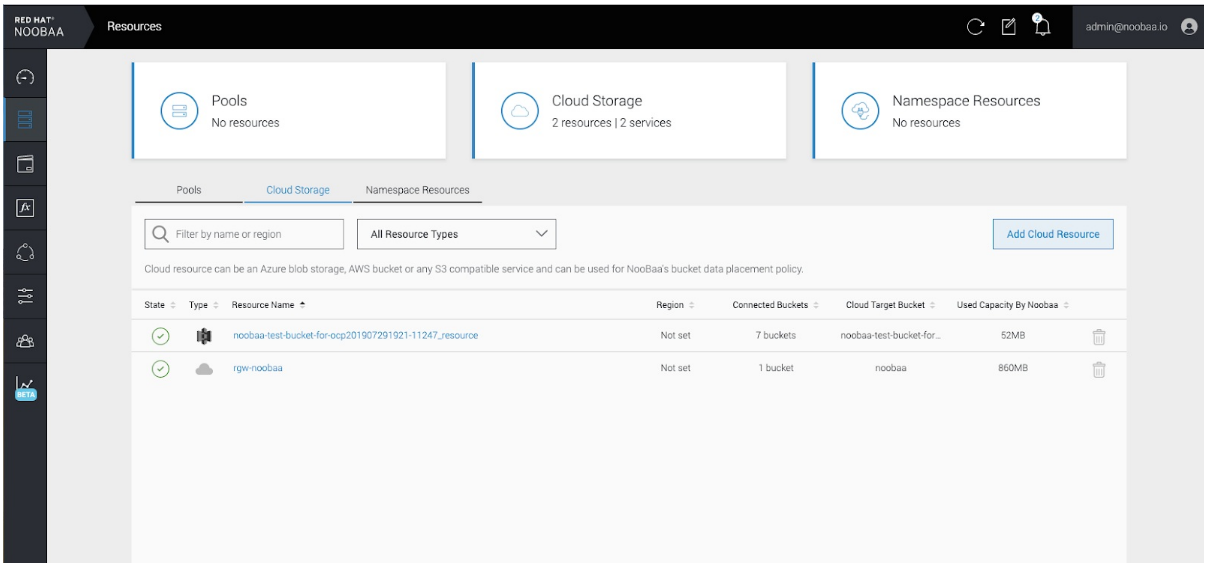
   c. To get the **<RGW endpoint>**, see Accessing the RADOS Object Gateway S3 endpoint .

4.4. ADDING STORAGE RESOURCES FOR HYBRID AND MULTICLOUD USING THE USER INTERFACE

Procedure

1. In your OpenShift Storage console, click **Overview → Object Service → Multicloud Object Gateway** link.

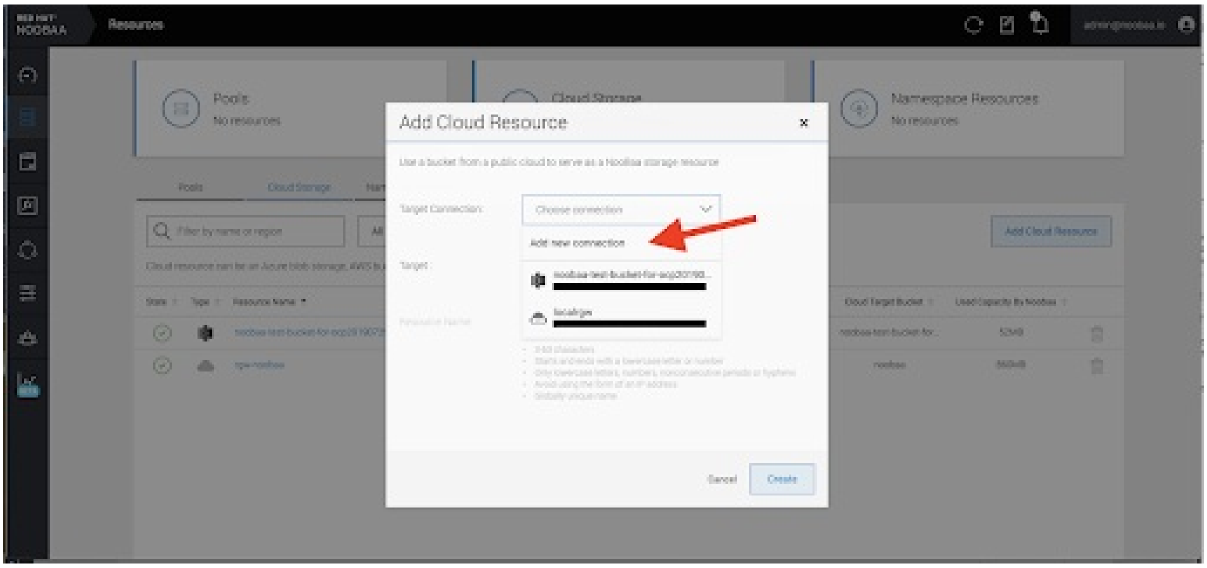
2. Select the **Resources** tab in the left, highlighted below. From the list that populates, select **Add Cloud Resource**.



Cloud resource can be an Azure blob storage, AWS bucket or any S3 compatible service and can be used for NooBaa's bucket data placement policy.

State	Type	Resource Name	Region	Connected Buckets	Cloud Target Bucket	Used Capacity By Noobaa
✓		noobaa-test-bucket-for-ocp201907291921-11247_resource	Not set	7 buckets	noobaa-test-bucket-for...	52MB
✓		rgw-noobaa	Not set	1 bucket	noobaa	860MB

3. Select **Add new connection**.



Use a bucket from a public cloud to serve as a NooBaa storage resource

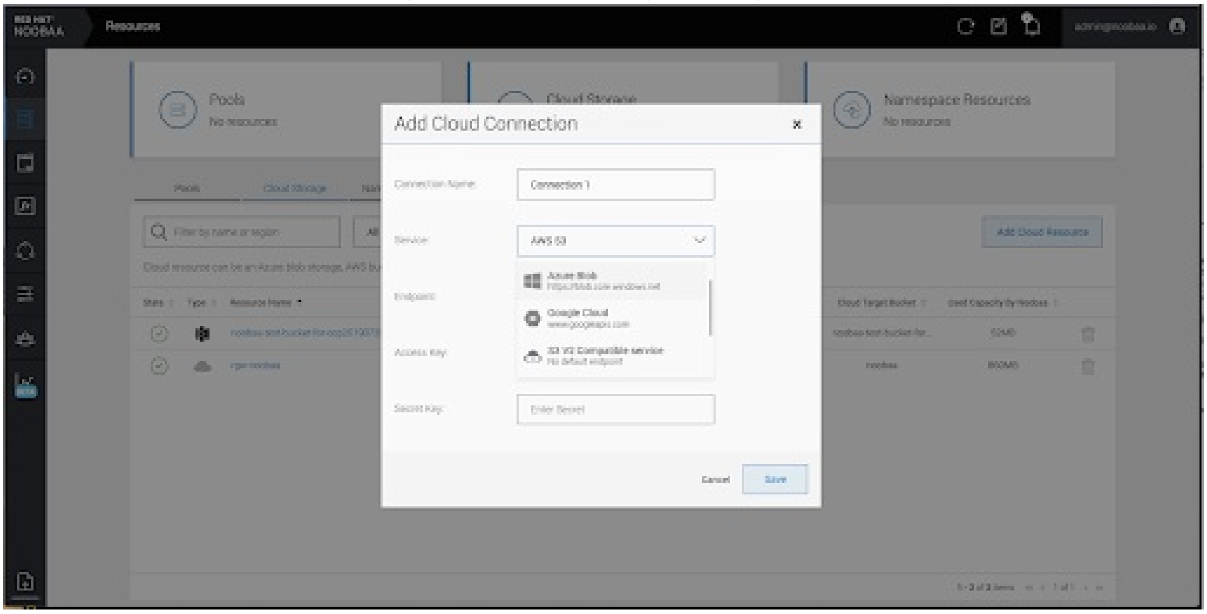
Target Connection:

Target:

noobaa-test-bucket-for-ocp201907...
[redacted]

localrgw
[redacted]

4. Select the relevant native cloud provider or S3 compatible option and fill in the details.



Connection Name: Connection 1

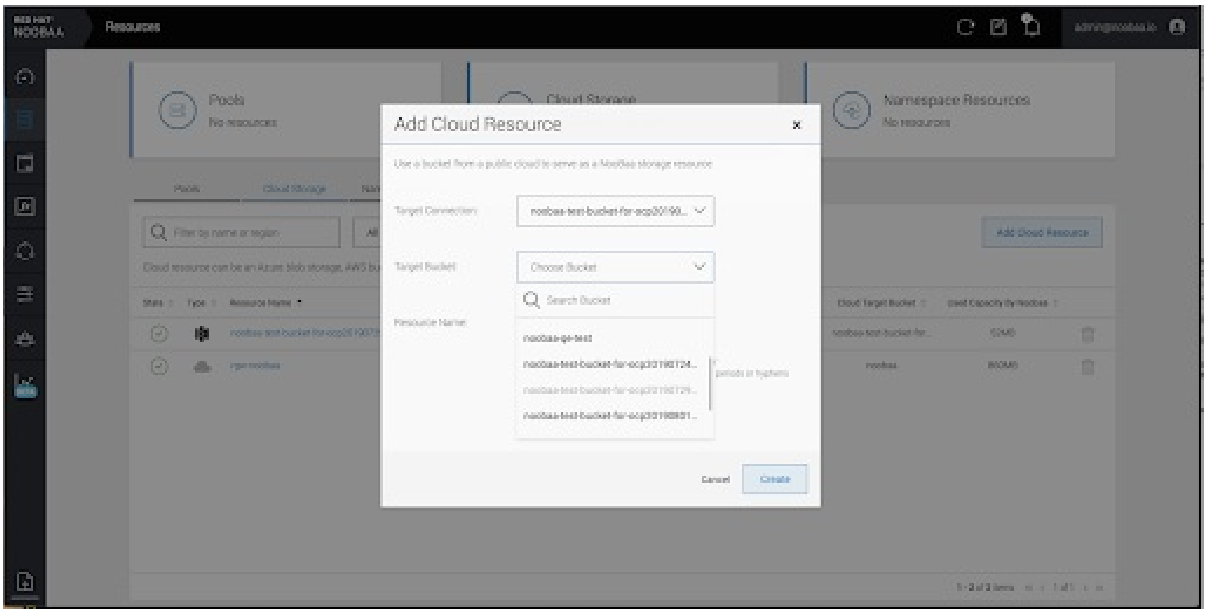
Service: AWS S3

Endpoint:

Access Key:

Secret Key: Enter Secret

5. Select the newly created connection and map it to the existing bucket.



Add Cloud Resource

Use a bucket from a public cloud to serve as a NooBaa storage resource

Target Connection: noobaa-test-bucket-for-ocp20190...

Target Bucket: Choose Bucket

Search Bucket

noobaa-qe-test

noobaa-test-bucket-for-ocp20190724...

noobaa-test-bucket-for-ocp20190729...

noobaa-test-bucket-for-ocp20190801...

Resource Name:

6. Repeat these steps to create as many backing stores as needed.

NOTE

Resources created in NooBaa UI cannot be used by OpenShift UI or MCG CLI.

4.5. CREATING A NEW BUCKET CLASS

Bucket class is a CRD representing a class of buckets that defines tiering policies and data placements for an Object Bucket Class (OBC).

Use this procedure to create a bucket class in OpenShift Container Storage.

Procedure

1. Click **Operators → Installed Operators** from the left pane of the OpenShift Web Console to view the installed operators.

2. Click **OpenShift Container Storage** Operator.

3. On the OpenShift Container Storage Operator page, scroll right and click the **Bucket Class** tab.

4. Click **Create Bucket Class**.

5. On the Create new Bucket Class page, perform the following:

    a. Enter a **Bucket Class Name** and click **Next**.

    b. In Placement Policy, select **Tier 1 - Policy Type**and click **Next**. You can choose either one of the options as per your requirements.

    - **Spread** allows spreading of the data across the chosen resources.

    - **Mirror** allows full duplication of the data across the chosen resources.

    - Click **Add Tier** to add another policy tier.

c. Select at least one **Backing Store** resource from the available list if you have selected Tier 1 - Policy Type as Spread and click **Next**. Alternatively, you can also create a new backing store.

> **NOTE**
>
> You need to select atleast 2 backing stores when you select Policy Type as Mirror in previous step.

d. Review and confirm Bucket Class settings.

e. Click **Create Bucket Class**.

Verification steps

1. Click **Operators → Installed Operators**.
2. Click **OpenShift Container Storage** Operator.
3. Search for the new Bucket Class or click **Bucket Class** tab to view all the Bucket Classes.

4.6. EDITING A BUCKET CLASS

Use the following procedure to edit the bucket class components through the YAML file by clicking the **edit** button on the Openshift web console.

Prerequisites

- Administrator access to OpenShift.

Procedure

1. Log into the **OpenShift Web Console**.
2. Click **Operators → Installed Operators**.
3. Click **OpenShift Container Storage Operator**.
4. On the OpenShift Container Storage Operator page, scroll right and click the **Bucket Class** tab.
5. Click on the action menu (⋮) next to the Bucket class you want to edit.
6. Click **Edit Bucket Class**.
7. You are redirected to the **YAML** file, make the required changes in this file and click **Save**.

4.7. EDITING BACKING STORES FOR BUCKET CLASS

Use the following procedure to edit an existing Multicloud Object Gateway bucket class to change the underlying backing stores used in a bucket class.

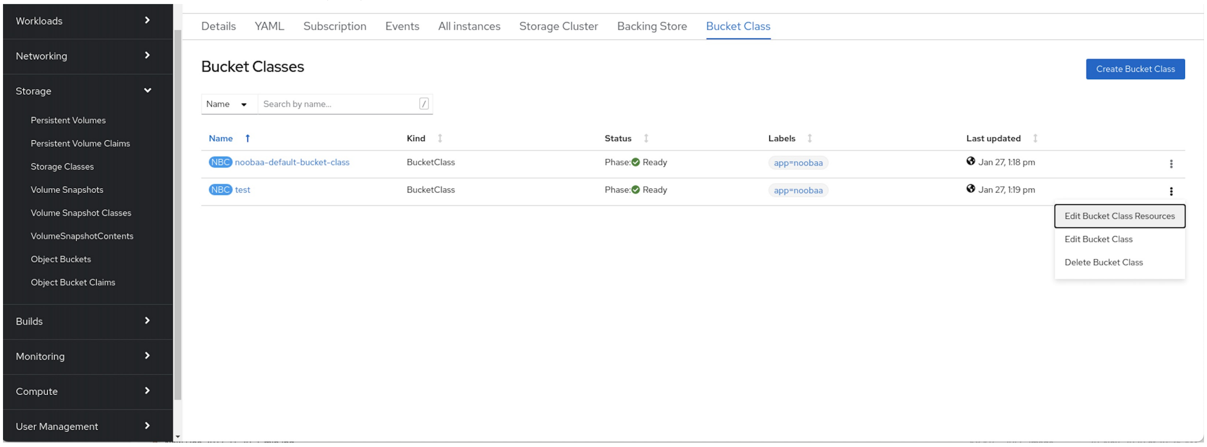
Prerequisites

- Administrator access to OpenShift Web Console.

