



Red Hat OpenShift Container Storage 4.6

Managing hybrid and multicloud resources

Hybrid and multicloud resource management for cluster and storage administrators

Red Hat OpenShift Container Storage 4.6 Managing hybrid and multicloud resources

Hybrid and multicloud resource management for cluster and storage administrators

Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. ABOUT THE MULTICLOUD OBJECT GATEWAY	3
CHAPTER 2. ACCESSING THE MULTICLOUD OBJECT GATEWAY WITH YOUR APPLICATIONS	4
2.1. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE TERMINAL	4
2.2. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE MCG COMMAND-LINE INTERFACE	6
CHAPTER 3. ALLOWING USER ACCESS TO THE MULTICLOUD OBJECT GATEWAY CONSOLE	10
CHAPTER 4. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD	12
4.1. CREATING A NEW BACKING STORE	12
4.2. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD USING THE MCG COMMAND LINE INTERFACE	13
4.2.1. Creating an AWS-backed backingstore	14
4.2.2. Creating an IBM COS-backed backingstore	15
4.2.3. Creating an Azure-backed backingstore	17
4.2.4. Creating a GCP-backed backingstore	18
4.2.5. Creating a local Persistent Volume-backed backingstore	20
4.3. CREATING AN S3 COMPATIBLE MULTICLOUD OBJECT GATEWAY BACKINGSTORE	21
4.4. ADDING STORAGE RESOURCES FOR HYBRID AND MULTICLOUD USING THE USER INTERFACE	22
4.5. CREATING A NEW BUCKET CLASS	25
CHAPTER 5. CONFIGURING NAMESPACE BUCKETS	28
5.1. ADDING PROVIDER CONNECTIONS TO THE MULTICLOUD OBJECT GATEWAY	28
5.2. ADDING NAMESPACE RESOURCES USING THE MULTICLOUD OBJECT GATEWAY	29
5.3. ADDING RESOURCES TO NAMESPACE BUCKETS USING THE MULTICLOUD OBJECT GATEWAY	29
5.4. AMAZON S3 API ENDPOINTS FOR OBJECTS IN NAMESPACE BUCKETS	30
CHAPTER 6. MIRRORING DATA FOR HYBRID AND MULTICLOUD BUCKETS	32
6.1. CREATING BUCKET CLASSES TO MIRROR DATA USING THE MCG COMMAND-LINE-INTERFACE	32
6.2. CREATING BUCKET CLASSES TO MIRROR DATA USING A YAML	32
6.3. CONFIGURING BUCKETS TO MIRROR DATA USING THE USER INTERFACE	33
CHAPTER 7. BUCKET POLICIES IN THE MULTICLOUD OBJECT GATEWAY	36
7.1. ABOUT BUCKET POLICIES	36
7.2. USING BUCKET POLICIES	36
7.3. CREATING AN AWS S3 USER IN THE MULTICLOUD OBJECT GATEWAY	37
CHAPTER 8. OBJECT BUCKET CLAIM	41
8.1. DYNAMIC OBJECT BUCKET CLAIM	41
8.2. CREATING AN OBJECT BUCKET CLAIM USING THE COMMAND LINE INTERFACE	43
8.3. CREATING AN OBJECT BUCKET CLAIM USING THE OPENSIFT WEB CONSOLE	45
8.4. ATTACHING AN OBJECT BUCKET CLAIM TO A DEPLOYMENT	48
8.5. VIEWING OBJECT BUCKETS USING THE OPENSIFT WEB CONSOLE	49
8.6. DELETING OBJECT BUCKET CLAIMS	50
CHAPTER 9. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE BY ADDING ENDPOINTS ...	52
9.1. S3 ENDPOINTS IN THE MULTICLOUD OBJECT GATEWAY	52
9.2. SCALING WITH STORAGE NODES	52
CHAPTER 10. AUTOMATIC SCALING OF MULTICLOUD OBJECT GATEWAY ENDPOINTS	56
CHAPTER 11. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT	57

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 11, 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

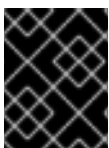
Example 2.1. Example

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 endpoints

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   : HKLbH1rSuVU0l/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.

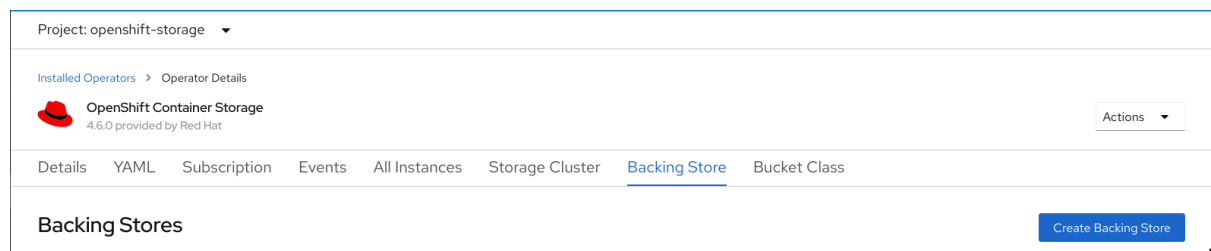
Prerequisites

- Administrator access to OpenShift.

Procedure

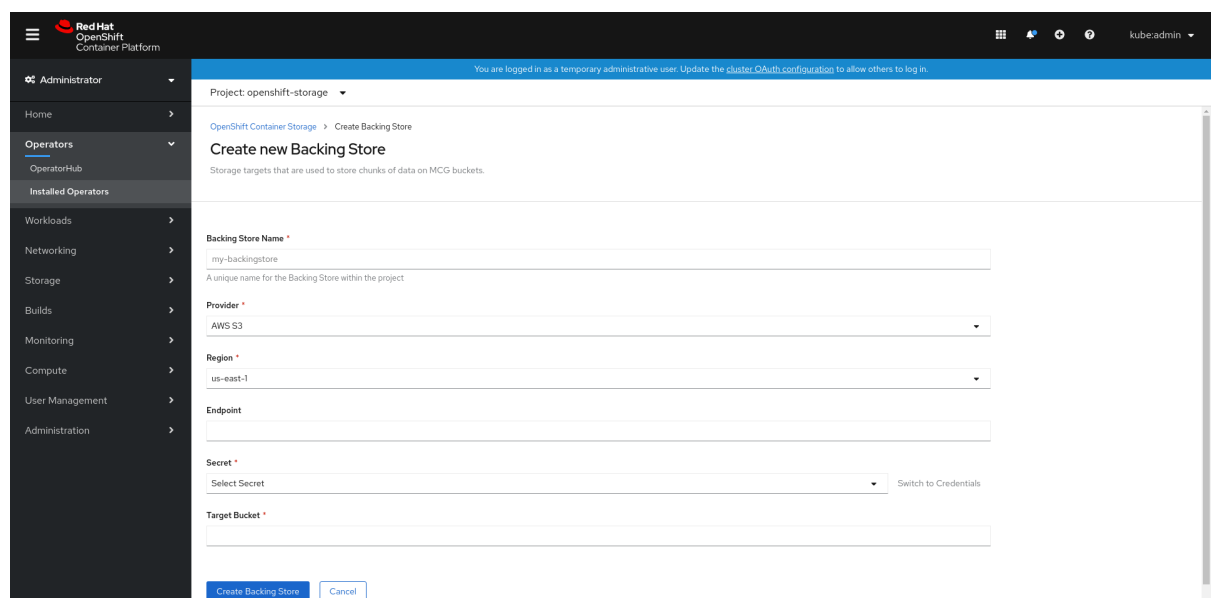
1. Click **Operators** → **Installed Operators** from the left pane of the OpenShift Web Console to view the installed operators.
2. Click **OpenShift Container Storage Operator**.
3. On the OpenShift Container Storage Operator page, scroll right and click the **Backing Store** tab.

