Managing hybrid and multicloud resources

Instructions for how to manage storage resources across a hybrid cloud or multicloud environment using the Multicloud Object Gateway (NooBaa).
Instructions for how to manage storage resources across a hybrid cloud or multicloud environment using the Multicloud Object Gateway (NooBaa).
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  ............... 8
  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 ........... 14

CHAPTER 4. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD ......................... 16
  4.1. CREATING A NEW BACKING STORE .................................................. 16
  4.2. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD USING THE MCG COMMAND LINE INTERFACE .......................... 17
    4.2.1. Creating an AWS-backed backingstore ......................................... 17
    4.2.2. Creating an IBM COS-backed backingstore ................................ 19
    4.2.3. Creating an Azure-backed backingstore ..................................... 21
    4.2.4. Creating a GCP-backed backingstore ......................................... 22
    4.2.5. Creating a local Persistent Volume-backed backingstore ................. 24
  4.3. ADDING STORAGE RESOURCES FOR HYBRID AND MULTICLOUD USING THE USER INTERFACE ....... 27
  4.4. CREATING AN S3 COMPATIBLE MULTICLOUD OBJECT GATEWAY BACKINGSTORE ............... 26

CHAPTER 5. MANAGING NAMESPACE BUCKETS .................................................. 32
  5.1. AMAZON S3 API ENDPOINTS FOR OBJECTS IN NAMESPACE BUCKETS ....................... 32
  5.2. ADDING A NAMESPACE BUCKET USING THE MULTICLOUD OBJECT GATEWAY CLI AND YAML ........................................................................ 33
    5.2.1. Adding an AWS S3 namespace bucket using YAML .............................. 33
    5.2.2. Adding an IBM COS namespace bucket using YAML ......................... 35
    5.2.3. Adding an AWS S3 namespace bucket using the Multicloud Object Gateway CLI ................. 38
    5.2.4. Adding an IBM COS namespace bucket using the Multicloud Object Gateway CLI .................. 40
  5.3. ADDING A NAMESPACE BUCKET USING THE OPENSF...
CHAPTER 9. OBJECT BUCKET CLAIM ................................................................. 56
  9.1. DYNAMIC OBJECT BUCKET CLAIM .................................................... 56
  9.2. CREATING AN OBJECT BUCKET CLAIM USING THE COMMAND LINE INTERFACE ..... 58
  9.3. CREATING AN OBJECT BUCKET CLAIM USING THE OPENSHIFT WEB CONSOLE ........ 61
  9.4. ATTACHING AN OBJECT BUCKET CLAIM TO A DEPLOYMENT ......................... 63
  9.5. VIEWING OBJECT BUCKETS USING THE OPENSHIFT WEB CONSOLE ................ 64
  9.6. DELETING OBJECT BUCKET CLAIMS .................................................... 65

CHAPTER 10. CACHING POLICY FOR OBJECT BUCKETS .................................. 67
  10.1. CREATING AN AWS CACHE BUCKET ....................................................... 67
  10.2. CREATING AN IBM COS CACHE BUCKET ................................................. 69

CHAPTER 11. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE BY ADDING ENDPOINTS ... 72
  11.1. AUTOMATIC SCALING OF MULTICLOUD OBJECT GATEWAY ENDPOINTS ........... 72
  11.2. SCALING THE MULTICLOUD OBJECT GATEWAY WITH STORAGE NODES ............. 72

CHAPTER 12. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT ....................... 74
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. In the **Component** section, choose **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 Multicloud Object Gateway (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 (RGW) S3 endpoint, see Accessing the RADOS Object Gateway S3 endpoint.

Prerequisites

- A running OpenShift Data Foundation Platform.
- Download the MCG command-line interface for easier management.

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

**NOTE**

Specify the appropriate architecture for enabling the repositories using the subscription manager.

- For IBM Power, use the following command:

```
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
```
- For IBM Z infrastructure, use the following command:

```
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
```

- Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found at Download RedHat OpenShift Data Foundation page.

**NOTE**

Choose the correct Product Variant according to your architecture.