- A bucket class.
- Backing stores.

Procedure

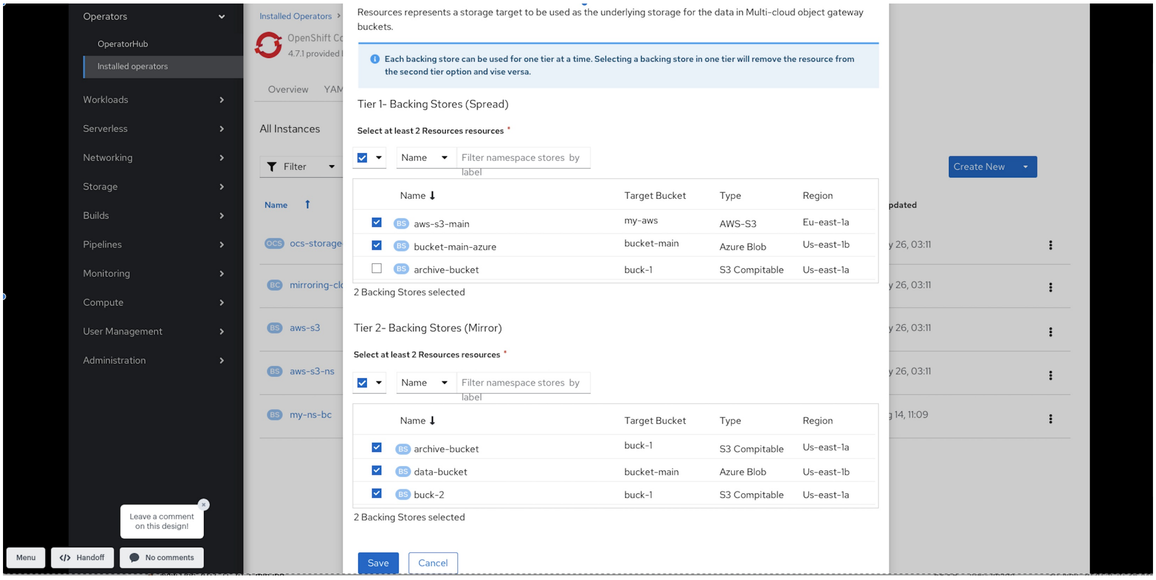
1. Click **Operators** → Installed Operators to view the installed operators.
2. Click **OpenShift Container Storage Operator**.
3. Click the **Bucket Class** tab.
4. Click on the action menu (⋮) next to the Bucket class you want to edit.



Bucket Classes

Name	Kind	Status	Labels	Last updated
NBC noobaa-default-bucket-class	BucketClass	Phase: Ready	app=noobaa	Jan 27, 1:18 pm
NBC test	BucketClass	Phase: Ready	app=noobaa	Jan 27, 1:19 pm

5. Click **Edit Bucket Class Resources**.
6. On the **Edit Bucket Class Resources** page, edit the bucket class resources either by adding a backing store to the bucket class or by removing a backing store from the bucket class. You can also edit bucket class resources created with one or two tiers and different placement policies.
    - To add a backing store to the bucket class, select the name of the backing store.
    - To remove a backing store from the bucket class, clear the name of the backing store.



Resources represents a storage target to be used as the underlying storage for the data in Multi-cloud object gateway buckets.

Each backing store can be used for one tier at a time. Selecting a backing store in one tier will remove the resource from the second tier option and vise versa.

Tier 1- Backing Stores (Spread)

Select at least 2 Resources resources *

	Name	Target Bucket	Type	Region
☑	BS aws-s3-main	my-aws	AWS-S3	Eu-east-1a
☑	BS bucket-main-azure	bucket-main	Azure Blob	Us-east-1b
□	BS archive-bucket	buck-1	S3 Compitable	Us-east-1a

2 Backing Stores selected

Tier 2- Backing Stores (Mirror)

Select at least 2 Resources resources *

	Name	Target Bucket	Type	Region
☑	BS archive-bucket	buck-1	S3 Compitable	Us-east-1a
☑	BS data-bucket	bucket-main	Azure Blob	Us-east-1b
☑	BS buck-2	buck-1	S3 Compitable	Us-east-1a

2 Backing Stores selected

7. Click **Save**.

CHAPTER 5. MANAGING NAMESPACE BUCKETS

Namespace buckets let you connect data repositories on different providers together, so you can interact with all of your data through a single unified view. Add the object bucket associated with each provider to the namespace bucket, and access your data through the namespace bucket to see all of your object buckets at once. This lets you write to your preferred storage provider while reading from multiple other storage providers, greatly reducing the cost of migrating to a new storage provider.

You can interact with objects in a namespace bucket using the S3 API. See S3 API endpoints for objects in namespace buckets for more information.

NOTE

A namespace bucket can only be used if its write target is available and functional.

5.1. AMAZON S3 API ENDPOINTS FOR OBJECTS IN NAMESPACE BUCKETS

You can interact with objects in namespace buckets using the Amazon Simple Storage Service (S3) API.

Red Hat OpenShift Container Storage 4.6 onwards supports the following namespace bucket operations:

- ListObjectVersions
- ListObjects
- PutObject
- CopyObject
- ListParts
- CreateMultipartUpload
- CompleteMultipartUpload
- UploadPart
- UploadPartCopy
- AbortMultipartUpload
- GetObjectAcl
- GetObject
- HeadObject
- DeleteObject
- DeleteObjects

See the Amazon S3 API reference documentation for the most up-to-date information about these operations and how to use them.

Additional resources

- Amazon S3 REST API Reference
- Amazon S3 CLI Reference

5.2. ADDING A NAMESPACE BUCKET USING THE MULTICLOUD OBJECT GATEWAY CLI AND YAML

For more information about namespace buckets, see Managing namespace buckets.

Depending on the type of your deployment and whether you want to use YAML or the Multicloud Object Gateway CLI, choose one of the following procedures to add a namespace bucket:

- Adding an AWS S3 namespace bucket using YAML
- Adding an IBM COS namespace bucket using YAML
- Adding an AWS S3 namespace bucket using the Multicloud Object Gateway CLI
- Adding an IBM COS namespace bucket using the Multicloud Object Gateway CLI

5.2.1. Adding an AWS S3 namespace bucket using YAML

Prerequisites

- A running OpenShift Container Storage Platform
- Access to the Multicloud Object Gateway, see Chapter 2, Accessing the Multicloud Object Gateway with your applications

Procedure

1. Create a secret with the credentials:

```
apiVersion: v1
kind: Secret
metadata:
name: <namespacestore-secret-name>
type: Opaque
data:
AWS_ACCESS_KEY_ID: <AWS ACCESS KEY ID ENCODED IN BASE64>
AWS_SECRET_ACCESS_KEY: <AWS SECRET ACCESS KEY ENCODED IN BASE64>
```

   a. You must supply and encode your own AWS access key ID and secret access key using Base64, and use the results in place of **<AWS ACCESS KEY ID ENCODED IN BASE64>** and **<AWS SECRET ACCESS KEY ENCODED IN BASE64>**. ii. Replace **<namespacestore-secret-name>** with a unique name.

2. Create a NamespaceStore resource using OpenShift Custom Resource Definitions (CRDs). A NamespaceStore represents underlying storage to be used as a read or write target for the data in the Multicloud Object Gateway namespace buckets. To create a NamespaceStore resource, apply the following YAML:

```
apiVersion: noobaa.io/v1alpha1
```

```
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: mybucketnamespace
  namespace: k8snamespace
spec:
  awsS3:
    secret:
      name: <namespacestore-secret-name>
      namespace: k8snamespace
    targetBucket: awsdatalake
  type: aws-s3
```

a. Replace **<namespacestore-secret-name>** with with the secret created in step 1.

3. Create a namespace bucket class that defines a namespace policy for the namespace buckets. The namespace policy requires a type of either **single** or **multi**.

- A namespace policy of type **single** requires the following configuration:

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
spec:
  namespacePolicy:
    type:
    single:
      resource: <resource>
```

Replace **<my-bucket-class>** with a unique namespace bucket class name.

Replace **<resource>** with a single namespace-store that will define the read and write target of the namespace bucket.

- A namespace policy of type **multi** requires the following configuration:

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
spec:
  namespacePolicy:
    type: Multi
    multi:
      writeResource: <write-resource>
```

```
readResources:
- <read-resources>
- <read-resources>
```

Replace **<my-bucket-class>** with a unique bucket class name.

Replace **write-resource** with a single namespace-store that will define the write target of the namespace bucket.

Replace **<read-resources** with a list of namespace-stores that will define the read targets of the namespace bucket.

4. Apply the following YAML to create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in step 2.

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: my-bucket-claim
  namespace: my-app
spec:
  generateBucketName: my-bucket
  storageClassName: noobaa.noobaa.io
  additionalConfig:
    bucketclass: <my-bucket-class>
```

   a. Replace **<my-bucket-class>** with the bucket class created in the previous step.

Once the OBC is provisioned by the operator, a bucket is created in the Multicloud Object Gateway, and the operator creates a Secret and ConfigMap with the same name of the OBC on the same namespace of the OBC.

5.2.2. Adding an IBM COS namespace bucket using YAML

Prerequisites

- A running OpenShift Container Storage Platform
- Access to the Multicloud Object Gateway, see Chapter 2, Accessing the Multicloud Object Gateway with your applications

Procedure

1. Create a secret with the credentials:

```
apiVersion: v1
kind: Secret
metadata:
name: <namespacestore-secret-name>
type: Opaque
data:
IBM_COS_ACCESS_KEY_ID: <IBM COS ACCESS KEY ID ENCODED IN BASE64>
IBM_COS_SECRET_ACCESS_KEY: <IBM COS SECRET ACCESS KEY ENCODED IN
BASE64>
```

   a. You must supply and encode your own IBM COS access key ID and secret access key using Base64, and use the results in place of **<IBM COS ACCESS KEY ID ENCODED IN BASE64>** and `<IBM COS SECRET ACCESS KEY ENCODED IN BASE64>.

   b. Replace **<namespacestore-secret-name>** with a unique name.

2. Create a NamespaceStore resource using OpenShift Custom Resource Definitions (CRDs). A NamespaceStore represents underlying storage to be used as a read or write target for the data in the Multicloud Object Gateway namespace buckets. To create a NamespaceStore resource, apply the following YAML:

```
apiVersion: noobaa.io/v1alpha1
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: k8snamespace
spec:
  s3Compatible:
    endpoint: <IBM COS ENDPOINT>
    secret:
      name: <namespacestore-secret-name>
      namespace: openshift-storage
    signatureVersion: v2
    targetBucket: BUCKET
  type: ibm-cos
```

   a. Replace **<IBM COS ENDPOINT>** with the appropriate IBM COS endpoint.

   b. Replace **<namespacestore-secret-name>** with the secret created in step 1.

3. Create a namespace bucket class that defines a namespace policy for the namespace buckets. The namespace policy requires a type of either **single** or **multi**.

   - A namespace policy of type **single** requires the following configuration:

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
spec:
  namespacePolicy:
    type:
    single:
      resource: <resource>
```

     Replace **<my-bucket-class>** with a unique namespace bucket class name.

     Replace **<resource>** with a single namespace-store that will define the read and write target of the namespace bucket.

- A namespace policy of type **multi** requires the following configuration:

  ```
  apiVersion: noobaa.io/v1alpha1
  kind: BucketClass
  metadata:
    labels:
      app: noobaa
    name: <my-bucket-class>
    namespace: openshift-storage
  spec:
    namespacePolicy:
      type: Multi
      multi:
        writeResource: <write-resource>
        readResources:
        - <read-resources>
        - <read-resources>
  ```

  Replace **<my-bucket-class>** with a unique bucket class name.

  Replace **write-resource** with a single namespace-store that will define the write target of the namespace bucket.

  Replace **<read-resources** with a list of namespace-stores that will define the read targets of the namespace bucket.

4. Apply the following YAML to create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in step 2.

   ```
   apiVersion: objectbucket.io/v1alpha1
   kind: ObjectBucketClaim
   metadata:
     name: my-bucket-claim
     namespace: my-app
   spec:
     generateBucketName: my-bucket
     storageClassName: noobaa.noobaa.io
     additionalConfig:
       bucketclass: <my-bucket-class>
   ```

   a. Replace **<my-bucket-class>** with the bucket class created in the previous step.

Once the OBC is provisioned by the operator, a bucket is created in the Multicloud Object Gateway, and the operator creates a Secret and ConfigMap with the same name of the OBC on the same namespace of the OBC.

5.2.3. Adding an AWS S3 namespace bucket using the Multicloud Object Gateway CLI

Prerequisites

- A running OpenShift Container Storage Platform

- Access to the Multicloud Object Gateway, see Chapter 2, Accessing the Multicloud Object Gateway with your applications

- Download the Multicloud Object Gateway command-line interface:

  ```
  # subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

  Alternatively, you can install the mcg package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package.

Procedure

1. Create a NamespaceStore resource. A NamespaceStore represents an underlying storage to be used as a read or write target for the data in Multicloud Object Gateway namespace buckets. From the MCG command-line interface, run the following command:

   ```
   noobaa namespacestore create aws-s3 <namespacestore> --access-key <AWS ACCESS KEY> --secret-key <AWS SECRET ACCESS KEY> --target-bucket <bucket-name> -n openshift-storage
   ```

   a. Replace **<namespacestore>** with the name of the NamespaceStore.

   b. Replace **<AWS ACCESS KEY>** and **<AWS SECRET ACCESS KEY>** with an AWS access key ID and secret access key you created for this purpose.

   c. Replace **<bucket-name>** with an existing AWS bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

2. Create a namespace bucket class that defines a namespace policy for the namespace buckets. The namespace policy requires a type of either **single** or **multi**.

   - Run the following command to create a namespace bucket class with a namespace policy of type **single**:

     ```
     noobaa bucketclass create namespace-bucketclass single <my-bucket-class> --resource <resource> -n openshift-storage
     ```

     Replace **<my-bucket-class>** with a unique bucket class name.

     Replace **<resource>** with a single namespace-store that will define the read and write target of the namespace bucket.

   - Run the following command to create a namespace bucket class with a namespace policy of type **multi**:

     ```
     noobaa bucketclass create namespace-bucketclass multi <my-bucket-class> --write-resource <write-resource> --read-resources <read-resources> -n openshift-storage
     ```

     Replace **<my-bucket-class>** with a unique bucket class name.

     Replace **write-resource** with a single namespace-store that will define the write target of the namespace bucket.

     Replace **<read-resources** with a list of namespace-stores separated by commas that will define the read targets of the namespace bucket.

3. Run the following command to create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in step 2.

   ```
   noobaa obc create my-bucket-claim -n openshift-storage --app-namespace my-app --
   bucketclass <custom-bucket-class>
   ```

   a. Replace **<custom-bucket-class>** with the name of the bucket class created in step 2.

Once the OBC is provisioned by the operator, a bucket is created in the Multicloud Object Gateway, and the operator creates a Secret and ConfigMap with the same name of the OBC on the same namespace of the OBC.

5.2.4. Adding an IBM COS namespace bucket using the Multicloud Object Gateway CLI

Prerequisites

- A running OpenShift Container Storage Platform
- Access to the Multicloud Object Gateway, see Chapter 2, Accessing the Multicloud Object Gateway with your applications
- Download the Multicloud Object Gateway command-line interface:

  ```
  # subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

  Alternatively, you can install the mcg package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package.