Figure 4.1. OpenShift Container Storage Operator page with backing store tab



4. Click **Create Backing Store**.

Figure 4.2. Create Backing Store page



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>
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: noobaa
    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>
```

- 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
```

- 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.
- 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.
- 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

- 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>
```

- Replace **<backingstore_name>** with the name of the backingstore.
- Replace **<AZURE ACCOUNT KEY>** and **<AZURE ACCOUNT NAME>** with an AZURE account key and account name you created for this purpose.
- 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.
- 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.
- 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

```

- 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:

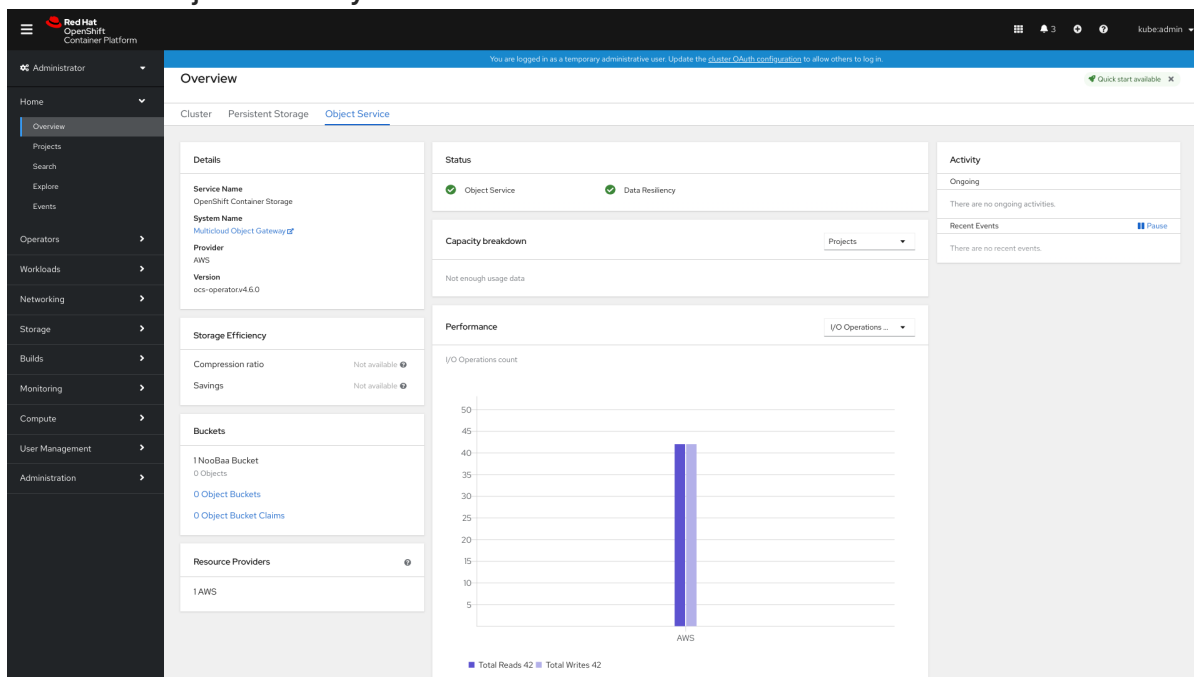
```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
    - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <backingstore-name>
  namespace: openshift-storage
spec:
  s3Compatible:
    endpoint: <RGW endpoint>
    secret:
      name: <backingstore-secret-name>
      namespace: 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](#).

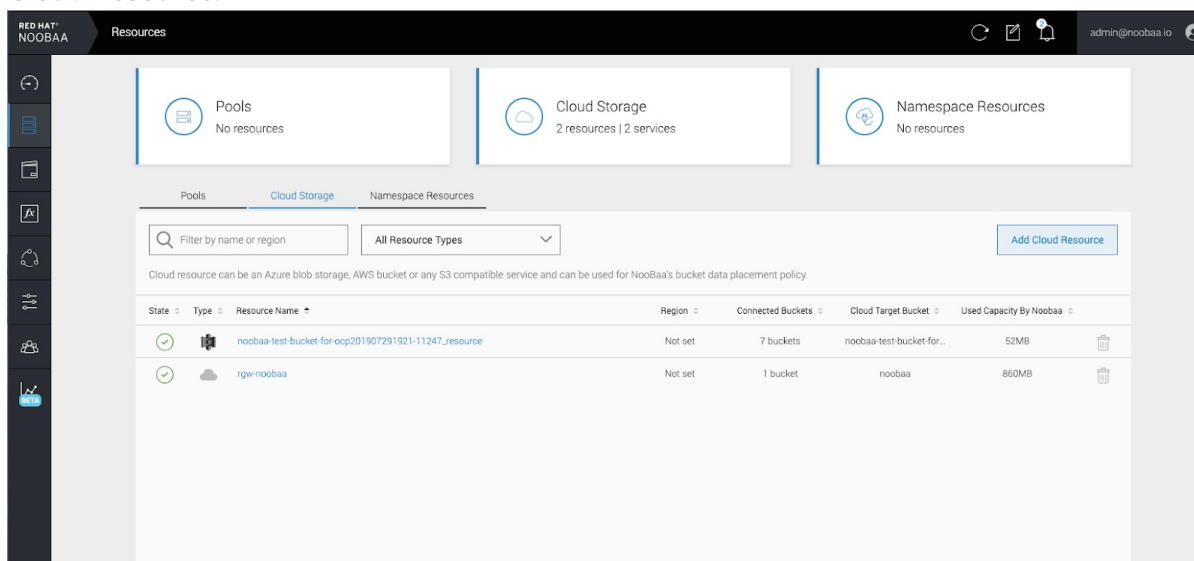
4.4. ADDING STORAGE RESOURCES FOR HYBRID AND MULTICLOUD USING THE USER INTERFACE

Procedure

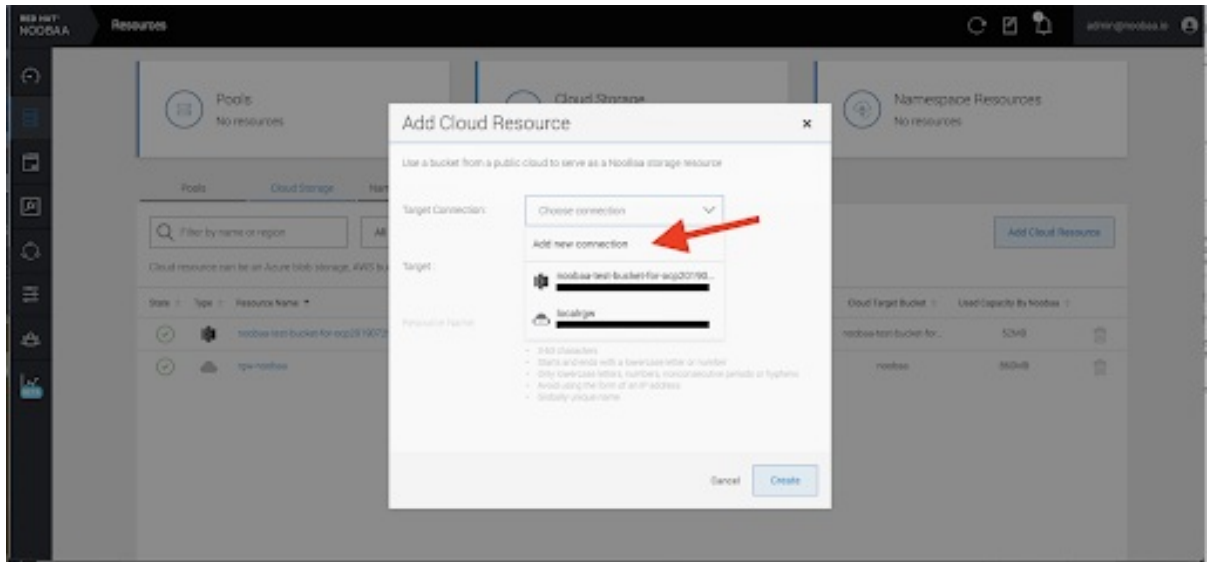
1. In your OpenShift Storage console, navigate to **Overview** → **Object Service** → select the **Multicloud Object Gateway** link:



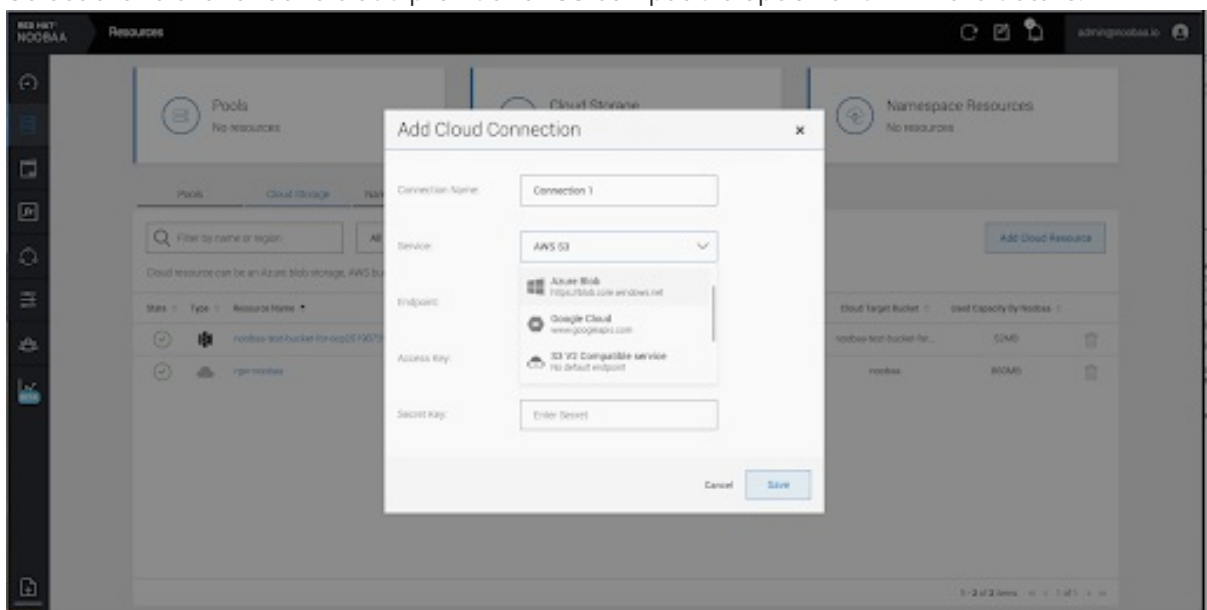
2. Select the **Resources** tab in the left, highlighted below. From the list that populates, select **Add Cloud Resource**:



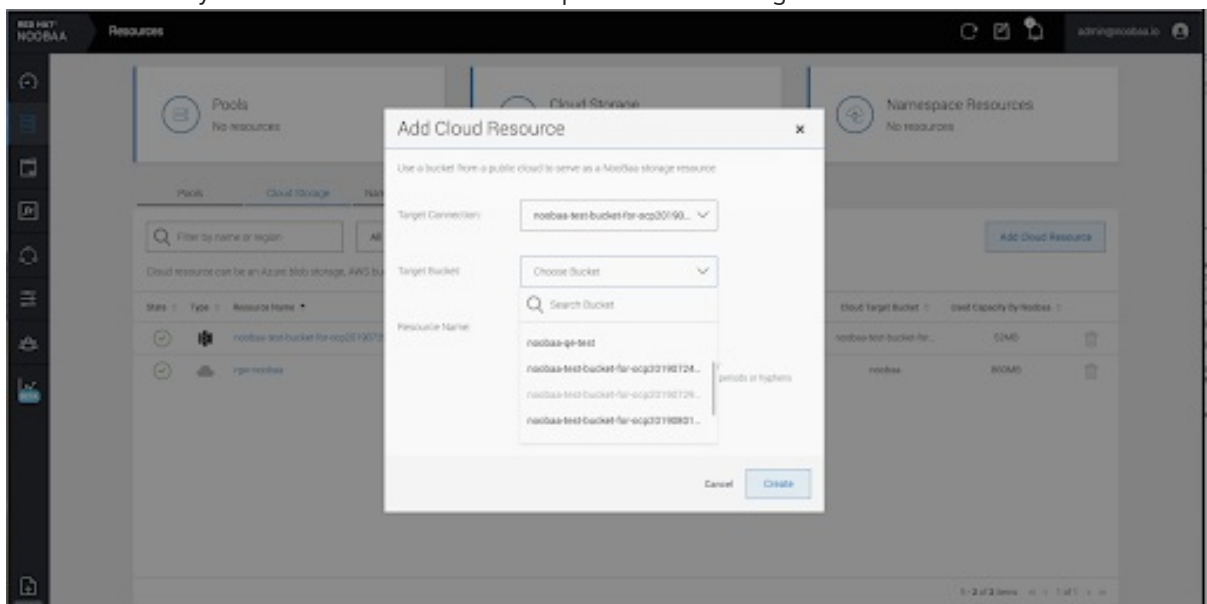
3. Select **Add new connection**:



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



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



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

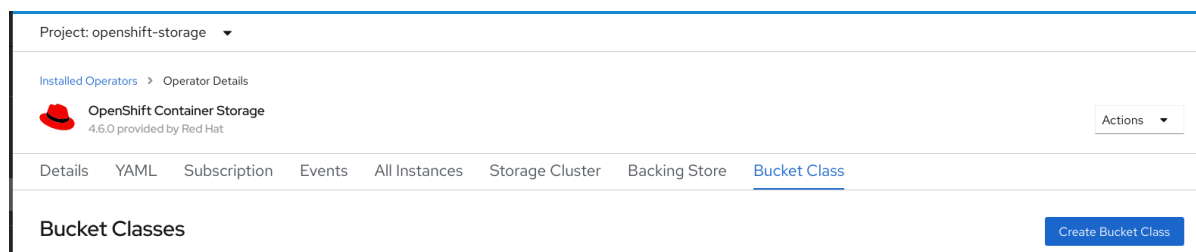
Bucket class is a CRD representing a class of buckets that defines tiering policies and data placements for an Object Bucket Class (OBC).

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

Procedure

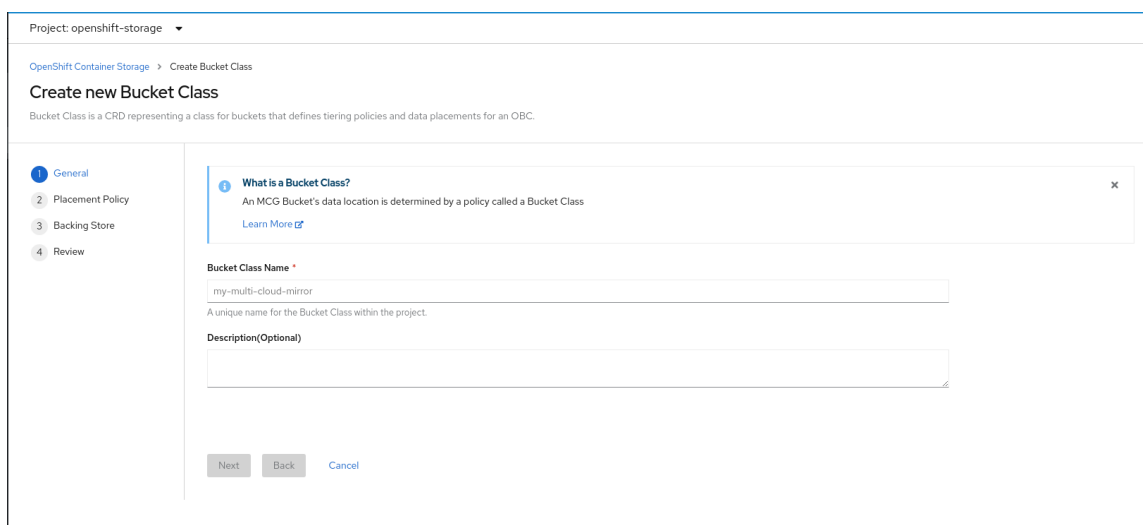
1. Click **Operators** → **Installed Operators** from the left pane of the OpenShift Web Console to view the installed operators.
2. Click **OpenShift Container Storage Operator**.
3. On the OpenShift Container Storage Operator page, scroll right and click the **Bucket Class** tab.

Figure 4.3. OpenShift Container Storage Operator page with 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**.

Figure 4.4. Create Bucket Class page



- 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.

Figure 4.5. Tier 1 - Policy Type selection page

Project: openshift-storage

OpenShift Container Storage > Create Bucket Class

Create new Bucket Class

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

- 1 General
- 2 Placement Policy