You can access the relevant endpoint, access key, and secret access key in 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”**

**Example 2.1. Example**

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

If the client application tries to access https://<bucket-name>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 Multicloud Object Gateway (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:
  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!
```

---

Red Hat OpenShift Data Foundation 4.9 Managing hybrid and multicloud resources
Welcome to NooBaa!
-----------------
NooBaa Core Version:
NooBaa Operator Version:

Let's get started:

1. Connect to Management console:

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

```bash
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:

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

2. Test S3 client:

```bash
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:
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:
### 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-odf-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

**NOTE**

Specify the appropriate architecture for enabling the repositories using the subscription manager.

- For IBM Power, use the following command:

  ```
  # subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
  ```

- For IBM Z infrastructure, use the following command:

  ```
  # subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
  ```

#### 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] CRD Status:
INFO[0000] Exists: CustomResourceDefinition "noobaas.noobaa.io"
INFO[0000] Exists: CustomResourceDefinition "backingstores.noobaa.io"
INFO[0000] Exists: CustomResourceDefinition "bucketclasses.noobaa.io"
INFO[0004] Exists: CustomResourceDefinition "objectbucketclaims.objectbucket.io"
INFO[0004] Exists: CustomResourceDefinition "objectbuckets.objectbucket.io"
```
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 -#
#------------------#

e-mail : admin@noobaa.io
password : HKLbH1rSuVU0I/souIkSiA==

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

ExternalDNS : [https://s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
https://a340f4e1315be11eaa3b70683061451e-943168195.us-east-2.elb.amazonaws.com:443]
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 the buckets in OpenShift Data Foundation:

```
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 (MCG) 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 Data Foundation Platform.

Procedure

1. Enable access to the MCG 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 MCG 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 Data Foundation 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 Storage → OpenShift Data Foundation.

3. In the Storage Systems tab, select the storage system and then click Overview → Object tab.

4. Select the Multicloud Object Gateway link.

5. 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 Data Foundation.

**Prerequisites**

- Administrator access to OpenShift Data Foundation.

**Procedure**

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.
2. Click the **Backing Store** tab.
3. Click **Create Backing Store**.
4. 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 the 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 the **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 the MCG that it can use this bucket for the system.
5. Click **Create Backing Store**.

**Verification steps**

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.
2. Click the **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 the cloud provider and clusters.

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.

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

**NOTE**

Specify the appropriate architecture for enabling the repositories using the subscription manager. For instance, in case of IBM Z infrastructure use the following command:

```bash
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
```

- Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found here `https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages`

**NOTE**

Choose the correct Product Variant according to your architecture.
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 the MCG 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:

   ```yaml
   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:

   ```yaml
   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 the MCG 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-odf-4-for-rhel-8-x86_64-rpms
  # yum install mcg
  ```

  **NOTE**
  Specify the appropriate architecture for enabling the repositories using the subscription manager. For instance,

  - For IBM Power, use the following command:

    ```
    # subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
    ```

  - For IBM Z infrastructure, use the following command:

    ```
    # subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
    ```

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

  **NOTE**
  Choose the correct Product Variant according to your architecture.

**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 the MCG 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:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
  namespace: openshift-storage
  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:

```yaml
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 the MCG 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.

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

  **NOTE**

  Specify the appropriate architecture for enabling the repositories using the subscription manager. For instance, in case of IBM Z infrastructure use the following command:

  ```bash
  # subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
  ```

- Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found here [https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages](https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages)

  **NOTE**

  Choose the correct Product Variant according to your architecture.

#### Procedure

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

   ```bash
   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 the MCG 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:
You can also add storage resources using a YAML:

1. Create a secret with the credentials:

```yaml
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:

```yaml
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 the MCG 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-odf-4-for-rhel-8-x86_64-rpms
# yum install mcg

**NOTE**

Specify the appropriate architecture for enabling the repositories using the subscription manager. For instance, in case of IBM Z infrastructure use the following command:

```bash
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
```

- Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found here [https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages](https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages)

**NOTE**

Choose the correct Product Variant according to your architecture.

**Procedure**

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

   ```bash
   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 the MCG 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:

   ```text
   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:

   ```json
   apiVersion: v1
cr kind: Secret
cr metadata:
      name: <backingstore-secret-name>
cr type: Opaque
cr 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:

```yaml
apiVersion: noobaa.io/v1alpha1
def: 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 the MCG 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.

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

  **NOTE**

  Specify the appropriate architecture for enabling the repositories using the subscription manager. For instance, in case of IBM Z infrastructure use the following command:

  ```bash
  # subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
  ```

- Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found here [https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages](https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages)
**NOTE**
Choose the correct Product Variant according to your architecture.