Procedure

1. Create a NamespaceStore resource. A NamespaceStore represents an underlying storage to be used as a read or write target for the data in Multicloud Object Gateway namespace buckets. From the MCG command-line interface, run the following command:

   ```
   noobaa namespacestore create ibm-cos <namespacestore> --endpoint <IBM COS
   ENDPOINT> --access-key <IBM ACCESS KEY> --secret-key <IBM SECRET ACCESS
   KEY> --target-bucket <bucket-name> -n openshift-storage
   ```

   a. Replace **<namespacestore>** with the name of the NamespaceStore.

   b. Replace **<IBM ACCESS KEY>**, **<IBM SECRET ACCESS KEY>**, **<IBM COS ENDPOINT>** with an IBM access key ID, secret access key and the appropriate regional endpoint that corresponds to the location of the existing IBM bucket.

   c. Replace **<bucket-name>** with an existing IBM bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.

2. Create a namespace bucket class that defines a namespace policy for the namespace buckets. The namespace policy requires a type of either **single** or **multi**.

- Run the following command to create a namespace bucket class with a namespace policy of type **single**:

  ```
  noobaa bucketclass create namespace-bucketclass single <my-bucket-class> --resource
  <resource> -n openshift-storage
  ```

  Replace **<my-bucket-class>** with a unique bucket class name.

  Replace **<resource>** with a single namespace-store that will define the read and write target of the namespace bucket.

- Run the following command to create a namespace bucket class with a namespace policy of type **multi**:

  ```
  noobaa bucketclass create namespace-bucketclass multi <my-bucket-class> --write-
  resource <write-resource> --read-resources <read-resources> -n openshift-storage
  ```

  Replace **<my-bucket-class>** with a unique bucket class name.

  Replace **write-resource** with a single namespace-store that will define the write target of the namespace bucket.

  Replace **<read-resources** with a list of namespace-stores separated by commas that will define the read targets of the namespace bucket.

3. Run the following command to create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in step 2.

   ```
   noobaa obc create my-bucket-claim -n openshift-storage --app-namespace my-app --
   bucketclass <custom-bucket-class>
   ```

   a. Replace **<custom-bucket-class>** with the name of the bucket class created in step 2.

Once the OBC is provisioned by the operator, a bucket is created in the Multicloud Object Gateway, and the operator creates a Secret and ConfigMap with the same name of the OBC on the same namespace of the OBC.

CHAPTER 6. MIRRORING DATA FOR HYBRID AND MULTICLOUD BUCKETS

The Multicloud Object Gateway (MCG) simplifies the process of spanning data across cloud provider and clusters.

Prerequisites

- You must first add a backing storage that can be used by the MCG, see Chapter 4, *Adding storage resources for hybrid or Multicloud*.

Then you create a bucket class that reflects the data management policy, mirroring.

Procedure

You can set up mirroring data three ways:

- Section 6.1, “Creating bucket classes to mirror data using the MCG command-line-interface”
- Section 6.2, “Creating bucket classes to mirror data using a YAML”
- Section 6.3, “Configuring buckets to mirror data using the user interface”

6.1. CREATING BUCKET CLASSES TO MIRROR DATA USING THE MCG COMMAND-LINE-INTERFACE

1. From the MCG command-line interface, run the following command to create a bucket class with a mirroring policy:

```
$ noobaa bucketclass create placement-bucketclass mirror-to-aws --backingstores=azure-resource,aws-resource --placement Mirror
```

2. Set the newly created bucket class to a new bucket claim, generating a new bucket that will be mirrored between two locations:

```
$ noobaa obc create  mirrored-bucket --bucketclass=mirror-to-aws
```

6.2. CREATING BUCKET CLASSES TO MIRROR DATA USING A YAML

1. Apply the following YAML.

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <bucket-class-name>
  namespace: openshift-storage
spec:
  placementPolicy:
    tiers:
    - backingStores:
        - <backing-store-1>
        - <backing-store-2>
        placement: Mirror
```

2. Add the following lines to your standard Object Bucket Claim (OBC):

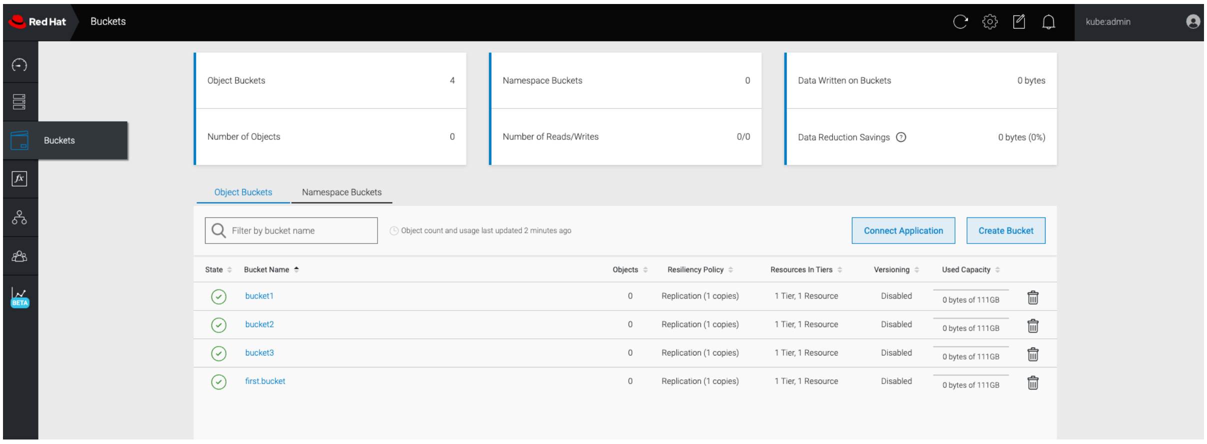
```
additionalConfig:
  bucketclass: mirror-to-aws
```

For more information about OBCs, see *Chapter 8, Object Bucket Claim*.

6.3. CONFIGURING BUCKETS TO MIRROR DATA USING THE USER INTERFACE

1. In your OpenShift Storage console, Click **Overview → Object Service → Multicloud Object Gateway** link.

2. On the NooBaa page, click the **buckets** icon on the left side. You will see a list of your buckets:



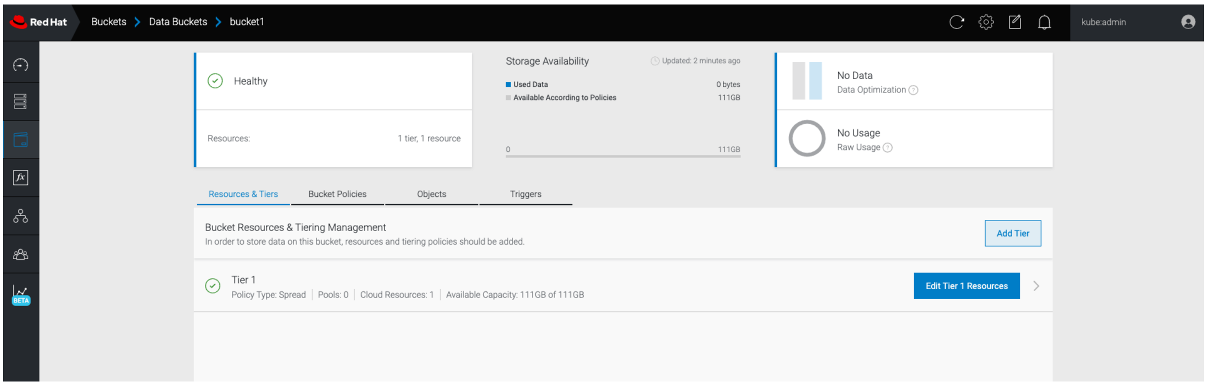
Object Buckets 4
Number of Objects 0
Namespace Buckets 0
Number of Reads/Writes 0/0
Data Written on Buckets 0 bytes
Data Reduction Savings 0 bytes (0%)

Object count and usage last updated 2 minutes ago

State	Bucket Name	Objects	Resiliency Policy	Resources In Tiers	Versioning	Used Capacity
✓	bucket1	0	Replication (1 copies)	1 Tier, 1 Resource	Disabled	0 bytes of 111GB
✓	bucket2	0	Replication (1 copies)	1 Tier, 1 Resource	Disabled	0 bytes of 111GB
✓	bucket3	0	Replication (1 copies)	1 Tier, 1 Resource	Disabled	0 bytes of 111GB
✓	first.bucket	0	Replication (1 copies)	1 Tier, 1 Resource	Disabled	0 bytes of 111GB

3. Click the bucket you want to update.

4. Click **Edit Tier 1 Resources**:



Healthy
Resources: 1 tier, 1 resource

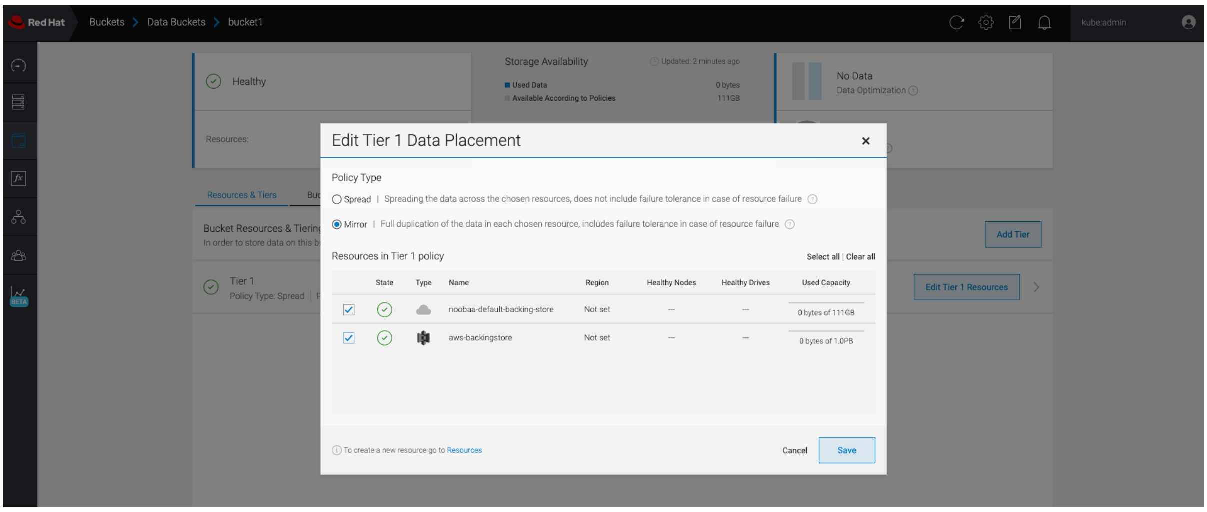
Storage Availability
Updated: 2 minutes ago
Used Data 0 bytes
Available According to Policies 111GB
0 111GB

No Data
Data Optimization
No Usage
Raw Usage

Bucket Resources & Tiering Management
In order to store data on this bucket, resources and tiering policies should be added.

Tier 1
Policy Type: Spread | Pools: 0 | Cloud Resources: 1 | Available Capacity: 111GB of 111GB

5. Select **Mirror** and check the relevant resources you want to use for this bucket. In the following example, the data between **noobaa-default-backing-store** which is on RGW and **AWS-backingstore** which is on AWS is mirrored:



Edit Tier 1 Data Placement

Policy Type

○ Spread | Spreading the data across the chosen resources, does not include failure tolerance in case of resource failure

● Mirror | Full duplication of the data in each chosen resource, includes failure tolerance in case of resource failure

Resources in Tier 1 policy

Select all | Clear all

	State	Type	Name	Region	Healthy Nodes	Healthy Drives	Used Capacity
☑	✓		noobaa-default-backing-store	Not set	—	—	0 bytes of 111GB
☑	✓		aws-backingstore	Not set	—	—	0 bytes of 1.0PB

To create a new resource go to Resources

Cancel Save

6. Click **Save**.

NOTE

Resources created in NooBaa UI cannot be used by OpenShift UI or MCG CLI.

CHAPTER 7. BUCKET POLICIES IN THE MULTICLOUD OBJECT GATEWAY

OpenShift Container Storage supports AWS S3 bucket policies. Bucket policies allow you to grant users access permissions for buckets and the objects in them.

7.1. ABOUT BUCKET POLICIES

Bucket policies are an access policy option available for you to grant permission to your AWS S3 buckets and objects. Bucket policies use JSON-based access policy language. For more information about access policy language, see AWS Access Policy Language Overview .

7.2. USING BUCKET POLICIES

Prerequisites

- A running OpenShift Container Storage Platform
- Access to the Multicloud Object Gateway, see Chapter 2, *Accessing the Multicloud Object Gateway with your applications*

Procedure

To use bucket policies in the Multicloud Object Gateway:

1. Create the bucket policy in JSON format. See the following example:

```
{
    "Version": "NewVersion",
    "Statement": [
        {
            "Sid": "Example",
            "Effect": "Allow",
            "Principal": [
                    "john.doe@example.com"
            ],
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::john_bucket"
            ]
        }
    ]
}
```

There are many available elements for bucket policies with regard to access permissions.

For details on these elements and examples of how they can be used to control the access permissions, see AWS Access Policy Language Overview .

For more examples of bucket policies, see AWS Bucket Policy Examples .

Instructions for creating S3 users can be found in Section 7.3, “Creating an AWS S3 user in the Multicloud Object Gateway”.

2. Using AWS S3 client, use the **put-bucket-policy** command to apply the bucket policy to your S3 bucket:

```
# aws --endpoint ENDPOINT --no-verify-ssl s3api put-bucket-policy --bucket MyBucket --
policy BucketPolicy
```

Replace ***ENDPOINT*** with the S3 endpoint

Replace ***MyBucket*** with the bucket to set the policy on

Replace ***BucketPolicy*** with the bucket policy JSON file

Add **--no-verify-ssl** if you are using the default self signed certificates

For example:

```
# aws --endpoint https://s3-openshift-storage.apps.gogo44.noobaa.org --no-verify-ssl s3api
put-bucket-policy -bucket MyBucket --policy file://BucketPolicy
```

For more information on the **put-bucket-policy** command, see the AWS CLI Command Reference for put-bucket-policy.

NOTE

The principal element specifies the user that is allowed or denied access to a resource, such as a bucket. Currently, Only NooBaa accounts can be used as principals. In the case of object bucket claims, NooBaa automatically create an account **obc-account.<generated bucket name>@noobaa.io**.

NOTE

Bucket policy conditions are not supported.

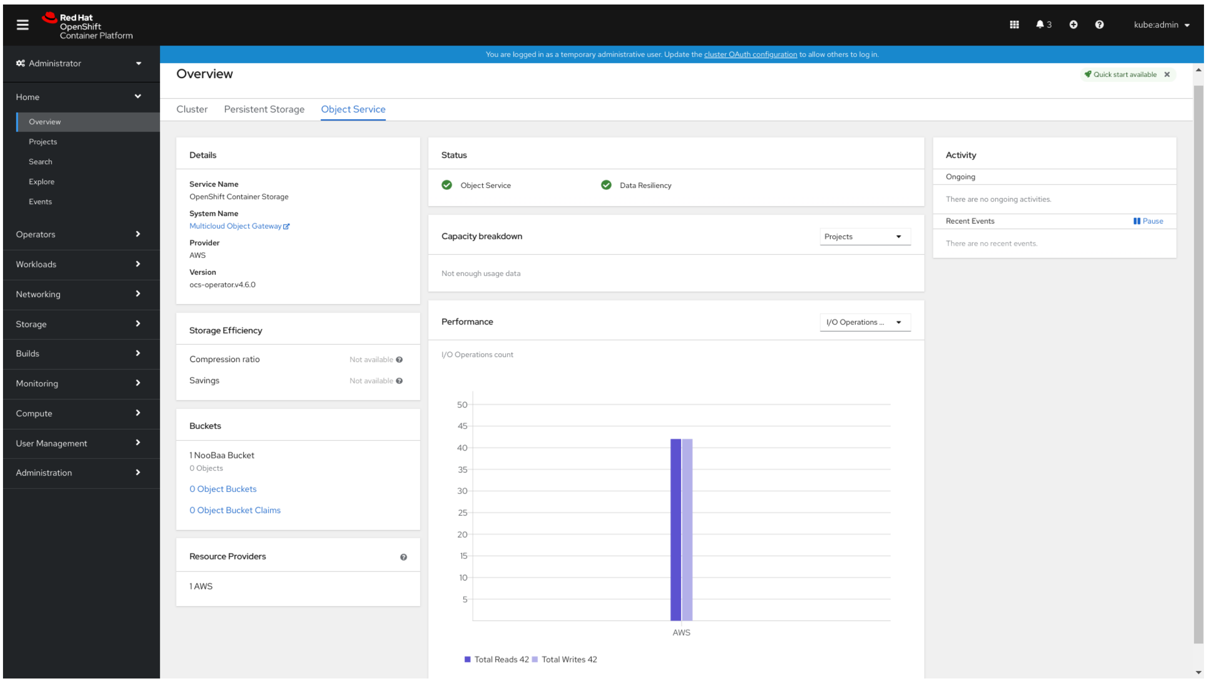
7.3. CREATING AN AWS S3 USER IN THE MULTICLOUD OBJECT GATEWAY

Prerequisites

- A running OpenShift Container Storage Platform
- Access to the Multicloud Object Gateway, see Chapter 2, *Accessing the Multicloud Object Gateway with your applications*

Procedure

1. In your OpenShift Storage console, navigate to **Overview** → **Object Service** → select the **Multicloud Object Gateway** link:



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Service Name
OpenShift Container Storage

System Name
Multicloud Object Gateway

Provider
AWS

Version
ocs-operator.v4.6.0

Object Service
Data Resiliency

Not enough usage data

Ongoing

There are no ongoing activities.

Recent Events

There are no recent events.

Storage Efficiency

Compression ratio Not available

Savings Not available

Buckets

1 NooBaa Bucket
0 Objects

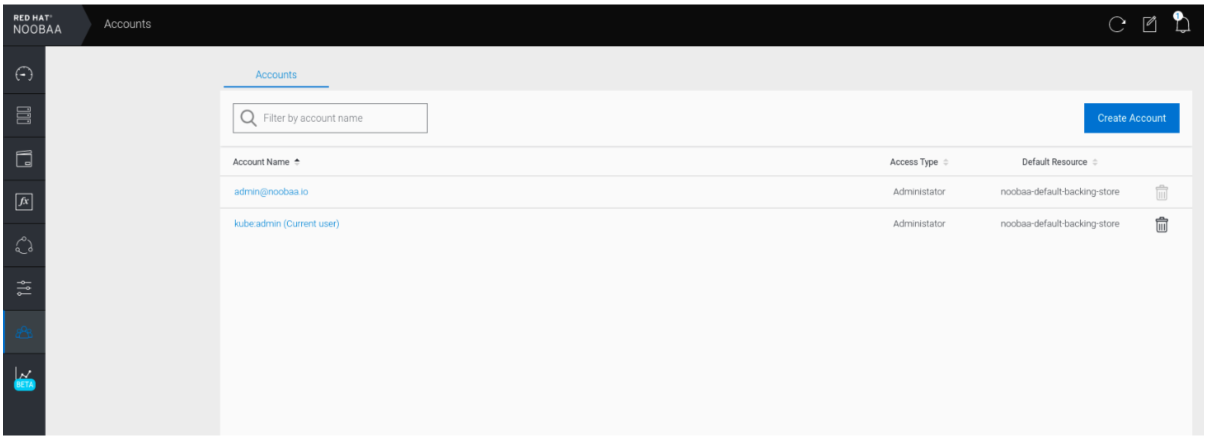
0 Object Buckets

0 Object Bucket Claims

Resource Providers

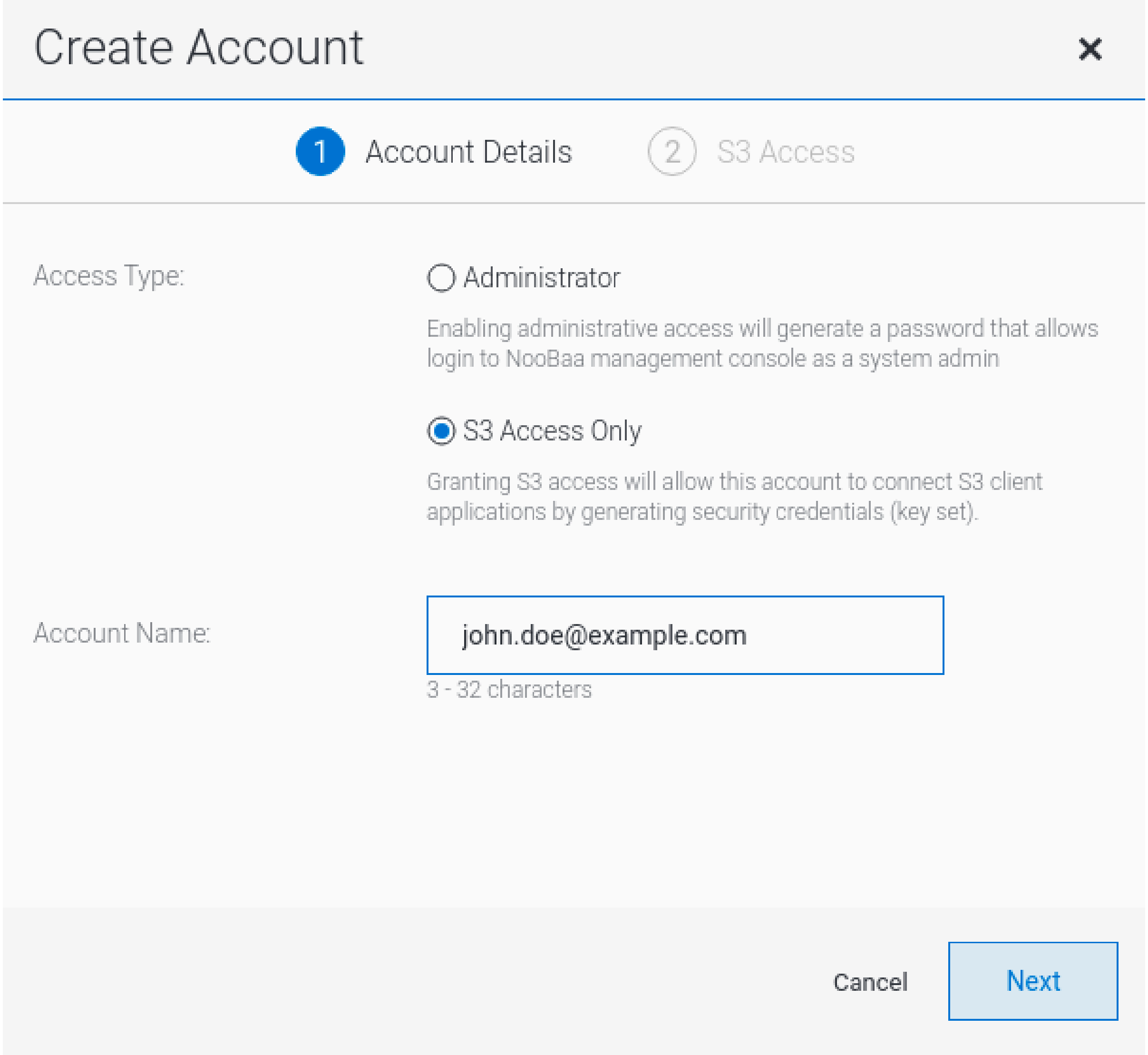
1 AWS

2. Under the **Accounts** tab, click **Create Account**



Account Name	Access Type	Default Resource
admin@noobaa.io	Administator	noobaa-default-backing-store
kube:admin (Current user)	Administator	noobaa-default-backing-store

3. Select **S3 Access Only**, provide the **Account Name**, for example, john.doe@example.com. Click Next:



Create Account ×

1 Account Details 2 S3 Access

Access Type:

○ Administrator

Enabling administrative access will generate a password that allows login to NooBaa management console as a system admin

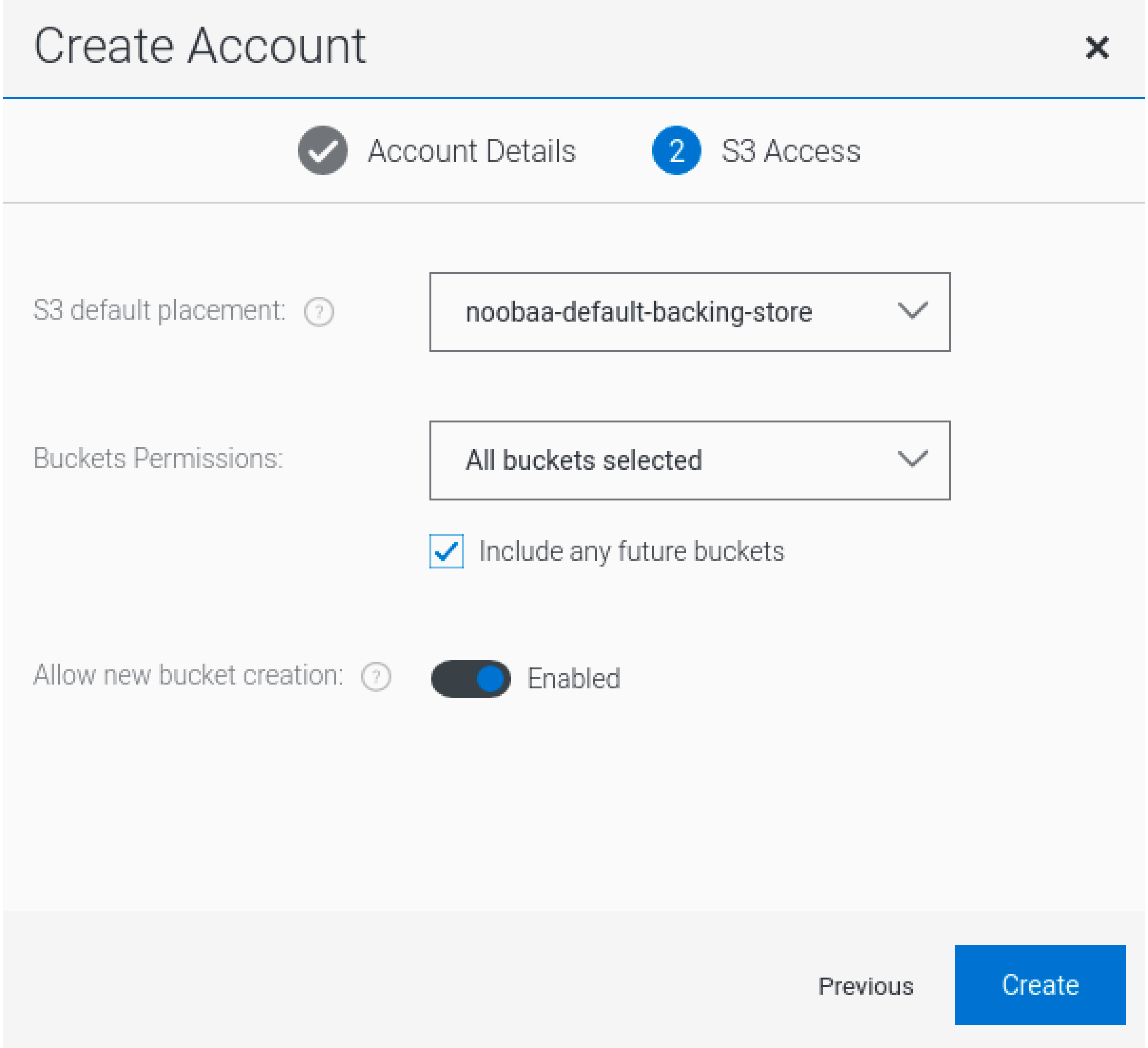
● S3 Access Only

Granting S3 access will allow this account to connect S3 client applications by generating security credentials (key set).

Account Name: john.doe@example.com

3 - 32 characters

4. Select **S3 default placement**, for example, noobaa-default-backing-store. Select **Buckets Permissions**. A specific bucket or all buckets can be selected. Click **Create**:



Create Account

Account Details

2 S3 Access

S3 default placement: noobaa-default-backing-store

Buckets Permissions: All buckets selected

☑ Include any future buckets

Allow new bucket creation: Enabled

CHAPTER 8. OBJECT BUCKET CLAIM

An Object Bucket Claim can be used to request an S3 compatible bucket backend for your workloads.

You can create an Object Bucket Claim three ways:

- Section 8.1, “Dynamic Object Bucket Claim”
- Section 8.2, “Creating an Object Bucket Claim using the command line interface”
- Section 8.3, “Creating an Object Bucket Claim using the OpenShift Web Console”

An object bucket claim creates a new bucket and an application account in NooBaa with permissions to the bucket, including a new access key and secret access key. The application account is allowed to access only a single bucket and can’t create new buckets by default.

8.1. DYNAMIC OBJECT BUCKET CLAIM

Similar to Persistent Volumes, you can add the details of the Object Bucket claim to your application’s YAML, and get the object service endpoint, access key, and secret access key available in a configuration map and secret. It is easy to read this information dynamically into environment variables of your application.

Procedure

1. Add the following lines to your application YAML:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <obc-name>
spec:
  generateBucketName: <obc-bucket-name>
  storageClassName: openshift-storage.noobaa.io
```

   These lines are the Object Bucket Claim itself.

   a. Replace **<obc-name>** with the a unique Object Bucket Claim name.

   b. Replace **<obc-bucket-name>** with a unique bucket name for your Object Bucket Claim.