- 3 Backing Store
- 4 Review

Tier 1 - Policy Type

- Spread
Spreading the data across the chosen resources. By default, a replica of one copy is used and does not include failure tolerance in case of resource failure.
- Mirror
Full duplication of the data in each chosen resource. By default, a replica of one copy per location is used. Includes failure tolerance in case of resource failure.
- [Add Tier](#)

[Next](#) [Back](#) [Cancel](#)

- 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](#).

Figure 4.6. Tier 1 - Backing Store selection page

Project: openshift-storage

OpenShift Container Storage > Create Bucket Class

Create new Bucket Class

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

- 1 General
- 2 Placement Policy
- 3 Backing Store
- 4 Review

Tier 1 - Backing Store (Spread)

[Create Backing Store](#)

Select at least 1 Backing Store resource *

Search Backing Store

Name	BucketName	Type	Region
<input checked="" type="checkbox"/> noobaa-default-backing-store	nb.1589272586147.apps.ebondare-dc25.q...	awsS3	us-east-2

1 resources selected

[Next](#) [Back](#) [Cancel](#)

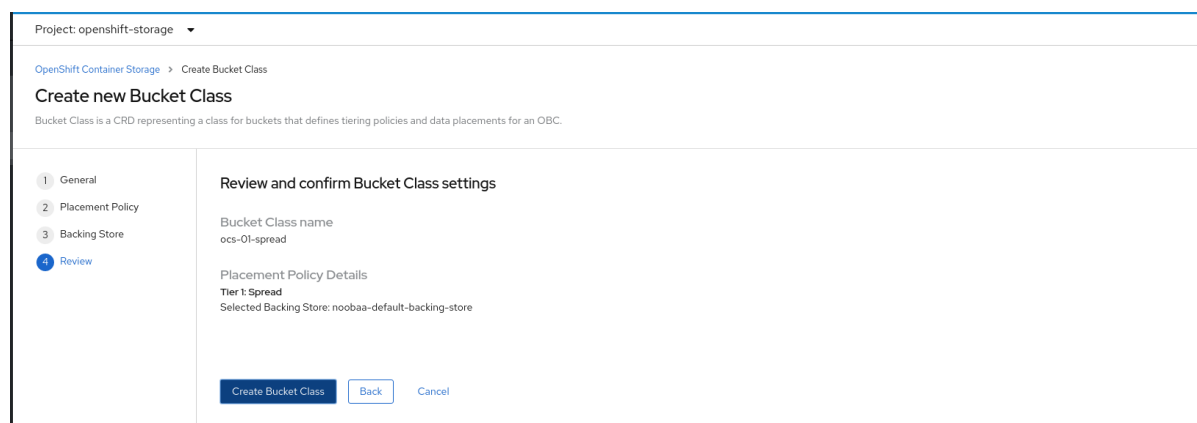


NOTE

You need to select at least 2 backing stores when you select Policy Type as Mirror in previous step.

- a. Review and confirm Bucket Class settings.

Figure 4.7. Bucket class settings review page



- b. 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.

CHAPTER 5. CONFIGURING 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.

1. [Connect your providers to the Multicloud Object Gateway](#) .
2. [Create a namespace resource for each of your providers](#) so they can be added to a namespace bucket.
3. [Add your namespace resources to a namespace bucket](#) and configure the bucket to read from and write to the appropriate namespace resources.

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. ADDING PROVIDER CONNECTIONS TO THE MULTICLOUD OBJECT GATEWAY

You need to add connections for each of your providers so that the Multicloud Object Gateway has access to the provider.

Prerequisites

- Administrative access to the OpenShift Console.

Procedure

1. In the OpenShift Console, click **Home** → **Overview** and click the **Object Service** tab.
2. Click **Multicloud Object Gateway** and log in if prompted.
3. Click **Accounts** and select an account to add the connection to.
4. Click **My Connections**.
5. Click **Add Connection**.
 - a. Enter a **Connection Name**.
 - b. Your cloud provider is shown in the **Service** dropdown by default. Change the selection to use a different provider.
 - c. Your cloud provider's default endpoint is shown in the **Endpoint** field by default. Enter an alternative endpoint if required.
 - d. Enter your **Access Key** for this cloud provider.

- e. Enter your **Secret Key** for this cloud provider.
- f. Click **Save**.

5.2. ADDING NAMESPACE RESOURCES USING THE MULTICLOUD OBJECT GATEWAY

Add existing storage to Multicloud Storage Gateway as namespace resources so that they can be included in namespace buckets for a unified view of existing storage targets, such as Amazon Web Services S3 buckets, Microsoft Azure blobs, and IBM Cloud Object Storage buckets.

Prerequisites