**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.
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 (MCG) can use any S3 compatible object storage as a backing store, for example, Red Hat Ceph Storage’s RADOS Object Gateway (RGW). The following procedure shows how to create an S3 compatible MCG backing store for Red Hat Ceph Storage’s RGW. Note that when the RGW is deployed, OpenShift Data Foundation operator creates an S3 compatible backingstore for MCG automatically.

Procedure

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

   ```yaml
   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 the MCG 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 the OpenShift Web Console, click Storage → OpenShift Data Foundation.

2. In the Storage Systems tab, select the storage system and then click Overview → Object tab.

3. Select the Multicloud Object Gateway link.

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

   2. Select Add new connection.
3. Select the relevant native cloud provider or S3 compatible option and fill in the details.

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

5. 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 Data Foundation.

Procedure

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.
2. Click the **Bucket Class** tab.
3. Click **Create Bucket Class**.
4. On the Create new Bucket Class page, perform the following:
   a. Select the bucket class type and enter a bucket class name.
      i. Select the **BucketClass type**. Choose one of the following options:
         • **Standard**: data will be consumed by a Multicloud Object Gateway (MCG), deduped, compressed and encrypted.
         • **Namespace**: data is stored on the NamespaceStores without performing de-duplication, compression or encryption. By default, **Standard** is selected.
      ii. Enter a **Bucket Class Name**.
      iii. 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.
   d. Review and confirm Bucket Class settings.

NOTE

You need to select at least 2 backing stores when you select Policy Type as **Mirror** in previous step.
e. Click **Create Bucket Class**

**Verification steps**

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.
2. Click the **Bucket Class** tab and search the new Bucket Class.

## 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 Web Console.

**Procedure**

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.
2. Click the **Bucket Class** tab.
3. Click the Action Menu (⋮) next to the Bucket class you want to edit.
4. Click **Edit Bucket Class**.
5. 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 (MCG) 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. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.
2. Click the **Bucket Class** tab.
3. Click the Action Menu (⋮) next to the Bucket class you want to edit.
4. Click **Edit Bucket Class Resources**.

5. 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.

6. 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.

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 the namespace buckets using the Amazon Simple Storage Service (S3) API.

Red Hat OpenShift Data Foundation 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.
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 (MCG) 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 Data Foundation Platform
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, Accessing the Multicloud Object Gateway with your applications.

Procedure

1. Create a secret with the credentials:

   ```yaml
   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>`.

   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 MCG namespace buckets. To create a NamespaceStore resource, apply the following YAML:

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

a. Replace `<resource-name>` with the name you want to give to the resource.
b. Replace `<namespacestore-secret-name>` with the secret created in step 1.
c. Replace `<namespace-secret>` with the namespace where the secret can be found.
d. Replace `<target-bucket>` with the target bucket you created for the NamespaceStore.

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:

  ```yaml
  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 the name of a single namespace-store that defines the read and write target of the namespace bucket.

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

  ```yaml
  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 the name of a single namespace-store that defines the write target of the namespace bucket.
- Replace `<read-resources>` with a list of the names of the namespace-stores that defines 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.

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

**NOTE**

For IBM Power and IBM Z infrastructure use `storageClassName` as `openshift-storage.noobaa.io`

a. Replace `<resource-name>` with the name you want to give to the resource.
b. Replace `<my-bucket>` with the name you want to give to the bucket.
c. 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 MCG, and the operator creates a Secret and ConfigMap with the same name and on the same namespace of the OBC.

### 5.2.2. Adding an IBM COS namespace bucket using YAML

**Prerequisites**

- A running OpenShift Data Foundation Platform.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, *Accessing the Multicloud Object Gateway with your applications.*
Procedure