2. You can add more lines to the YAML file to automate the use of the Object Bucket Claim. The example below is the mapping between the bucket claim result, which is a configuration map with data and a secret with the credentials. This specific job will claim the Object Bucket from NooBaa, which will create a bucket and an account.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
```

```
    - image: <your application image>
      name: test
      env:
        - name: BUCKET_NAME
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_NAME
        - name: BUCKET_HOST
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_HOST
        - name: BUCKET_PORT
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_PORT
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: <obc-name>
              key: AWS_ACCESS_KEY_ID
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: <obc-name>
              key: AWS_SECRET_ACCESS_KEY
```

a. Replace all instances of <obc-name> with your Object Bucket Claim name.

b. Replace <your application image> with your application image.

3. Apply the updated YAML file:

```
# oc apply -f <yaml.file>
```

a. Replace **<yaml.file>** with the name of your YAML file.

4. To view the new configuration map, run the following:

```
# oc get cm <obc-name>
```

a. Replace **obc-name** with the name of your Object Bucket Claim.
You can expect the following environment variables in the output:

- **BUCKET_HOST** - Endpoint to use in the application
- **BUCKET_PORT** - The port available for the application
  - The port is related to the **BUCKET_HOST**. For example, if the **BUCKET_HOST** is https://my.example.com, and the **BUCKET_PORT** is 443, the endpoint for the object service would be https://my.example.com:443.
- **BUCKET_NAME** - Requested or generated bucket name

- **AWS_ACCESS_KEY_ID** - Access key that is part of the credentials
- **AWS_SECRET_ACCESS_KEY** - Secret access key that is part of the credentials

IMPORTANT

Retrieve the **AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY**. The names are used so that it is compatible with the AWS S3 API. You need to specify the keys while performing S3 operations, especially when you read, write or list from the Multicloud Object Gateway (MCG) bucket. The keys are encoded in Base64. Decode the keys before using them.

```
# oc get secret <obc_name> -o yaml
```

<obc_name>

Specify the name of the object bucket claim.

8.2. CREATING AN OBJECT BUCKET CLAIM USING THE COMMAND LINE INTERFACE

When creating an Object Bucket Claim using the command-line interface, you get a configuration map and a Secret that together contain all the information your application needs to use the object storage service.

Prerequisites

- Download the MCG command-line interface:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

Procedure

1. Use the command-line interface to generate the details of a new bucket and credentials. Run the following command:

```
# noobaa obc create <obc-name> -n openshift-storage
```

Replace **<obc-name>** with a unique Object Bucket Claim name, for example, **myappobc**.

Additionally, you can use the **--app-namespace** option to specify the namespace where the Object Bucket Claim configuration map and secret will be created, for example, **myapp-namespace**.

Example output:

```
INFO[0001]   Created: ObjectBucketClaim "test21obc"
```

The MCG command-line-interface has created the necessary configuration and has informed OpenShift about the new OBC.

2. Run the following command to view the Object Bucket Claim:

```
# oc get obc -n openshift-storage
```

Example output:

```
NAME        STORAGE-CLASS                 PHASE   AGE
test21obc   openshift-storage.noobaa.io   Bound   38s
```

3. Run the following command to view the YAML file for the new Object Bucket Claim:

```
# oc get obc test21obc -o yaml -n openshift-storage
```

Example output:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
  - objectbucket.io/finalizer
  generation: 2
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  resourceVersion: "40756"
  selfLink: /apis/objectbucket.io/v1alpha1/namespaces/openshift-
storage/objectbucketclaims/test21obc
  uid: 64f04cba-f662-11e9-bc3c-0295250841af
spec:
  ObjectBucketName: obc-openshift-storage-test21obc
  bucketName: test21obc-933348a6-e267-4f82-82f1-e59bf4fe3bb4
  generateBucketName: test21obc
  storageClassName: openshift-storage.noobaa.io
status:
  phase: Bound
```

4. Inside of your **openshift-storage** namespace, you can find the configuration map and the secret to use this Object Bucket Claim. The CM and the secret have the same name as the Object Bucket Claim. To view the secret:

```
# oc get -n openshift-storage secret test21obc -o yaml
```

Example output:

```
Example output:
apiVersion: v1
data:
  AWS_ACCESS_KEY_ID: c0M0R2xVanF3ODR3bHBkVW94cmY=
  AWS_SECRET_ACCESS_KEY:
Wi9kcFluSWxHRzlWaFlzNk1hc0xma2JXcjM1MVhqa051SlBleXpmOQ==
kind: Secret
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
  - objectbucket.io/finalizer
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  ownerReferences:
  - apiVersion: objectbucket.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ObjectBucketClaim
    name: test21obc
    uid: 64f04cba-f662-11e9-bc3c-0295250841af
  resourceVersion: "40751"
  selfLink: /api/v1/namespaces/openshift-storage/secrets/test21obc
  uid: 65117c1c-f662-11e9-9094-0a5305de57bb
type: Opaque
```

The secret gives you the S3 access credentials.

5. To view the configuration map:

```
# oc get -n openshift-storage cm test21obc -o yaml
```

Example output:

```
apiVersion: v1
data:
  BUCKET_HOST: 10.0.171.35
  BUCKET_NAME: test21obc-933348a6-e267-4f82-82f1-e59bf4fe3bb4
  BUCKET_PORT: "31242"
  BUCKET_REGION: ""
  BUCKET_SUBREGION: ""
kind: ConfigMap
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
  - objectbucket.io/finalizer
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  ownerReferences:
  - apiVersion: objectbucket.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ObjectBucketClaim
    name: test21obc
    uid: 64f04cba-f662-11e9-bc3c-0295250841af
```

```
resourceVersion: "40752"
selfLink: /api/v1/namespaces/openshift-storage/configmaps/test21obc
uid: 651c6501-f662-11e9-9094-0a5305de57bb
```

The configuration map contains the S3 endpoint information for your application.

8.3. CREATING AN OBJECT BUCKET CLAIM USING THE OPENSHIFT WEB CONSOLE

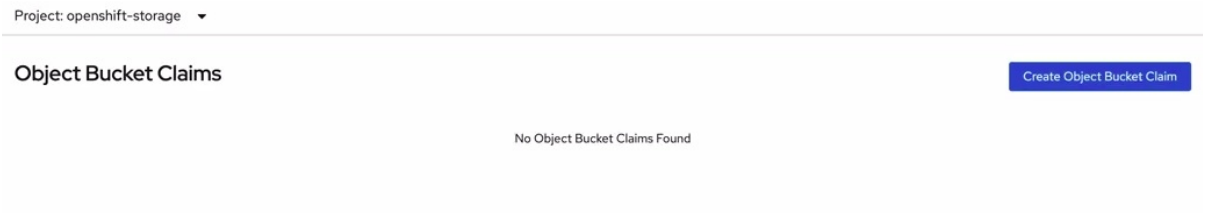
You can create an Object Bucket Claim (OBC) using the OpenShift Web Console.

Prerequisites

- Administrative access to the OpenShift Web Console.
- In order for your applications to communicate with the OBC, you need to use the configmap and secret. For more information about this, see Section 8.1, “Dynamic Object Bucket Claim” .

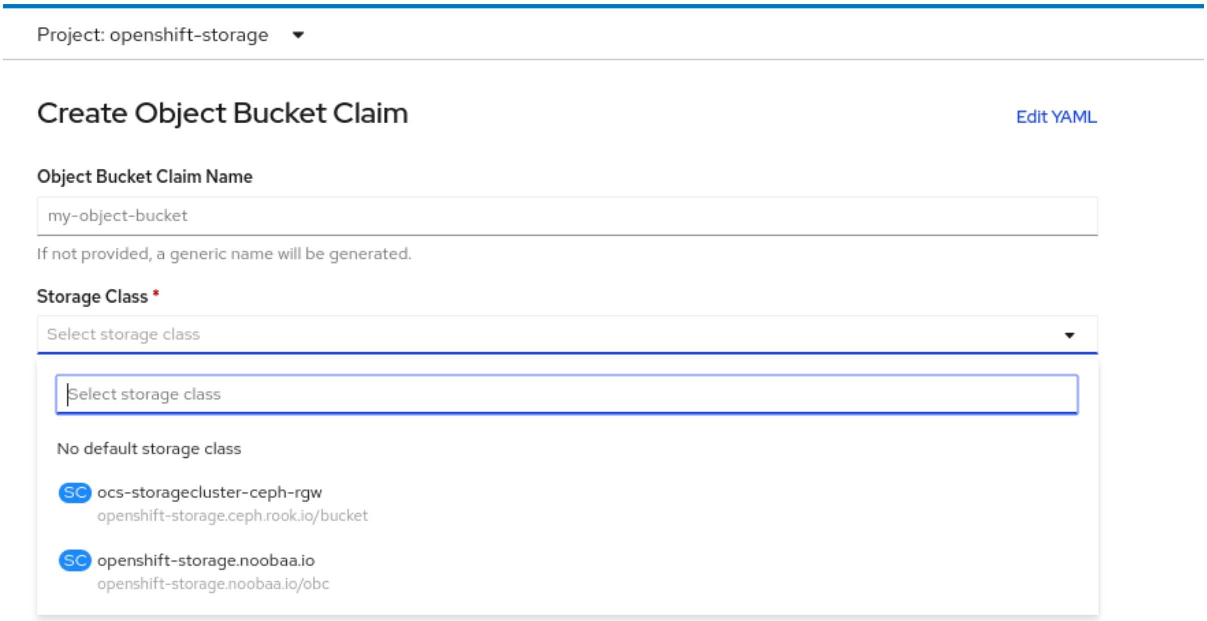
Procedure

1. Log into the OpenShift Web Console.
2. On the left navigation bar, click **Storage → Object Bucket Claims**.
3. Click **Create Object Bucket Claim**



No Object Bucket Claims Found

4. Enter a name for your object bucket claim and select the appropriate storage class based on your deployment, internal or external, from the dropdown menu:
Internal mode



my-object-bucket

If not provided, a generic name will be generated.

Select storage class

No default storage class

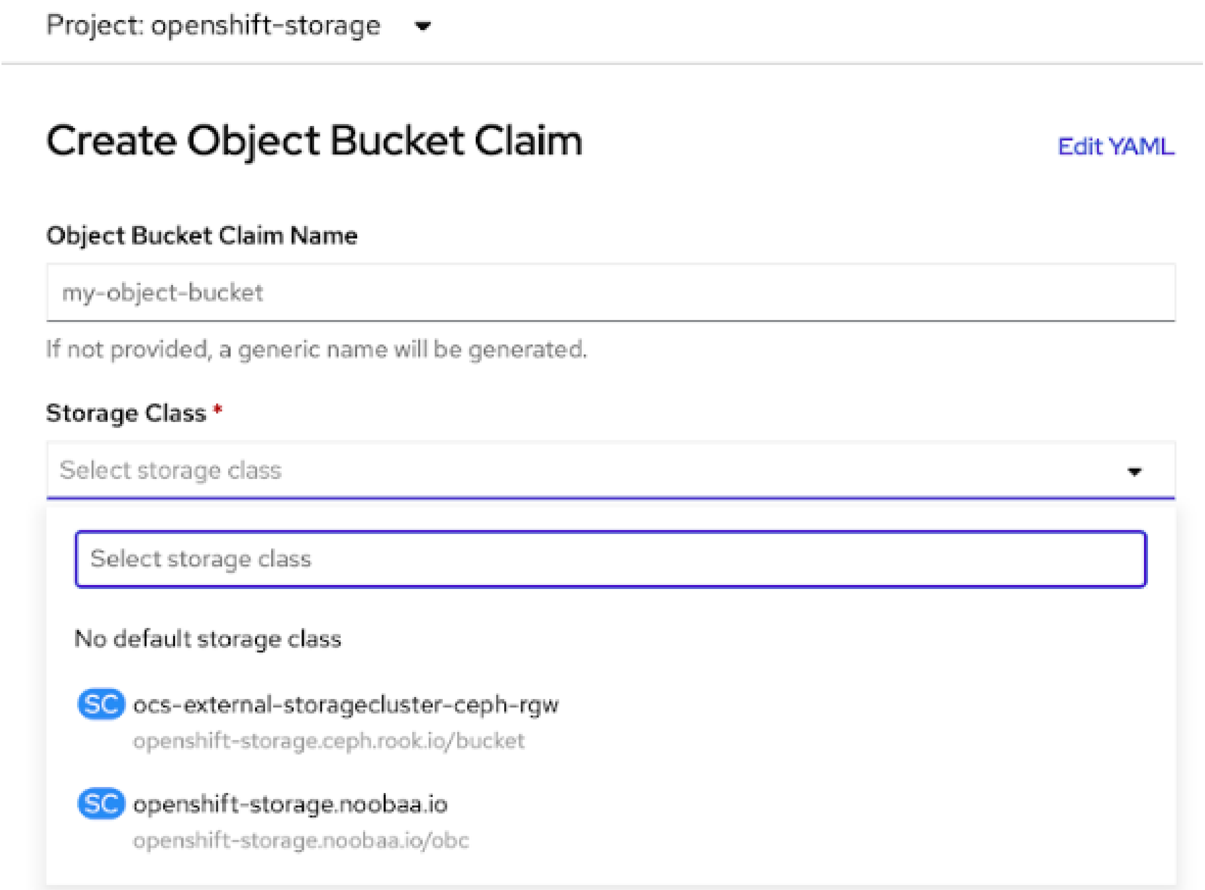
SC ocs-storagecluster-ceph-rgw
openshift-storage.ceph.rook.io/bucket

SC openshift-storage.noobaa.io
openshift-storage.noobaa.io/obc

The following storage classes, which were created after deployment, are available for use:

- **ocs-storagecluster-ceph-rgw** uses the Ceph Object Gateway (RGW)
- **openshift-storage.noobaa.io** uses the Multicloud Object Gateway

External mode


Project: openshift-storage

Create Object Bucket Claim

Edit YAML

Object Bucket Claim Name

my-object-bucket

If not provided, a generic name will be generated.