- Administrative access to the OpenShift Console.
- Target connections (providers) are already added to the Multicloud Object Gateway. See [Section 5.1, “Adding provider connections to the Multicloud Object Gateway”](#) for details.

Procedure

1. In the OpenShift Console, click **Home** → **Overview** and click on the **Object Service** tab.
2. Click **Multicloud Storage Gateway** and log in if prompted.
3. Click **Resources**, and click the **Namespace Resources** tab.
4. Click **Create Namespace Resource**
 - a. In **Target Connection**, select the connection to be used for this namespace’s storage provider.
If you need to add a new connection, click **Add New Connection** and enter your provider details; see [Section 5.1, “Adding provider connections to the Multicloud Object Gateway”](#) for more information.
 - b. In **Target Bucket**, select the name of the bucket to use as a target.
 - c. Enter a **Resource Name** for your namespace resource.
 - d. Click **Create**.

Verification

- Verify that the new resource is listed with a green check mark in the **State** column, and 0 buckets in the **Connected Namespace Buckets** column.

5.3. ADDING RESOURCES TO NAMESPACE BUCKETS USING THE MULTICLOUD OBJECT GATEWAY

Add namespace resources to namespace buckets for a unified view of your storage across various providers. You can also configure read and write behaviour so that only one provider accepts new data, while all providers allow existing data to be read.

Prerequisites

- Ensure that all namespace resources you want to handle in a bucket have been added to the Multicloud Object Gateway: [Adding namespace resources using the Multicloud Object Gateway](#).

Procedure

1. In the OpenShift Console, click **Home** → **Overview** and click the **Object Service** tab.
2. Click **Multicloud Object Gateway** and log in if prompted.
3. Click **Buckets**, and click on the **Namespace Buckets** tab.
4. Click **Create Namespace Bucket**
 - a. On the **Choose Name** tab, specify a **Name** for the namespace bucket and click **Next**.
 - b. On the **Set Placement** tab:
 - i. Under **Read Policy**, select the checkbox for each namespace resource that the namespace bucket should read data from.
 - ii. Under **Write Policy**, specify which namespace resource the namespace bucket should write data to.
 - iii. Click **Next**.
 - c. Do not make changes on the **Set Caching Policy** tab in a production environment. This tab is provided as a Development Preview and is subject to support limitations.
 - d. Click **Create**.

Verification

- Verify that the namespace bucket is listed with a green check mark in the **State** column, the expected number of read resources, and the expected write resource name.

5.4. 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 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](#)

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 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. This YAML is a hybrid example that mirrors data between local Ceph storage and AWS:

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  name: hybrid-class
  labels:
    app: noobaa
spec:
  placementPolicy:
    tiers:
      - tier:
          mirrors:
```