1. Create a secret with the credentials:

```yaml
apiVersion: v1
description: 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 MCG namespace buckets. To create a NamespaceStore resource, apply the following YAML:

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

c. Replace `<namespace-secret>` with the namespace where the secret can be found.

d. Replace `<target-bucket>` with the target bucket you created for the NamespaceStore.

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 the name of a single namespace-store that defines 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 the name of a single namespace-store that defines the write target of the namespace bucket.
- Replace `<read-resources>` with a list of the names of namespace-stores that defines 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: <resource-name>
  namespace: openshift-storage
spec:
  generateBucketName: <my-bucket>
storageClassName: openshift-storage.noobaa.io
additionalConfig:
  bucketclass: <my-bucket-class>

NOTE
For IBM Power and IBM Z infrastructure use storageClassName as openshift-storage.noobaa.io

a. Replace <resource-name> with the name you want to give to the resource.
b. Replace <my-bucket> with the name you want to give to the bucket.
c. 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 MCG, and the operator creates a Secret and ConfigMap with the same name and 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 Data Foundation Platform.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, Accessing the Multicloud Object Gateway with your applications.
- Download the MCG command-line interface:

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

NOTE
Specify the appropriate architecture for enabling the repositories using subscription manager. For instance, in case of IBM Z infrastructure use the following command:

```bash
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
```

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

NOTE
Choose the correct Product Variant according to your architecture.

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 MCG 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 the MCG 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
     ```

     a. Replace <resource-name> with the name you want to give the resource.

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

     c. Replace <resource> with a single namespace-store that defines 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
     ```

     a. Replace <resource-name> with the name you want to give the resource.

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

     c. Replace <write-resource> with a single namespace-store that defines the write target of the namespace bucket.

     d. Replace <read-resources> with a list of namespace-stores separated by commas that defines 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 <bucket-name> with a bucket name of your choice.

   b. 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 MCG, and the operator creates a Secret and ConfigMap with the same name and 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 Data Foundation Platform.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, *Accessing the Multicloud Object Gateway with your applications*.
- Download the MCG command-line interface:

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

  **NOTE**

  Specify the appropriate architecture for enabling the repositories using subscription manager.

  - For IBM Power, use the following command:

    ```bash
    # subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
    ```

  - For IBM Z infrastructure, use the following command:

    ```bash
    # subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
    ```

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

  **NOTE**

  Choose the correct Product Variant according to your architecture.

#### 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 MCG namespace buckets. From the MCG command-line interface, run the following command:

   ```bash
   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 the MCG 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:

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

  - Replace `<resource-name>` with the name you want to give the resource.
  - Replace `<my-bucket-class>` with a unique bucket class name.
  - Replace `<resource>` with a single namespace-store that defines 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:

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

  - Replace `<resource-name>` with the name you want to give the resource.
  - Replace `<my-bucket-class>` with a unique bucket class name.
  - Replace `<write-resource>` with a single namespace-store that defines the write target of the namespace bucket.
  - Replace `<read-resources>` with a list of namespace-stores separated by commas that defines 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.

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

  a. Replace `<bucket-name>` with a bucket name of your choice.
  
b. 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 MCG, and the operator creates a Secret and ConfigMap with the same name and on the same namespace of the OBC.

5.3. ADDING A NAMESPACE BUCKET USING THE OPENSHIFT CONTAINER PLATFORM USER INTERFACE
With the release of OpenShift Data Foundation 4.8, namespace buckets can be added using the OpenShift Container Platform user interface. For more information about namespace buckets, see Managing namespace buckets.

**Prerequisites**

- OpenShift Container Platform with OpenShift Data Foundation operator installed.
- Access to the Multicloud Object Gateway (MCG).

**Procedure**

1. Log into the OpenShift Web Console.
2. Click Storage → OpenShift Data Foundation.
3. Click the **Namespace Store** tab to create a **namespacestore** resources to be used in the namespace bucket.
   a. Click **Create namespace store**
   b. Enter a namespacestore name.
   c. Choose a provider.
   d. Choose a region.
   e. Either select an existing secret, or click **Switch to credentials** to create a secret by entering a secret key and secret access key.
   f. Choose a target bucket.
   g. Click **Create**.
   h. Verify the namespacestore is in the **Ready** state.
   i. Repeat these steps until you have the desired amount of resources.
4. Click the **Bucket Class** tab → **Create a new Bucket Class**
   a. Select the **Namespace** radio button.
   b. Enter a Bucket Class name.
   c. Add a description (optional).
   d. Click **Next**.
5. Choose a namespace policy type for your namespace bucket, and then click **Next**.
6. Select the target resource(s).
   - If your namespace policy type is **Single**, you need to choose a read resource.
   - If your namespace policy type is **Multi**, you need to choose read resources and a write resource.
If your namespace policy type is **Cache**, you need to choose a Hub namespace store that defines the read and write target of the namespace bucket.