Storage Class *

Select storage class

Select storage class

No default storage class

SC ocs-external-storagecluster-ceph-rgw
openshift-storage.ceph.rook.io/bucket

SC openshift-storage.noobaa.io
openshift-storage.noobaa.io/obc

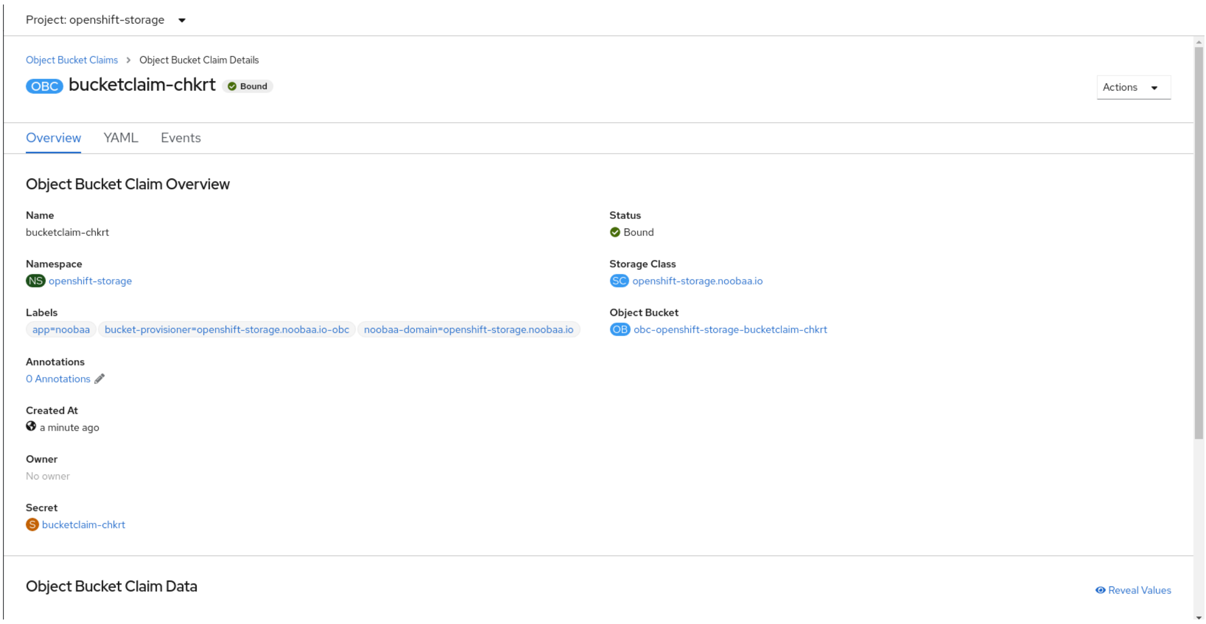
The following storage classes, which were created after deployment, are available for use:

- **ocs-external-storagecluster-ceph-rgw** uses the Ceph Object Gateway (RGW)
- **openshift-storage.noobaa.io** uses the Multicloud Object Gateway

NOTE

The RGW OBC storage class is only available with fresh installations of OpenShift Container Storage version 4.5. It does not apply to clusters upgraded from previous OpenShift Container Storage releases.

5. Click **Create**.
   Once you create the OBC, you are redirected to its detail page:


Project: openshift-storage
Object Bucket Claims > Object Bucket Claim Details
OBC bucketclaim-chkrt Bound
Actions
Overview
YAML
Events
Reveal Values

Object Bucket Claim Overview

Name
bucketclaim-chkrt

Status
Bound

Namespace
NS openshift-storage

Storage Class
SC openshift-storage.noobaa.io

Labels
app=noobaa | bucket-provisioner=openshift-storage.noobaa.io-obc | noobaa-domain=openshift-storage.noobaa.io

Object Bucket
OB obc-openshift-storage-bucketclaim-chkrt

Annotations
0 Annotations

Created At
a minute ago

Owner
No owner

Secret
S bucketclaim-chkrt

Object Bucket Claim Data

Additional Resources

- *Chapter 8, Object Bucket Claim*

8.4. ATTACHING AN OBJECT BUCKET CLAIM TO A DEPLOYMENT

Once created, Object Bucket Claims (OBCs) can be attached to specific deployments.

Prerequisites

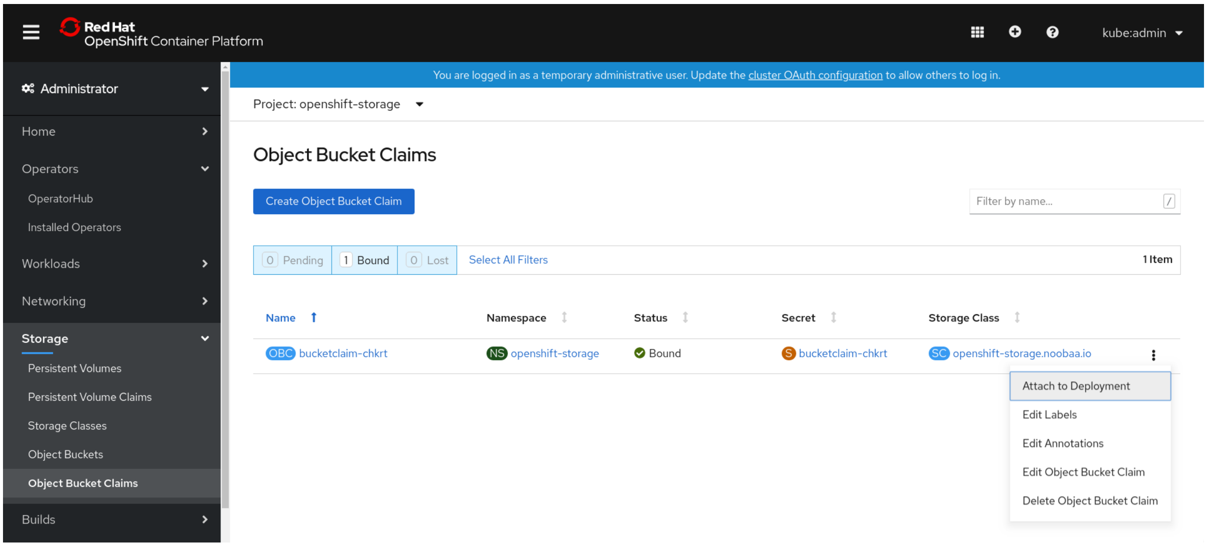
- Administrative access to the OpenShift Web Console.

Procedure

1. On the left navigation bar, click **Storage** → **Object Bucket Claims**.

2. Click the action menu (⋮) next to the OBC you created.

3. From the drop down menu, select **Attach to Deployment**

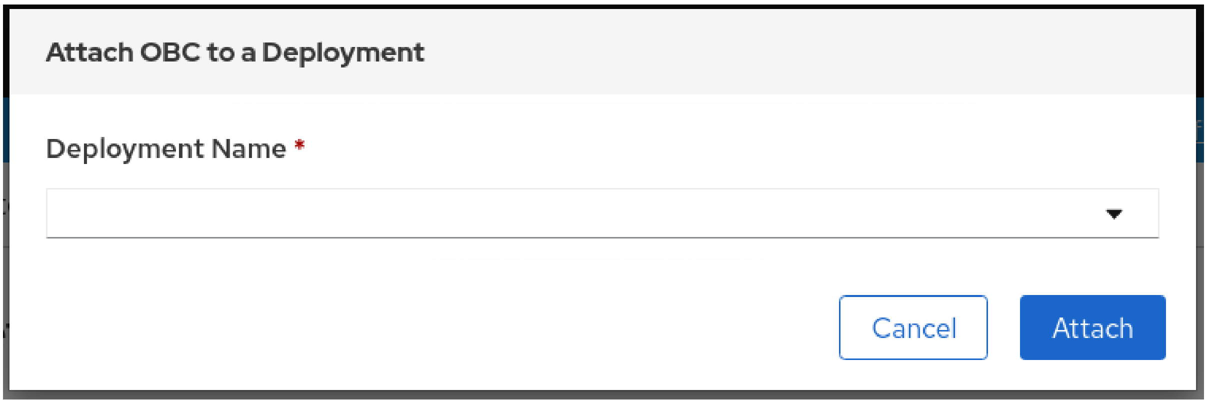


You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Object Bucket Claims

Name	Namespace	Status	Secret	Storage Class
OBC bucketclaim-chkrt	NS openshift-storage	Bound	S bucketclaim-chkrt	SC openshift-storage.noobaa.io

4. Select the desired deployment from the Deployment Name list, then click **Attach**:



Attach OBC to a Deployment

Deployment Name *

Additional Resources

- *Chapter 8, Object Bucket Claim*

8.5. VIEWING OBJECT BUCKETS USING THE OPENSHIFT WEB CONSOLE

You can view the details of object buckets created for Object Bucket Claims (OBCs) using the OpenShift Web Console.

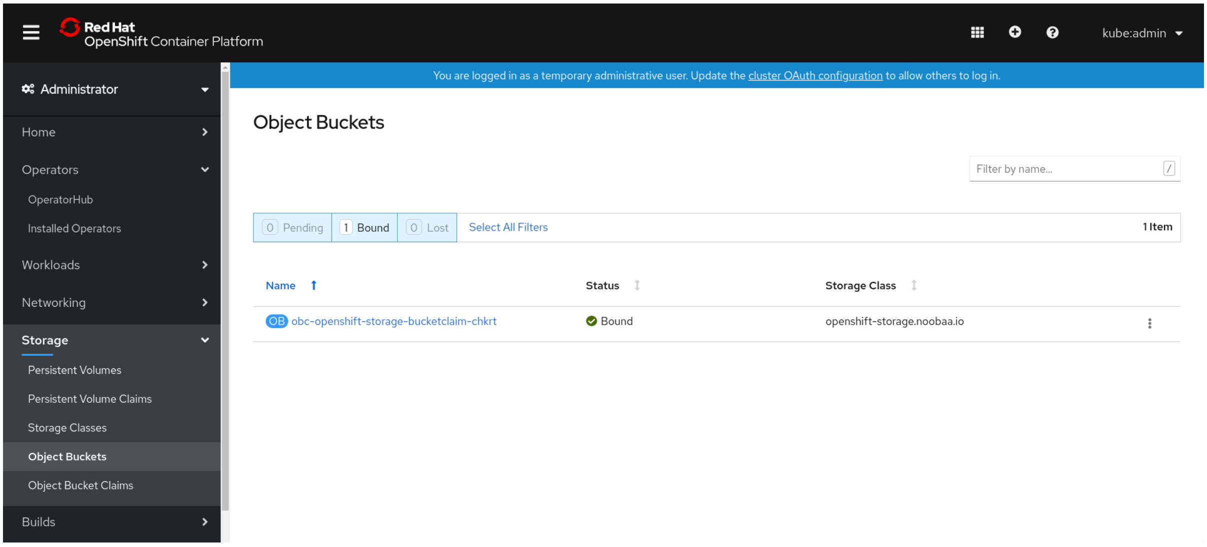
Prerequisites

- Administrative access to the OpenShift Web Console.

Procedure

To view the object bucket details:

1. Log into the OpenShift Web Console.
2. On the left navigation bar, click **Storage → Object Buckets**:



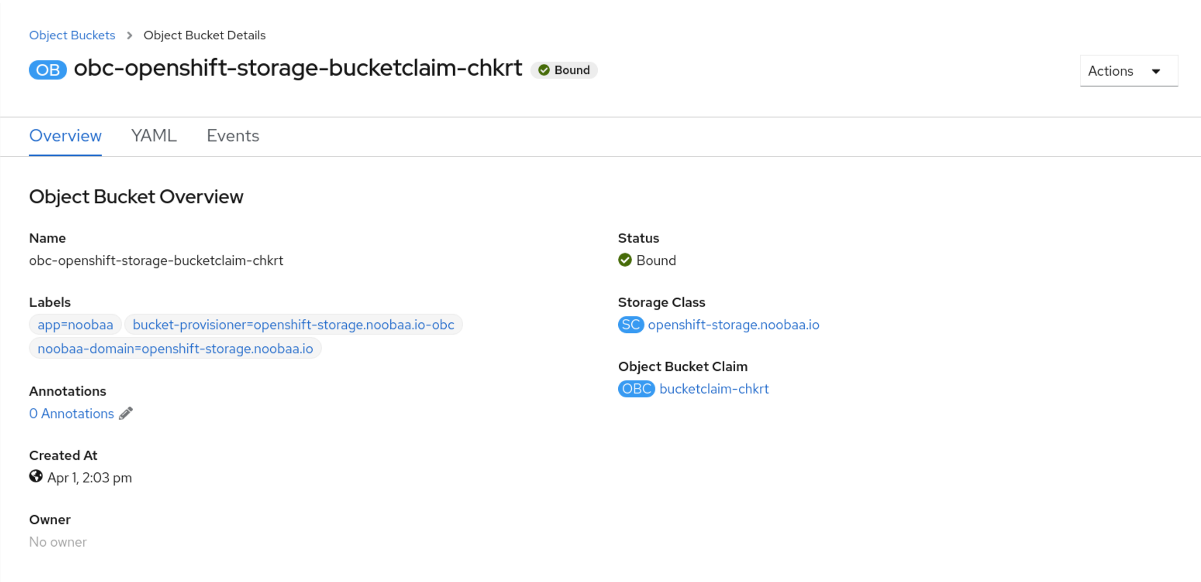
You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Object Buckets

Name	Status	Storage Class
OB obc-openshift-storage-bucketclaim-chkrt	Bound	openshift-storage.noobaa.io

You can also navigate to the details page of a specific OBC and click the **Resource** link to view the object buckets for that OBC.

3. Select the object bucket you want to see details for. You are navigated to the object bucket's details page:



Object Buckets > Object Bucket Details

OB obc-openshift-storage-bucketclaim-chkrt ✓ Bound

Actions

Overview YAML Events

Object Bucket Overview

Name
obc-openshift-storage-bucketclaim-chkrt

Labels
app=noobaa bucket-provisioner=openshift-storage.noobaa.io-obc
noobaa-domain=openshift-storage.noobaa.io

Annotations
0 Annotations

Created At
Apr 1, 2:03 pm

Owner
No owner

Status
✓ Bound

Storage Class
SC openshift-storage.noobaa.io

Object Bucket Claim
OBC bucketclaim-chkrt

Additional Resources

- Chapter 8, *Object Bucket Claim*

8.6. DELETING OBJECT BUCKET CLAIMS

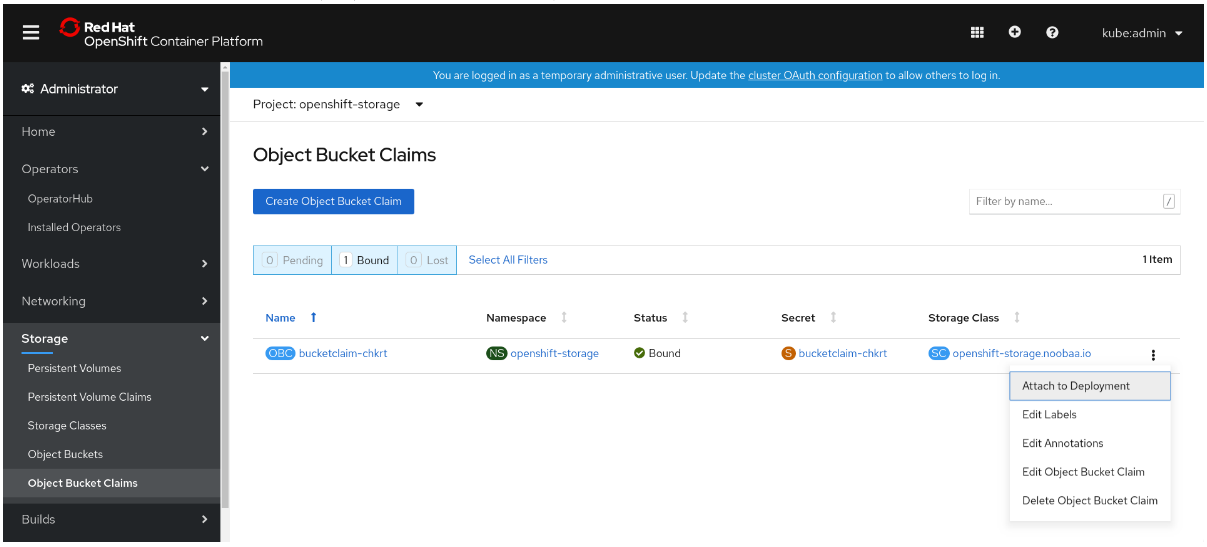
Prerequisites

- Administrative access to the OpenShift Web Console.