- mirror:
 - spread:
 - cos-east-us
- mirror:
 - spread:
 - noobaa-test-bucket-for-ocp201907291921-11247_resource

- 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](#).

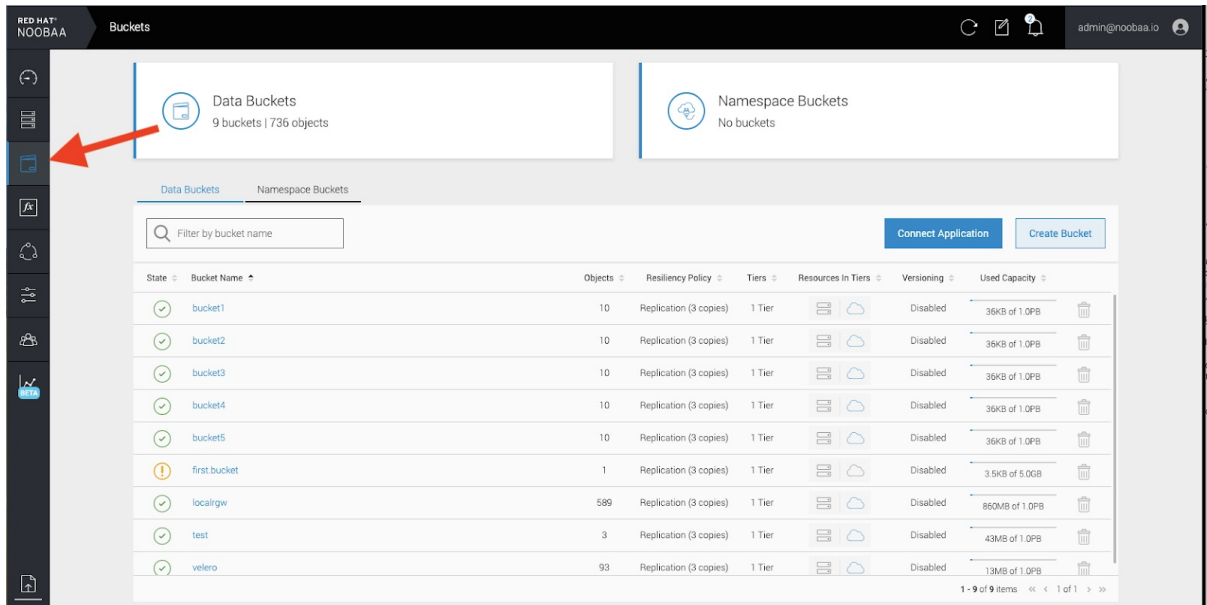
6.3. CONFIGURING BUCKETS TO MIRROR DATA USING THE USER INTERFACE

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

The screenshot shows the OpenShift Storage console interface. The left sidebar contains navigation options like Administrator, Home, Overview, Projects, Search, Explore, Events, Operators, Workloads, Networking, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area is titled 'Overview' and is divided into several sections:

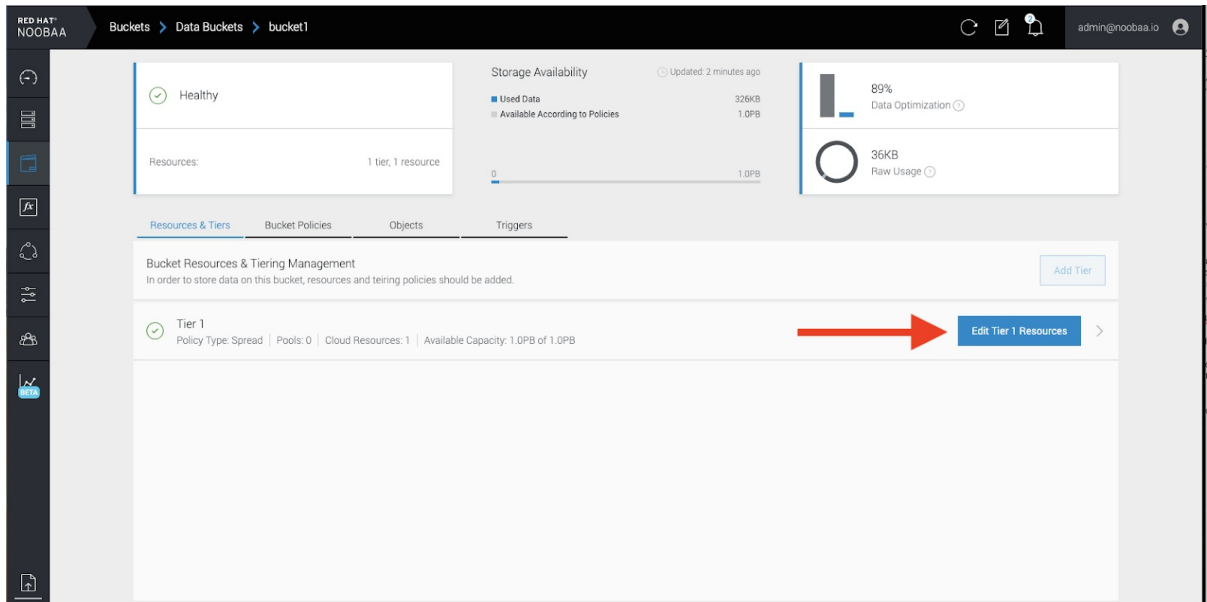
- Details:** Service Name: OpenShift Container Storage; System Name: Multicloud Object Gateway; Provider: AWS; Version: ocp-operatorv4.6.0.
- Status:** Object Service and Data Resiliency are both shown with green checkmarks.
- Capacity breakdown:** Shows 'Projects' as the selected category with the note 'Not enough usage data'.
- Performance:** Shows 'I/O Operations' as the selected category. A bar chart displays 'Total Reads 42' and 'Total Writes 42' for the 'AWS' provider.
- Activity:** Shows 'Ongoing' activities and 'Recent Events'.

- Click the **buckets** icon on the left side. You will see a list of your buckets:

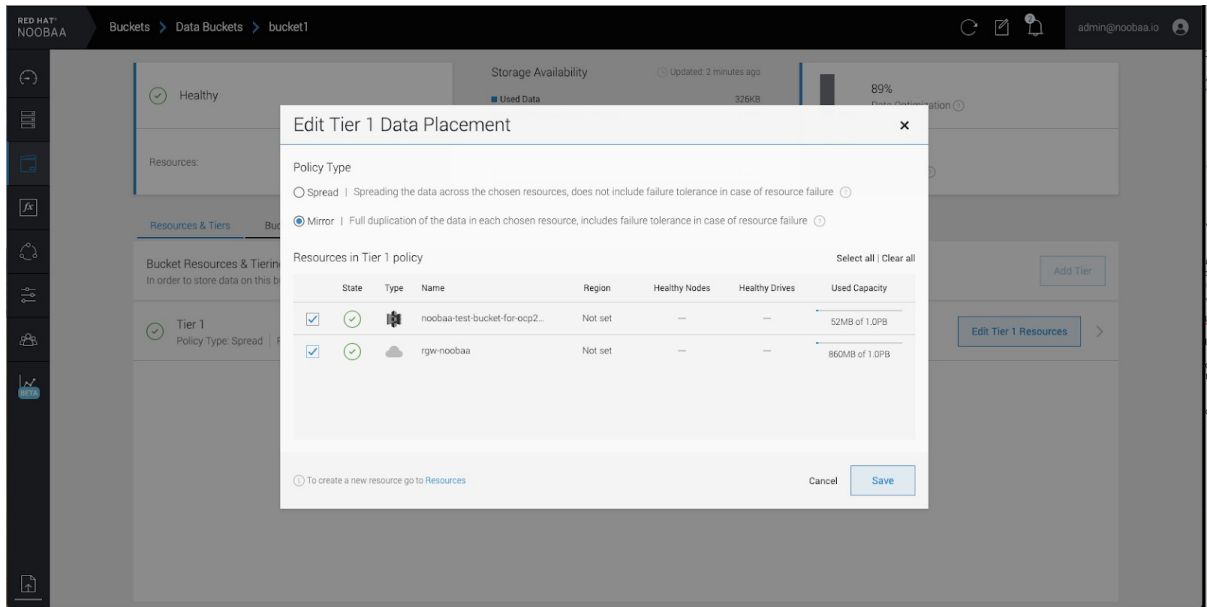


3. Click the bucket you want to update.

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



5. Select **Mirror** and check the relevant resources you want to use for this bucket. In the following example, we mirror data between on prem Ceph RGW to AWS:



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. For details on these elements and examples of how they can be used, 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"](#).