7. Click **Next**.

8. Review your new bucket class, and then click **Create Bucketclass**.

9. On the **BucketClass** page, verify that your newly created resource is in the **Created** phase.

10. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.

11. In the **Status** card, click **Storage System** and click the storage system link from the pop up that appears.

12. In the **Object** tab, click **Multicloud Object Gateway → Buckets → Namespace Buckets** tab.

13. 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 created in step 5 that the namespace bucket should read data from.
      ii. If the namespace policy type you are using is **Multi**, then Under **Write Policy**, specify which namespace resource the namespace bucket should write data to.
      iii. Click **Next**.
   c. 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.
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 in 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 Multicloud Object Gateway (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. This YAML is a hybrid example that mirrors data between local Ceph storage and AWS:

   ```
   apiVersion: noobaa.io/v1alpha1
   kind: BucketClass
   metadata:
     labels:
       app: noobaa
       name: <bucket-class-name>
       namespace: openshift-storage
   spec:
     placementPolicy:
       tiers:
         - backingStores:
   ```
2. Add the following lines to your standard Object Bucket Claim (OBC):

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

For more information about OBCs, see Chapter 9, Object Bucket Claim.

### 6.3. CONFIGURING BUCKETS TO MIRROR DATA USING THE USER INTERFACE

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.

2. In the **Status** card, click **Storage System** and click the storage system link from the pop up that appears.

3. In the **Object** tab, click the **Multicloud Object Gateway** link.

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

![Bucket List](image)

5. Click the bucket you want to update.

6. Click **Edit Tier 1 Resources**

![Edit Tier 1 Resources](image)

7. 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:
8. Click Save.

**NOTE**

Resources created in NooBaa UI cannot be used by OpenShift UI or Multicloud Object Gateway (MCG) CLI.
OpenShift Data Foundation 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 Data Foundation Platform.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, *Accessing the Multicloud Object Gateway with your applications*.

**Procedure**

To use bucket policies in the MCG:

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

   ```json
   {  
     "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
```

   a. Replace `ENDPOINT` with the S3 endpoint.

   b. Replace `MyBucket` with the bucket to set the policy on.

   c. Replace `BucketPolicy` with the bucket policy JSON file.

   d. 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 Data Foundation Platform.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, Accessing the Multicloud Object Gateway with your applications

**Procedure**

1. In the OpenShift Web Console, click **Storage → OpenShift Data Foundation**.

2. In the **Status** card, click **Storage System** and click the storage system link from the pop up that appears.

3. In the **Object** tab, click the **Multicloud Object Gateway** link.
4. Under the Accounts tab, click Create Account.

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

6. 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

S3 default placement:  nocbaa-default-backing-store

Buckets Permissions:  All buckets selected

Include any future buckets

Allow new bucket creation:  Enabled

Previous  Create
CHAPTER 8. MULTICLOUD OBJECT GATEWAY BUCKET AND BUCKET CLASS REPLICATION

Data replication between buckets provides higher resiliency and better collaboration options. These buckets can be either data buckets backed by any supported storage solution (S3, Azure, and so on), or namespace buckets (where PV Pool and GCP are not supported).

- For more information on how to create a backingstore, see Adding storage resources for hybrid or Multicloud using the MCG command line interface.
- For more information on how to create a namespacestore, see Adding a namespace bucket using the Multicloud Object Gateway CLI and YAML.

A bucket replication policy is composed of a list of replication rules. Each rule defines the destination bucket, and can specify a filter based on an object key prefix. Configuring a complementing replication policy on a second bucket results in bi-directional replication.

Prerequisites

- A running OpenShift Data Foundation Platform.
- Access to the Multicloud Object Gateway (MCG), see Accessing the Multicloud Object Gateway with your applications.
- Download the MCG command-line interface.

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

**IMPORTANT**

Specify the appropriate architecture for enabling the repositories using the subscription manager. For instance, in case of IBM Power use the following command:

```bash
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
```

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

**IMPORTANT**

Choose the correct Product Variant according to your architecture.

**NOTE**

Certain MCG features are only available in certain MCG versions, and the appropriate MCG CLI tool version must be used to fully utilize MCG’s features.

To replicate a bucket, see Replicating a bucket to another bucket.
To set a bucket class replication policy, see Setting a bucket class replication policy.