Procedure

1. On the left navigation bar, click **Storage → Object Bucket Claims**.

2. click on the action menu (⋮) next to the Object Bucket Claim you want to delete.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Project: openshift-storage

Object Bucket Claims

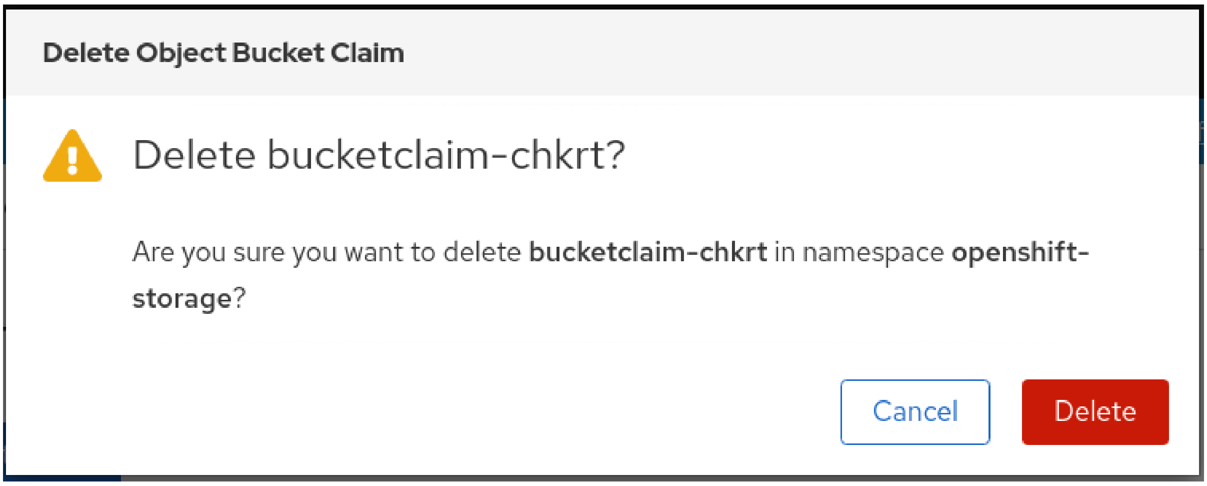
Create Object Bucket Claim

Filter by name...

0 Pending 1 Bound 0 Lost Select All Filters 1 Item

Name	Namespace	Status	Secret	Storage Class
OBC bucketclaim-chkrt	NS openshift-storage	✓ Bound	S bucketclaim-chkrt	SC openshift-storage.noobaa.io

3. Select **Delete Object Bucket Claim** from menu.



Delete Object Bucket Claim

Delete bucketclaim-chkrt?

Are you sure you want to delete **bucketclaim-chkrt** in namespace **openshift-storage**?

4. Click **Delete**.

Additional Resources

- Chapter 8, *Object Bucket Claim*

CHAPTER 9. CACHING POLICY FOR OBJECT BUCKETS

A cache bucket is a namespace bucket with a hub target and a cache target. The hub target is an S3 compatible large object storage bucket. The cache bucket is the local Multicloud Object Gateway bucket. You can create a cache bucket that caches an AWS bucket or an IBM COS bucket.

IMPORTANT

Cache buckets are a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information, see Technology Preview Features Support Scope .

- AWS S3
- IBM COS

9.1. CREATING AN AWS CACHE BUCKET

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

  Alternatively, you can install the mcg package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package.

Procedure

1. Create a NamespaceStore resource. A NamespaceStore represents an underlying storage to be used as a read or write target for the data in Multicloud Object Gateway namespace buckets. From the MCG command-line interface, run the following command:

```
noobaa namespacestore create aws-s3 <namespacestore> --access-key <AWS ACCESS
KEY> --secret-key <AWS SECRET ACCESS KEY> --target-bucket <bucket-name>
```

   a. Replace **<namespacestore>** with the name of the namespacestore.

   b. Replace **<AWS ACCESS KEY>** and **<AWS SECRET ACCESS KEY>** with an AWS access key ID and secret access key you created for this purpose.

   c. Replace **<bucket-name>** with an existing AWS bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.
   You can also add storage resources by applying a YAML. First create a secret with credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: <namespacestore-secret-name>
type: Opaque
data:
  AWS_ACCESS_KEY_ID: <AWS ACCESS KEY ID ENCODED IN BASE64>
  AWS_SECRET_ACCESS_KEY: <AWS SECRET ACCESS KEY ENCODED IN BASE64>
```

You must supply and encode your own AWS access key ID and secret access key using Base64, and use the results in place of **<AWS ACCESS KEY ID ENCODED IN BASE64>** and **<AWS SECRET ACCESS KEY ENCODED IN BASE64>**.

Replace **<namespacestore-secret-name>** with a unique name.

Then apply the following YAML:

```
apiVersion: noobaa.io/v1alpha1
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <namespacestore>
  namespace: openshift-storage
spec:
  awsS3:
    secret:
      name: <namespacestore-secret-name>
      namespace: <namespace-secret>
    targetBucket: <target-bucket>
  type: aws-s3
```

d. Replace **<namespacestore>** with a unique name.

e. Replace **<namespacestore-secret-name>** with the secret created in the previous step.

f. Replace **<namespace-secret>** with the namespace used to create the secret in the previous step.

g. Replace **<target-bucket>** with the AWS S3 bucket you created for the namespacestore.

2. Run the following command to create a bucket class:

```
noobaa bucketclass create namespace-bucketclass cache <my-cache-bucket-class> --backingstores <backing-store> --hub-resource <namespacestore>
```

a. Replace **<my-cache-bucket-class>** with a unique bucket class name.

b. Replace **<backing-store>** with the relevant backing store. You can list one or more backingstores separated by commas in this field.

c. Replace **<namespacestore>** with the namespacestore created in the previous step.

3. Run the following command to create a bucket using an Object Bucket Claim resource that uses the bucket class defined in step 2.

   ```
   noobaa obc create <my-bucket-claim> my-app --bucketclass <custom-bucket-class>
   ```

   a. Replace **<my-bucket-claim>** with a unique name.

   b. Replace **<custom-bucket-class>** with the name of the bucket class created in step 2.

9.2. CREATING AN IBM COS CACHE BUCKET

Prerequisites

- Download the Multicloud Object Gateway (MCG) command-line interface:

  ```
  # subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

  Alternatively, you can install the mcg package from the OpenShift Container Storage RPMs found here https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package.

Procedure

1. Create a NamespaceStore resource. A NamespaceStore represents an underlying storage to be used as a read or write target for the data in Multicloud Object Gateway namespace buckets. From the MCG command-line interface, run the following command:

   ```
   noobaa namespacestore create ibm-cos <namespacestore> --endpoint <IBM COS
   ENDPOINT> --access-key <IBM ACCESS KEY> --secret-key <IBM SECRET ACCESS
   KEY> --target-bucket <bucket-name>
   ```

   a. Replace **<namespacestore>** with the name of the NamespaceStore.

   b. Replace **<IBM ACCESS KEY>**, **<IBM SECRET ACCESS KEY>**, **<IBM COS ENDPOINT>** with an IBM access key ID, secret access key and the appropriate regional endpoint that corresponds to the location of the existing IBM bucket.

   c. Replace **<bucket-name>** with an existing IBM bucket name. This argument tells Multicloud Object Gateway which bucket to use as a target bucket for its backing store, and subsequently, data storage and administration.
   You can also add storage resources by applying a YAML. First, Create a secret with the credentials:

      ```
      apiVersion: v1
      kind: Secret
      metadata:
        name: <namespacestore-secret-name>
      type: Opaque
      data:
        IBM_COS_ACCESS_KEY_ID: <IBM COS ACCESS KEY ID ENCODED IN BASE64>
        IBM_COS_SECRET_ACCESS_KEY: <IBM COS SECRET ACCESS KEY ENCODED
      IN BASE64>
      ```

You must supply and encode your own IBM COS access key ID and secret access key using Base64, and use the results in place of **<IBM COS ACCESS KEY ID ENCODED IN BASE64>** and <IBM COS SECRET ACCESS KEY ENCODED IN BASE64>`.

Replace **<namespacestore-secret-name>** with a unique name.

Then apply the following YAML:

```
apiVersion: noobaa.io/v1alpha1
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <namespacestore>
  namespace: openshift-storage
spec:
  s3Compatible:
    endpoint: <IBM COS ENDPOINT>
    secret:
      name: <backingstore-secret-name>
      namespace: <namespace-secret>
    signatureVersion: v2
    targetBucket: <target-bucket>
  type: ibm-cos
```

d. Replace **<namespacestore>** with a unique name.

e. Replace **<IBM COS ENDPOINT>** with the appropriate IBM COS endpoint.

f. Replace **<backingstore-secret-name>** with the secret created in the previous step.

g. Replace **<namespace-secret>** with the namespace used to create the secret in the previous step.

h. Replace **<target-bucket>** with the AWS S3 bucket you created for the namespacestore.

2. Run the following command to create a bucket class:

```
noobaa bucketclass create namespace-bucketclass cache <my-bucket-class> --
backingstores <backing-store> --hubResource <namespacestore>
```

a. Replace **<my-bucket-class>** with a unique bucket class name.

b. Replace **<backing-store>** with the relevant backing store. You can list one or more backingstores separated by commas in this field.

c. Replace **<namespacestore>** with the namespacestore created in the previous step.

3. Run the following command to create a bucket using an Object Bucket Claim resource that uses the bucket class defined in step 2.

```
noobaa obc create <my-bucket-claim> my-app --bucketclass <custom-bucket-class>
```

a. Replace **<my-bucket-claim>** with a unique name.

b. Replace **<custom-bucket-class>** with the name of the bucket class created in step 2.

CHAPTER 10. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE BY ADDING ENDPOINTS

The Multicloud Object Gateway performance may vary from one environment to another. In some cases, specific applications require faster performance which can be easily addressed by scaling S3 endpoints, which is a Technology Preview feature.

The Multicloud Object Gateway resource pool is a group of NooBaa daemon containers that provide two types of services enabled by default:

- Storage service
- S3 endpoint service

IMPORTANT

Scaling Multicloud Object Gateway performance by adding endpoints is a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information, see Technology Preview Features Support Scope .

10.1. S3 ENDPOINTS IN THE MULTICLOUD OBJECT GATEWAY

The S3 endpoint is a service that every Multicloud Object Gateway provides by default that handles the heavy lifting data digestion in the Multicloud Object Gateway. The endpoint service handles the inline data chunking, deduplication, compression, and encryption, and it accepts data placement instructions from the Multicloud Object Gateway.

10.2. SCALING WITH STORAGE NODES

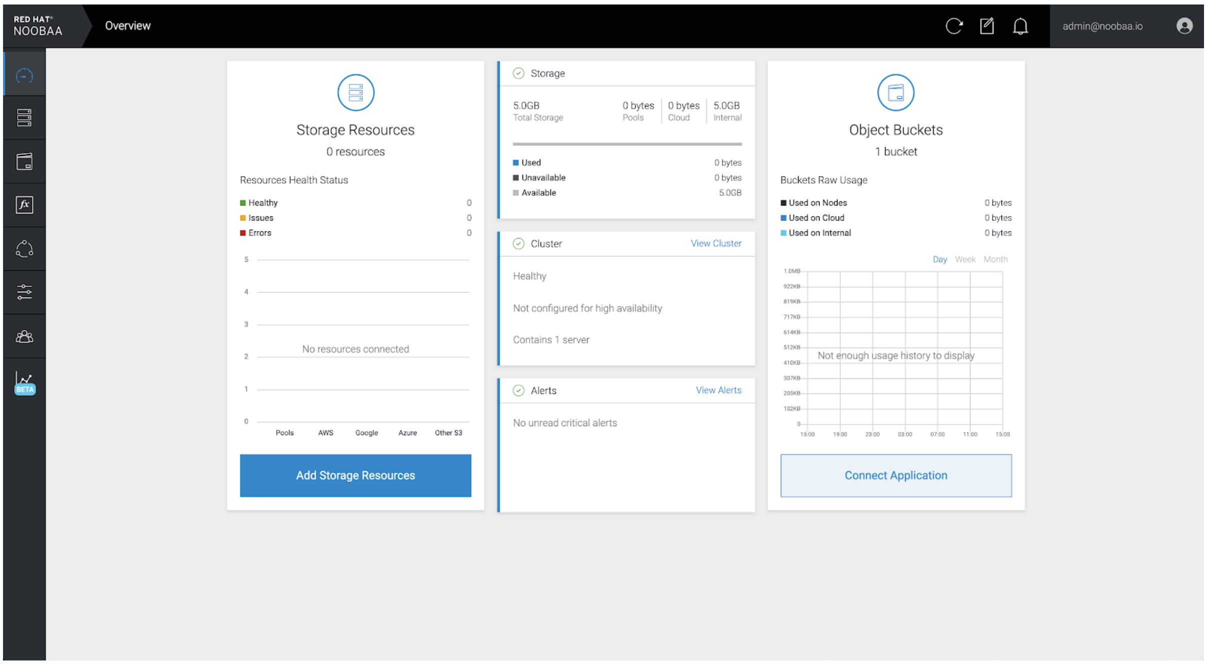
Prerequisites

- A running OpenShift Container Storage cluster on OpenShift Container Platform with access to the Multicloud Object Gateway.