- 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.

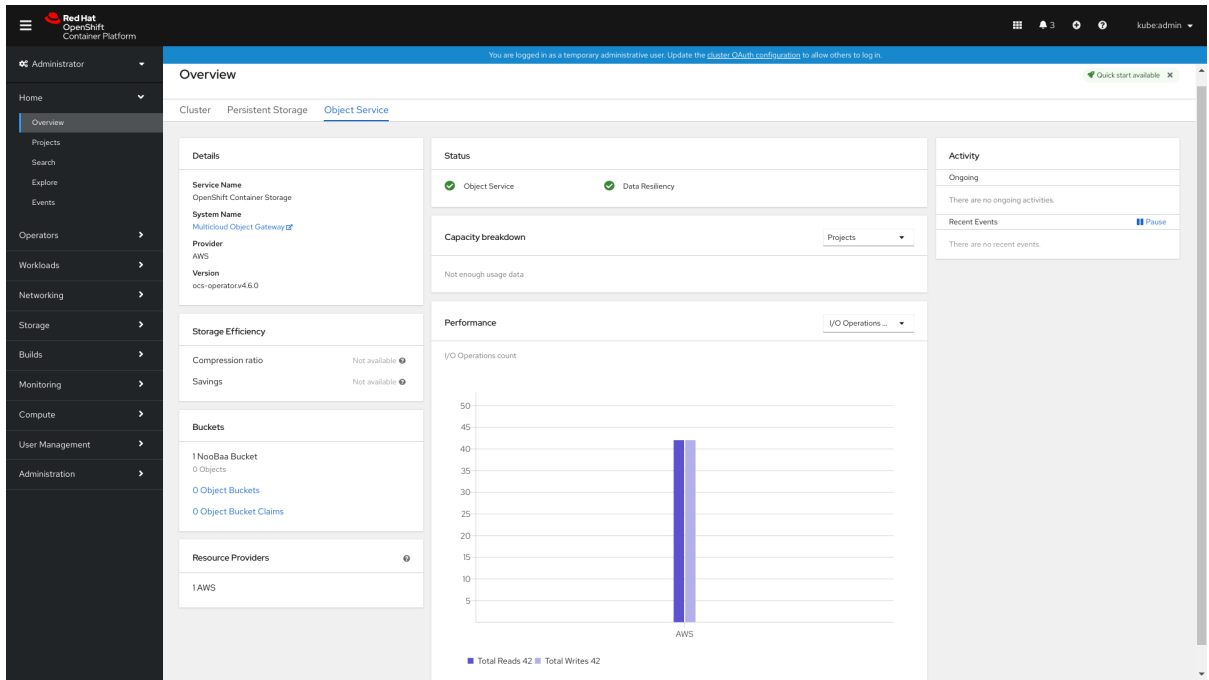
7.3. CREATING AN AWS S3 USER IN THE MULTICLOUD OBJECT GATEWAY

Prerequisites

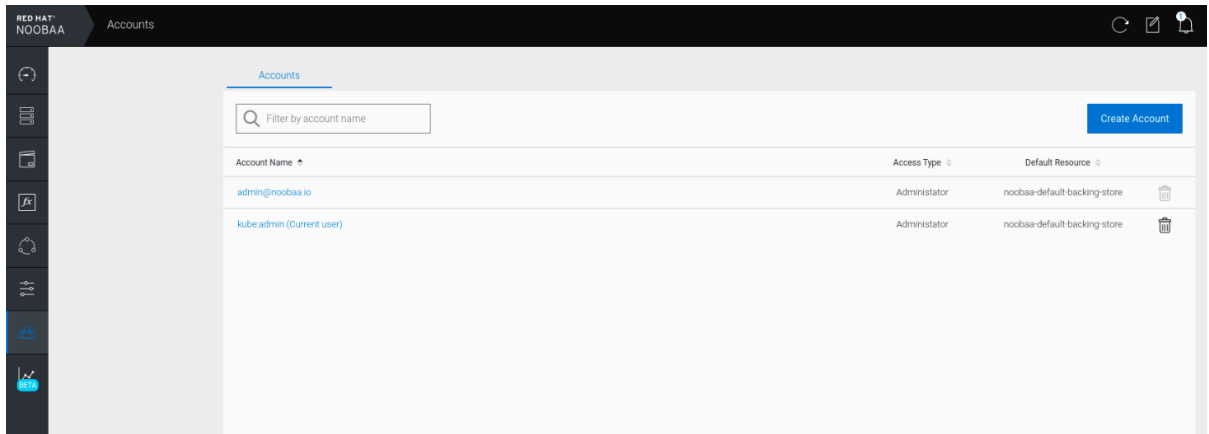
- A running OpenShift Container Storage Platform
- Access to the Multicloud Object Gateway, see [Chapter 2, Accessing the Multicloud Object Gateway with your applications](#)

Procedure

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



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



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:
3 - 32 characters

Cancel

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: ? ▼

Buckets Permissions: ▼

Include any future buckets

Allow new bucket creation: ? Enabled

[Previous](#) [Create](#)

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

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:
Wi9kcFluSWxHRzIiWafIzNk1hc0xma2JXcjM1MVhqa051SIBleXpmOQ==
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.

- 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 OPENSIFT 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**



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

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

If not provided, a generic name will be generated.

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

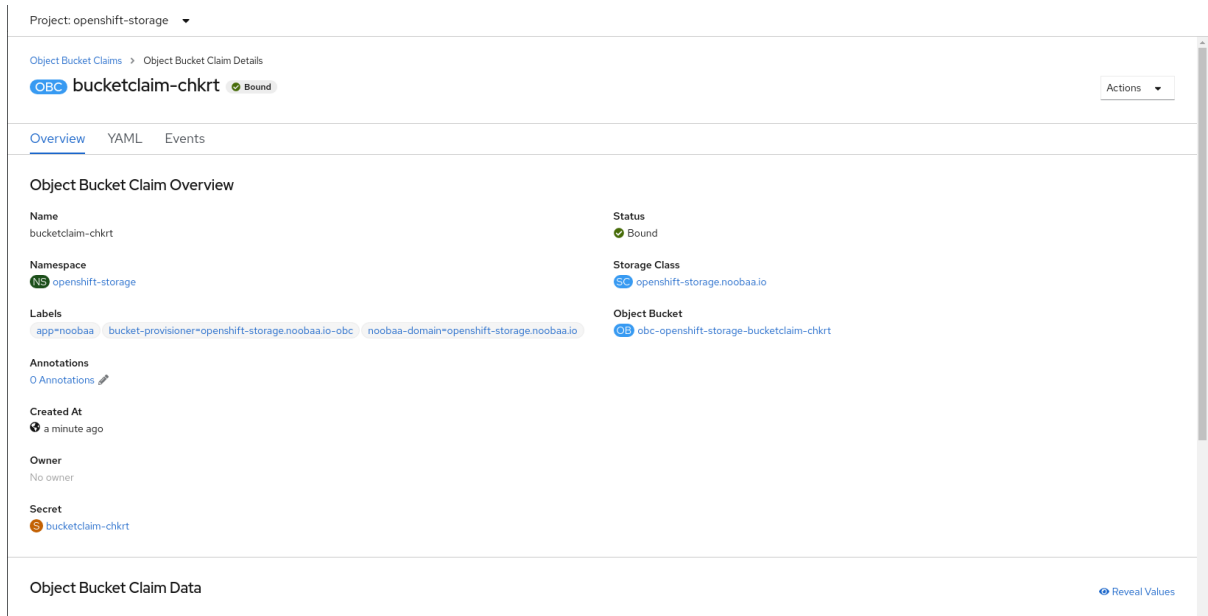


NOTE

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:



Additional Resources

- [Chapter 8, Object Bucket Claim](#)

8.4. ATTACHING AN OBJECT BUCKET CLAIM TO A DEPLOYMENT

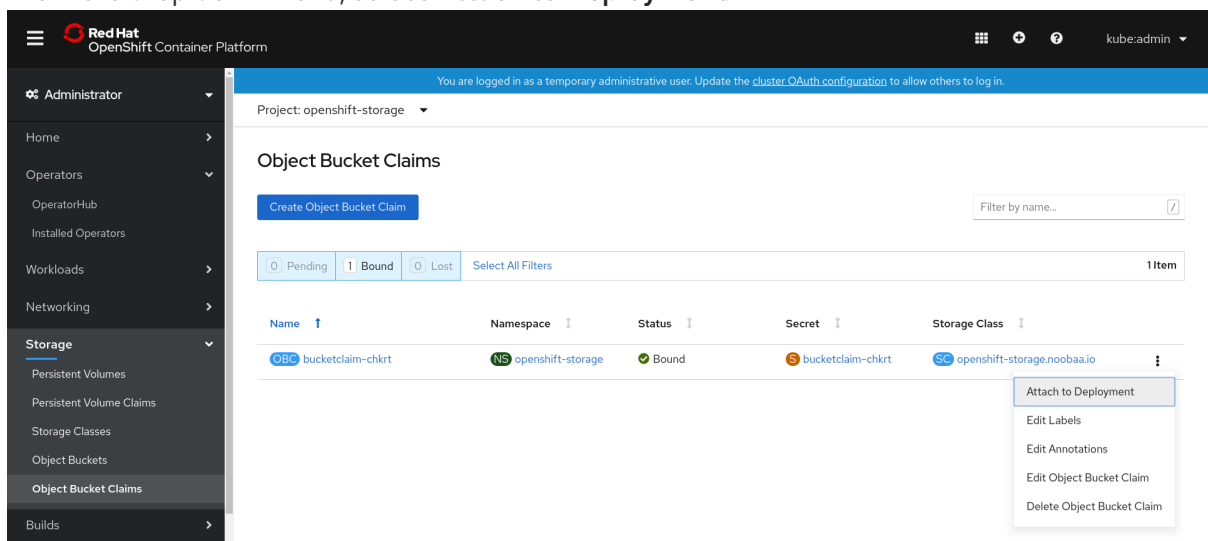
Once created, Object Bucket Claims (OBCs) can be attached to specific deployments.

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 (\vdots) next to the OBC you created.
3. From the drop down menu, select **Attach to Deployment**



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

Attach OBC to a Deployment

Deployment Name *

Cancel
Attach

Additional Resources

- [Chapter 8, Object Bucket Claim](#)

8.5. VIEWING OBJECT BUCKETS USING THE OPENSIFT WEB CONSOLE

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

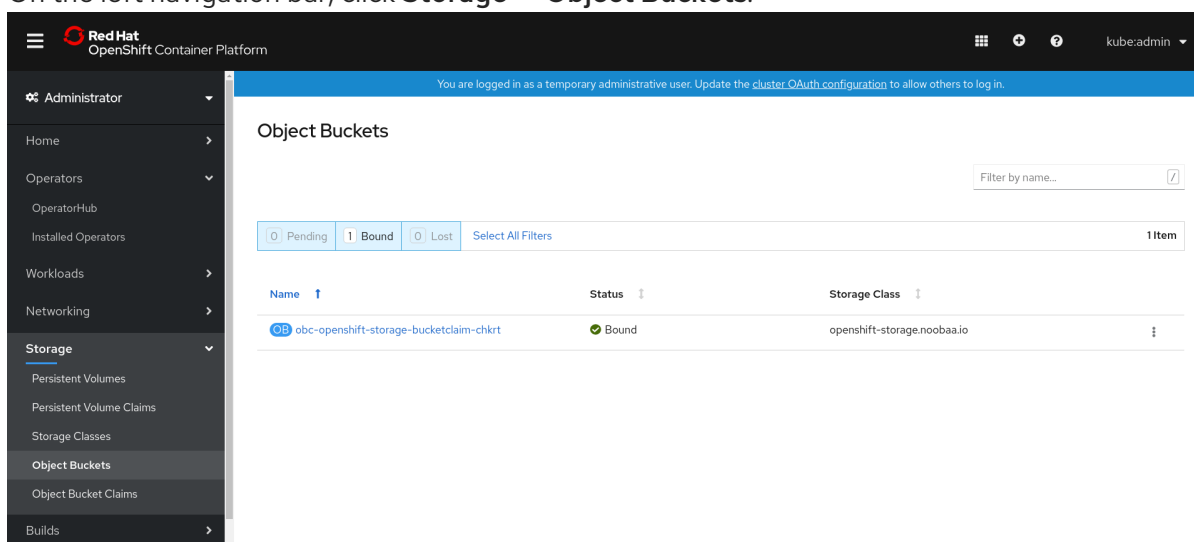
Prerequisites

- 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 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	Status ✔ Bound
Labels app=noobaa bucket-provisioner=openshift-storage.noobaa.io-obc noobaa-domain=openshift-storage.noobaa.io	Storage Class SC openshift-storage.noobaa.io
Annotations 0 Annotations	Object Bucket Claim OBC bucketclaim-chkrt
Created At 🕒 Apr 1, 2:03 pm	
Owner No owner	

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.

Red Hat OpenShift Container Platform

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

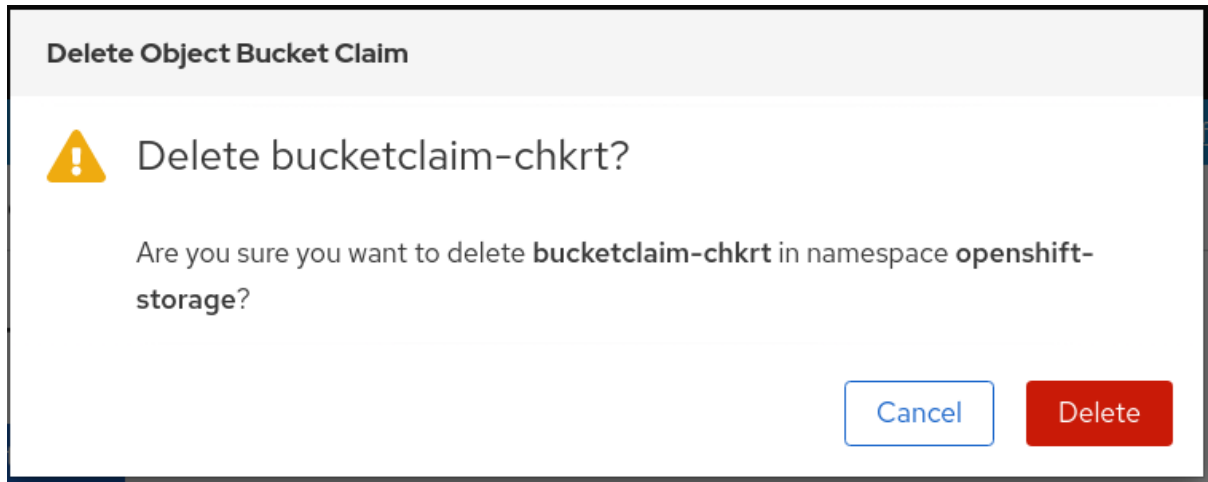
[Create Object Bucket Claim](#)

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

- Attach to Deployment
- Edit Labels
- Edit Annotations
- Edit Object Bucket Claim
- Delete Object Bucket Claim

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



4. Click **Delete**.

Additional Resources

- [Chapter 8, Object Bucket Claim](#)

CHAPTER 9. 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](#).