8.1. REPLICATING A BUCKET TO ANOTHER BUCKET

You can set the bucket replication policy in two ways:

- Replicating a bucket to another bucket using the MCG command-line interface.
- Replicating a bucket to another bucket using a YAML.

8.1.1. Replicating a bucket to another bucket using the MCG command-line interface

Applications that require a Multicloud Object Gateway (MCG) bucket to have a specific replication policy can create an Object Bucket Claim (OBC) and define the replication policy parameter in a JSON file.

Procedure

- From the MCG command-line interface, run the following command to create an OBC with a specific replication policy:

  ```bash
  noobaa obc create <bucket-claim-name> -n openshift-storage --replication-policy /path/to/json-file.json
  ```

  `<bucket-claim-name>`

  Specify the name of the bucket claim.

  `/path/to/json-file.json`

  Is the path to a JSON file which defines the replication policy.

  Example JSON file:

  ```json
  [{ "rule_id": "rule-1", "destination_bucket": "first.bucket", "filter": {"prefix": "repl"}}]
  ```

  "prefix"

  Is optional. It is the prefix of the object keys that should be replicated, and you can even leave it empty, for example, `{"prefix": ""}`.

  Example 8.1. Example

  ```bash
  noobaa obc create my-bucket-claim -n openshift-storage --replication-policy /path/to/json-file.json
  ```

8.1.2. Replicating a bucket to another bucket using a YAML

Applications that require a Multicloud Object Gateway (MCG) data bucket to have a specific replication policy can create an Object Bucket Claim (OBC) and add the `spec.additionalConfig.replication-policy` parameter to the OBC.

Procedure
- Apply the following YAML:

```yaml
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <desired-bucket-claim>
  namespace: <desired-namespace>
spec:
  generateBucketName: <desired-bucket-name>
  storageClassName: openshift-storage.noobaa.io
  additionalConfig:
    replication-policy: ["rule_id": ","destination_bucket": "first.bucket", "filter": "prefix": ","object name prefix>" ]"

<desired-bucket-claim>
  Specify the name of the bucket claim.

<desired-namespace>
  Specify the namespace.

<desired-bucket-name>
  Specify the prefix of the bucket name.

"rule_id"
  Specify the ID number of the rule, for example, {"rule_id": "rule-1"}.

"destination_bucket"
  Specify the name of the destination bucket, for example, {"destination_bucket": "first.bucket"}.

"prefix"
  Is optional. It is the prefix of the object keys that should be replicated, and you can even leave it empty, for example, {"prefix": ""}.
```

Additional information

- For more information about OBCs, see Object Bucket Claim.

### 8.2. Setting a Bucket Class Replication Policy

It is possible to set up a replication policy that automatically applies to all the buckets created under a certain bucket class. You can do this in two ways:

- Setting a bucket class replication policy using the MCG command-line interface
- Setting a bucket class replication policy using a YAML

#### 8.2.1. Setting a bucket class replication policy using the MCG command-line interface

Applications that require a Multicloud Object Gateway (MCG) bucket class to have a specific replication policy can create a bucketclass and define the replication-policy parameter in a JSON file.

It is possible to set a bucket class replication policy for two types of bucket classes:

- Placement
Procedure

- Namespace

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

  ```
  noobaa -n openshift-storage bucketclass create placement-bucketclass <bucketclass-name>
  --backingstores <backingstores> --replication-policy=/path/to/json-file.json
  ```

  **<bucketclass-name>**
  Specify the name of the bucket class.

  **<backingstores>**
  Specify the name of a backingstore. It is possible to pass several backingstores separated by commas.

  **/path/to/json-file.json**
  Is the path to a JSON file which defines the replication policy.
  Example JSON file:

  ```json
  [ {
    "rule_id": "rule-1",
    "destination_bucket": "first.bucket",
    "filter": {
      "prefix": "repl"
    }
  }]
  ```

  **"prefix"**
  Is optional. It is the prefix of the object keys that should be replicated, and you can even leave it empty, for example, `{"prefix": ""}`.

  **Example 8.2. Example**

  ```
  noobaa -n openshift-storage bucketclass create placement-bucketclass bc --backingstores azure-blob-ns --replication-policy=/path/to/json-file.json
  ```

  This example creates a placement bucket class with a specific replication policy defined in the JSON file.

8.2.2. Setting a bucket class replication policy using a YAML

Applications that require a Multicloud Object Gateway (MCG) bucket class to have a specific replication policy can create a bucket class using the `spec.replicationPolicy` field.