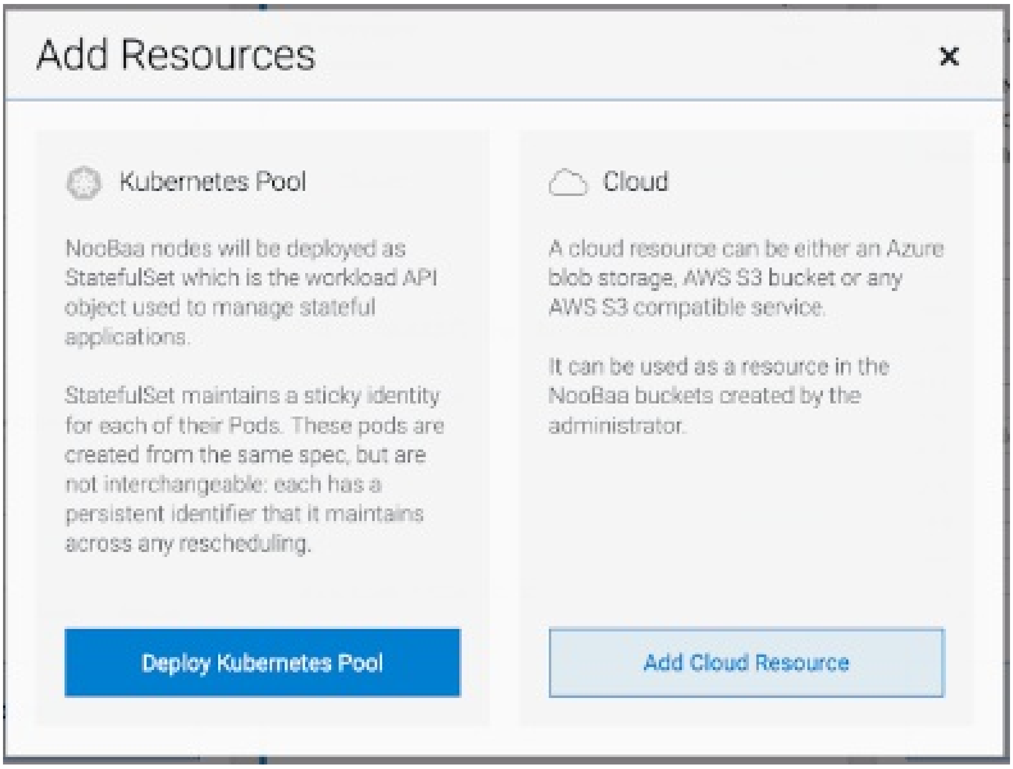
A storage node in the Multicloud Object Gateway is a NooBaa daemon container attached to one or more Persistent Volumes and used for local object service data storage. NooBaa daemons can be deployed on Kubernetes nodes. This can be done by creating a Kubernetes pool consisting of StatefulSet pods.

Procedure

1. In the Multicloud Object Gateway user interface, from the **Overview** page, click **Add Storage Resources**:



2. In the window, click **Deploy Kubernetes Pool**.



Add Resources

Kubernetes Pool

NooBaa nodes will be deployed as StatefulSet which is the workload API object used to manage stateful applications.

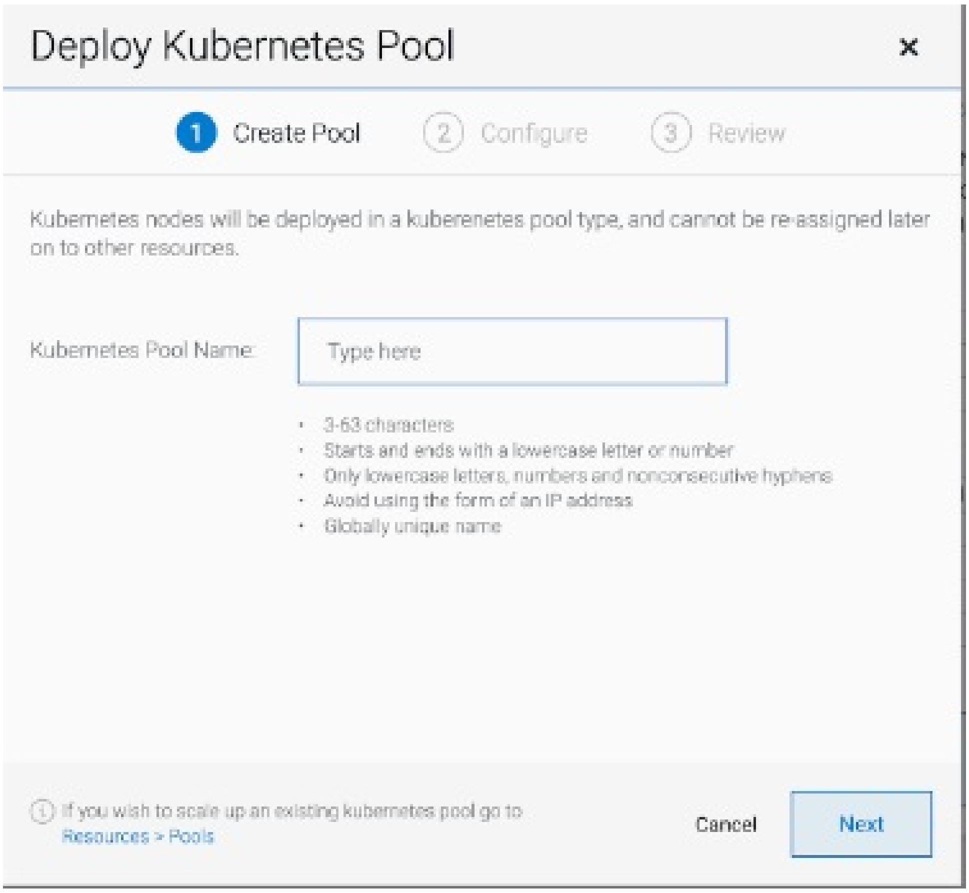
StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

Cloud

A cloud resource can be either an Azure blob storage, AWS S3 bucket or any AWS S3 compatible service.

It can be used as a resource in the NooBaa buckets created by the administrator.

3. In the **Create Pool** step create the target pool for the future installed nodes.



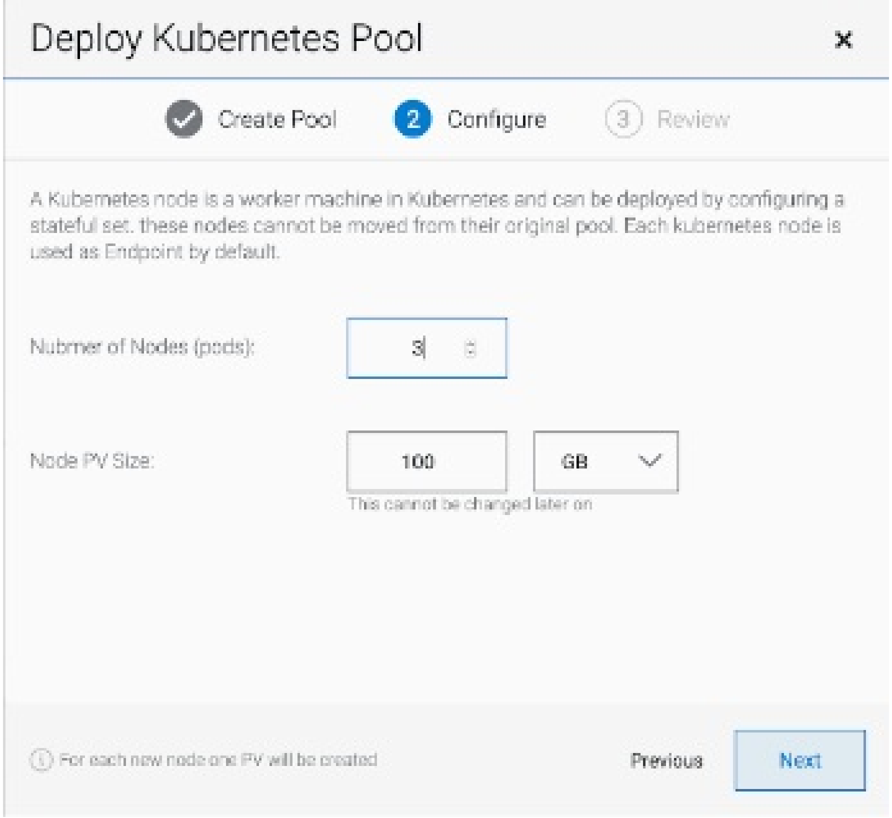
Kubernetes nodes will be deployed in a kuberenetes pool type, and cannot be re-assigned later on to other resources.

Kubernetes Pool Name: Type here

- 3-63 characters
- Starts and ends with a lowercase letter or number
- Only lowercase letters, numbers and nonconsecutive hyphens
- Avoid using the form of an IP address
- Globally unique name

If you wish to scale up an existing kubernetes pool go to Resources > Pools

4. In the **Configure** step, configure the number of requested pods and the size of each PV. For each new pod, one PV is be created.



A Kubernetes node is a worker machine in Kubernetes and can be deployed by configuring a stateful set. these nodes cannot be moved from their original pool. Each kubernetes node is used as Endpoint by default.

Nubmer of Nodes (pods): 3

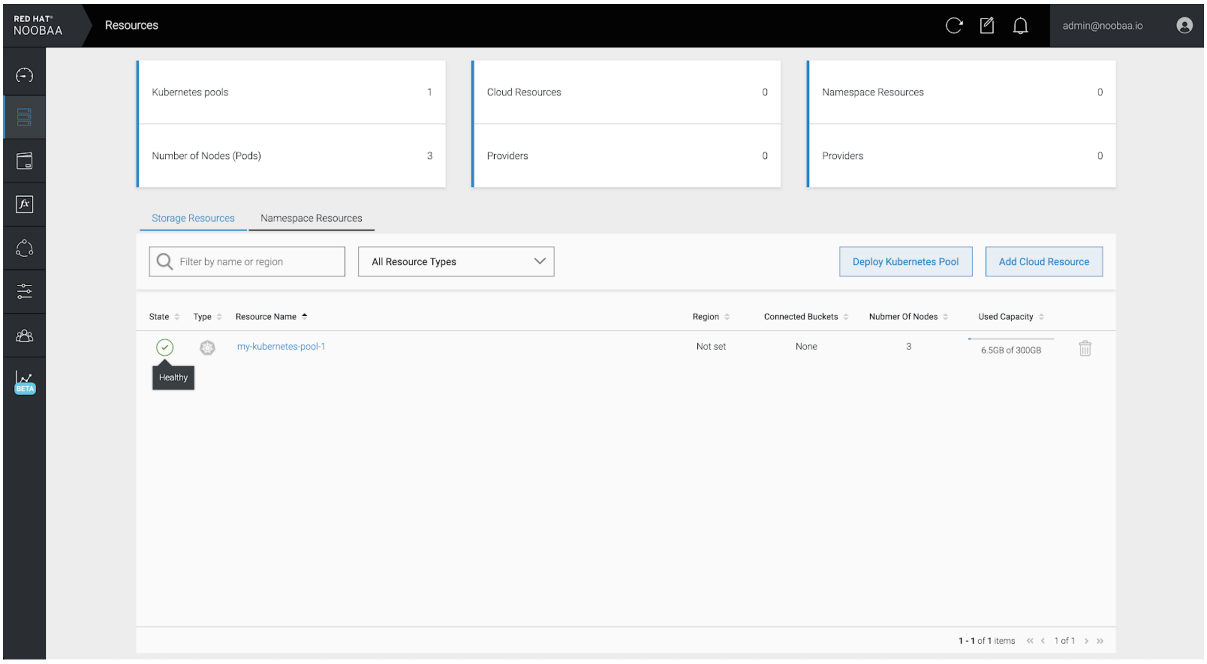
Node PV Size: 100 GB

This cannot be changed later on

For each new node one PV will be created

5. In the **Review** step, you can find the details of the new pool and select the deployment method you wish to use: local or external deployment. If local deployment is selected, the Kubernetes nodes will deploy within the cluster. If external deployment is selected, you will be provided with a YAML file to run externally.

6. All nodes will be assigned to the pool you chose in the first step, and can be found under **Resources → Storage resources → Resource name**:



Kubernetes pools	1	Cloud Resources	0	Namespace Resources	0
Number of Nodes (Pods)	3	Providers	0	Providers	0

State	Type	Resource Name	Region	Connected Buckets	Nubmer Of Nodes	Used Capacity
Healthy		my-kubernetes-pool-1	Not set	None	3	6.5GB of 300GB

1 - 1 of 1 items « ‹ 1 of 1 › »

CHAPTER 11. AUTOMATIC SCALING OF MULTICLOUD OBJECT GATEWAY ENDPOINTS

The number of MultiCloud Object Gateway (MCG) endpoints scale automatically when the load on the MCG S3 service increases or decreases. {product-name-short} clusters are deployed with one active MCG endpoint. Each MCG endpoint pod is configured by default with 1 CPU and 2Gi memory request, with limits matching the request. When the CPU load on the endpoint crosses over an 80% usage threshold for a consistent period of time, a second endpoint is deployed lowering the load on the first endpoint. When the average CPU load on both endpoints falls below the 80% threshold for a consistent period of time, one of the endpoints is deleted. This feature improves performance and serviceability of the MCG.

CHAPTER 12. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT

Users can access the RADOS Object Gateway (RGW) endpoint directly.

Prerequisites

- A running OpenShift Container Storage Platform

Procedure

1. Run **oc get service** command to get the RGW service name.

```
$ oc get service

NAME                                        TYPE
rook-ceph-rgw-ocs-storagecluster-cephobjectstore   ClusterIP

CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
172.30.99.207    <none>        80/TCP     4d15h
```

2. Run **oc expose** command to expose the RGW service.

```
$ oc expose svc/<RGW service name> --hostname=<route name>
```

Replace **<RGW-service name>** with the RGW service name from the previous step.

Replace **<route name>** with a route you want to create for the RGW service.

For example:

```
$ oc expose svc/rook-ceph-rgw-ocs-storagecluster-cephobjectstore --hostname=rook-ceph-rgw-ocs.ocp.host.example.com
```

3. Run **oc get route** command to confirm **oc expose** is successful and there is an RGW route.

```
$ oc get route

NAME                                        HOST/PORT                         PATH
rook-ceph-rgw-ocs-storagecluster-cephobjectstore   rook-ceph-rgw-
ocsocp.host.example.com

SERVICES                                        PORT        TERMINATION   WILDCARD
rook-ceph-rgw-ocs-storagecluster-cephobjectstore   http        <none>
```

Verify

- To verify the **ENDPOINT**, run the following command:

```
aws s3 --no-verify-ssl --endpoint <ENDPOINT> ls
```

Replace **<ENDPOINT>** with the route that you get from the command in the above step 3.

For example:

```
$ aws s3 --no-verify-ssl --endpoint http://rook-ceph-rgw-ocs.ocp.host.example.com ls
```

NOTE

To get the access key and secret of the default user **ocs-storagecluster-cephobjectstoreuser**, run the following commands:

- Access key:

  ```
  $ oc get secret rook-ceph-object-user-ocs-storagecluster-cephobjectstore-ocs-storagecluster-cephobjectstoreuser -n openshift-storage -o yaml | grep -w "AccessKey:" | head -n1 | awk '{print $2}' | base64 --decode
  ```

- Secret key:

  ```
  $ oc get secret rook-ceph-object-user-ocs-storagecluster-cephobjectstore-ocs-storagecluster-cephobjectstoreuser -n openshift-storage -o yaml | grep -w "SecretKey:" | head -n1 | awk '{print $2}' | base64 --decode
  ```