9.1. 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.

9.2. SCALING WITH STORAGE NODES

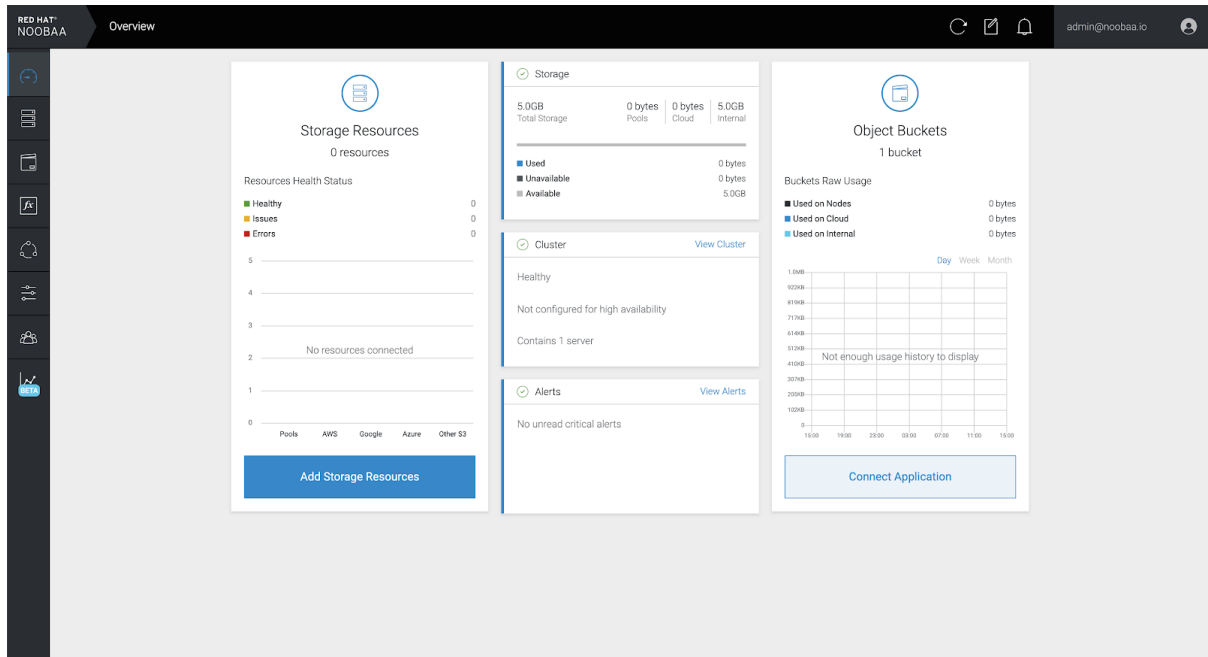
Prerequisites

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

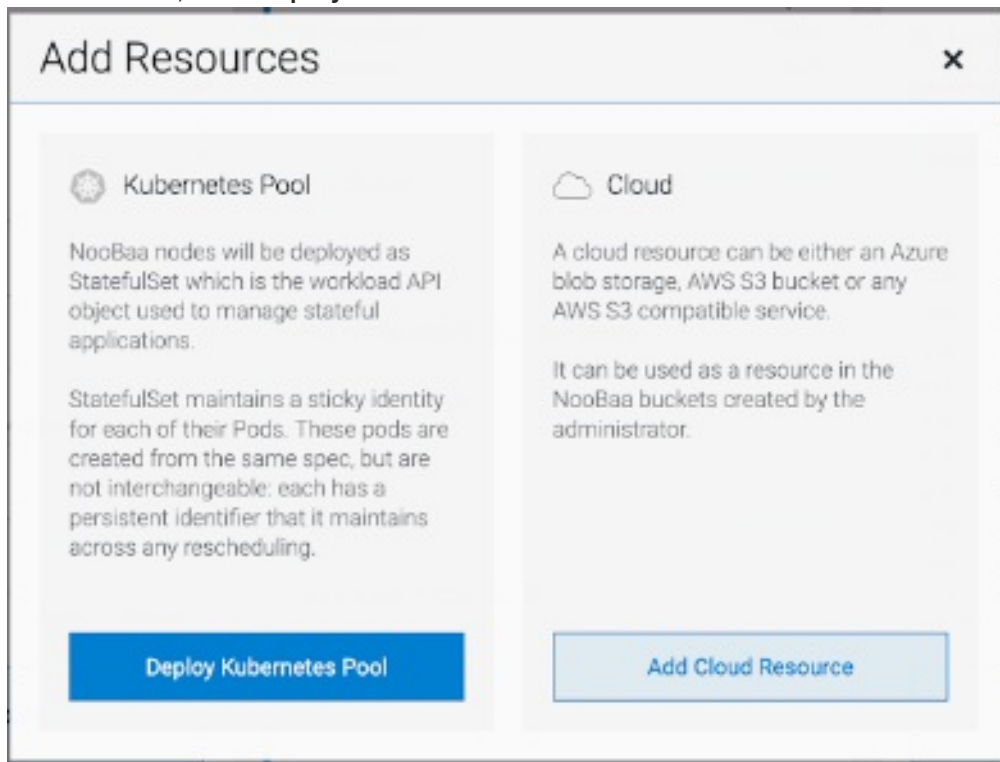
A storage node in the Multicloud Object Gateway is a NooBaa daemon container attached to one or more Persistent Volumes and used for local object service data storage. NooBaa daemons can be deployed on Kubernetes nodes. This can be done by creating a Kubernetes pool consisting of StatefulSet pods.

Procedure

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



- In the window, click **Deploy Kubernetes Pool**



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

Deploy Kubernetes Pool [Close]

1 Create Pool 2 Configure 3 Review

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

Kubernetes Pool Name:

- 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

① If you wish to scale up an existing kubernetes pool go to [Resources > Pools](#)

Cancel **Next**

- 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.

Deploy Kubernetes Pool [Close]

✓ Create Pool 2 Configure 3 Review

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.

Number of Nodes (pods): [Decrease] [Increase]

Node PV Size: [Dropdown]

This cannot be changed later on

① For each new node one PV will be created

Previous **Next**

- 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**:

The screenshot displays the Red Hat NOOBAA Resources management interface. At the top, there are three summary cards: 'Kubernetes pools' with a count of 1, 'Cloud Resources' with a count of 0, and 'Namespace Resources' with a count of 0. Below these, a table shows 'Number of Nodes (Pods)' as 3 and 'Providers' as 0. The main area is titled 'Storage Resources' and features a search bar, a filter dropdown set to 'All Resource Types', and two buttons: 'Deploy Kubernetes Pool' and 'Add Cloud Resource'. A table below lists the resources with columns for State, Type, Resource Name, Region, Connected Buckets, Number of Nodes, and Used Capacity. One resource is listed: 'my.kubernetes.pool-1' with a 'Healthy' status, 'Not set' region, 'None' buckets, 3 nodes, and 6.5GB of 300GB capacity. A pagination bar at the bottom indicates '1 - 1 of 1 items'.

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

CHAPTER 10. AUTOMATIC SCALING OF MULTICLOUD OBJECT GATEWAY ENDPOINTS

As of OpenShift Container Storage version 4.6, the number of MultiCloud Object Gateway endpoints scale automatically when the load on the MultiCloud Object Gateway S3 service increases or decreases. OpenShift Container storage 4.6 clusters are deployed with one active MultiCloud Object Gateway endpoint. Each MultiCloud Object Gateway 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 MultiCloud Object Gateway.

CHAPTER 11. 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

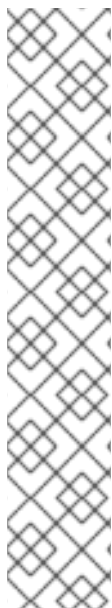
- To verify the **ENDPOINT**, run the following command:

```
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 -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 -o yaml | grep -w "SecretKey:" | head -n1 | awk '{print $2}' | base64 --decode
```