Procedure

1. Apply the following YAML:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: BucketClass
   metadata:
     labels:
       app: <desired-app-label>
       name: <desired-bucketclass-name>
       namespace: <desired-namespace>
   spec:
   ```
This YAML is an example that creates a placement bucket class. Each Object bucket claim (OBC) object that is uploaded to the bucket is filtered based on the prefix and is replicated to first.bucket.

**<desired-app-label>**
Specify a label for the app.

**<desired-bucketclass-name>**
Specify the bucket class name.

**<desired-namespace>**
Specify the namespace in which the bucket class gets created.

**<backingstore>**
Specify the name of a backingstore. It is possible to pass several backingstores.

"rule_id"
Specify the ID number of the rule, for example, \{"rule_id": "rule-1"\}.

"destination_bucket"
Specify the name of the destination bucket, for example, \{"destination_bucket": "first.bucket"\}.

"prefix"
Is optional. It is the prefix of the object keys that should be replicated, and you can even leave it empty, for example, \{"prefix": ""\}. 

placementPolicy:
tiers:
- backingstores:
  - <backingstore>
placement: Spread
replicationPolicy: [{ "rule_id": "<rule id>", "destination_bucket": "first.bucket", "filter": {"prefix": "<object name prefix>"}]}]
CHAPTER 9. 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 in three ways:

- Section 9.1, “Dynamic Object Bucket Claim”
- Section 9.2, “Creating an Object Bucket Claim using the command line interface”
- Section 9.3, “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.

9.1. DYNAMIC OBJECT BUCKET CLAIM

Similar to Persistent Volumes, you can add the details of the Object Bucket claim (OBC) 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:

```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 OBC itself.

a. Replace `<obc-name>` with the a unique OBC name.

b. Replace `<obc-bucket-name>` with a unique bucket name for your OBC.

2. You can add more lines to the YAML file to automate the use of the OBC. 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 claims the Object Bucket from NooBaa, which
creates a bucket and an account.

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

56
- 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 OBC name.

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

3. Apply the updated YAML file:

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

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

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

```bash
# oc get cm `<obc-name>` -o yaml
```

Replace `<obc-name>` with the name of your OBC.

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

When creating an Object Bucket Claim (OBC) 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 Multicloud Object Gateway (MCG) command-line interface.

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

**NOTE**

Specify the appropriate architecture for enabling the repositories using the subscription manager.

- For IBM Power, use the following command:

```
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
```

- For IBM Z infrastructure, use the following command:

```
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
```

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 OBC name, for example, `myappobc`.

---

- **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
```

Specify the name of the object bucket claim.
Additionally, you can use the `--app-namespace` option to specify the namespace where the OBC 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 OBC:

```
# 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 OBC:

```
# 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 OBC. The CM and the secret have the same name as the OBC. Run the following command to view the secret:
The secret gives you the S3 access credentials.

5. Run the following command 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

Red Hat OpenShift Data Foundation 4.9 Managing hybrid and multicloud resources
The configuration map contains the S3 endpoint information for your application.

9.3. 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 9.1, "Dynamic Object Bucket Claim".

Procedure

1. Log into the OpenShift Web Console.
2. On the left navigation bar, click **Storage → Object Bucket Claims → Create Object Bucket Claim**.
   a. 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 (MCG)

**External mode**

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

- **ocs-external-storagecluster-ceph-rgw** uses the RGW
• **openshift-storage.noobaa.io** uses the MCG

**NOTE**

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

b. Click **Create**.

Once you create the OBC, you are redirected to its detail page:

![Object Bucket Claim Details](image)

**Additional Resources**

- **Chapter 9, Object Bucket Claim**

**9.4. 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.
   a. From the drop-down menu, select **Attach to Deployment**
b. Select the desired deployment from the Deployment Name list, then click **Attach**.

**Attach OBC to a Deployment**

`Deployment Name`  

`Cancel`  

`Attach`

### Additional Resources

- Chapter 9, *Object Bucket Claim*

### 9.5. 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**

1. Log into the OpenShift Web Console.

2. On the left navigation bar, click **Storage** → **Object Buckets**.
Alternatively, 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 Details page.

Additional Resources

- Chapter 9, Object Bucket Claim

9.6. 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 the Action menu (⋮) next to the Object Bucket Claim (OBC) you want to delete.

   a. Select Delete Object Bucket Claim
b. Click **Delete**.

**Additional Resources**

- **Chapter 9, Object Bucket Claim**
CHAPTER 10. 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 (MCG) bucket. You can create a cache bucket that caches an AWS bucket or an IBM COS bucket.

- AWS S3
- IBM COS

10.1. CREATING AN AWS CACHE BUCKET

Prerequisites

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

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

**NOTE**

Specify the appropriate architecture for enabling the repositories using the subscription manager. In case of IBM Z infrastructure use the following command:

```
# subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
```

Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found here [https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package](https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package).

**NOTE**

Choose the correct Product Variant according to your architecture.

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 the MCG 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 the MCG 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:

```yaml
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:

```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:

```bash
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 backing stores separated by commas in this field.

c. Replace `<namespacestore>` with the namespace store created in the previous step.

3. Run the following command to create a bucket using an Object Bucket Claim (OBC) 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.

### 10.2. CREATING AN IBM COS CACHE BUCKET

#### Prerequisites

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

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

  **NOTE**

  Specify the appropriate architecture for enabling the repositories using the subscription manager.

  - For IBM Power, use the following command:

    ```
    # subscription-manager repos --enable=rh-odf-4-for-rhel-8-ppc64le-rpms
    ```

  - For IBM Z infrastructure, use the following command:

    ```
    # subscription-manager repos --enable=rh-odf-4-for-rhel-8-s390x-rpms
    ```

  Alternatively, you can install the MCG package from the OpenShift Data Foundation RPMs found here [https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package](https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package).

  **NOTE**

  Choose the correct Product Variant according to your architecture.

#### 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 the MCG namespace buckets. From the MCG command-line interface, run the following command:
noobaa namespaciostore 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 the MCG 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:

```yaml
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:

```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:

   ```bash
   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.

   ```bash
   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 11. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE BY ADDING ENDPOINTS

The Multicloud Object Gateway (MCG) 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.

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

- Storage service
- S3 endpoint service

**S3 endpoint service**

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

### 11.1. 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. OpenShift Data Foundation 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.

### 11.2. SCALING THE MULTICLOUD OBJECT GATEWAY WITH STORAGE NODES

**Prerequisites**

- A running OpenShift Data Foundation cluster on OpenShift Container Platform with access to the Multicloud Object Gateway (MCG).

A storage node in the MCG is a NooBaa daemon container attached to one or more Persistent Volumes (PVs) 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. Log in to **OpenShift Web Console**.
2. From the MCG user interface, click **Overview → Add Storage Resources**.
3. In the window, click **Deploy Kubernetes Pool**
4. In the **Create Pool** step create the target pool for the future installed nodes.
5. In the **Configure** step, configure the number of requested pods and the size of each PV. For each new pod, one PV is to be created.

6. 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.

7. All nodes will be assigned to the pool you chose in the first step, and can be found under **Resources → Storage resources → Resource name**.
CHAPTER 12. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT

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

Prerequisites

- A running OpenShift Data Foundation Platform

Procedure

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

   ```bash
   $ oc get service -n openshift-storage
   NAME                   TYPE            CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
   (...)                  (...)           (...)            (...)         (...)      (...)
   rook-ceph-rgw-ocs-storagecluster-cephobjectstore  ClusterIP  172.30.145.254   <none>   80/TCP,443/TCP  5d7h
   (...)
   ```

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

   ```bash
   $ oc expose svc/<RGW service name> --hostname=<route name>
   a. Replace `<RGW-service name>` with the RGW service name from the previous step.
   b. Replace `<route name>` with a route you want to create for the RGW service.
      For example:

   ```bash
   $ 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.

   ```bash
   $ oc get route -n openshift-storage
   Example output:
   NAME                                                 HOST/PORT                                PATH
   rook-ceph-rgw-ocs-storagecluster-cephobjectstore      rook-ceph-rgw-ocs.ocp.host.example.com
   SERVICES                                             PORT     TERMINATION WILDCARD
   rook-ceph-rgw-ocs-storagecluster-cephobjectstore      http     <none>
   ```

Verification
To verify the ENDPOINT, run the following command:

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

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

For example:

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

**IMPORTANT**

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

- Access key:

  ```bash
  $ 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:

  ```bash
  $ 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
  ```