



Red Hat OpenShift Data Foundation 4.12

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).

Red Hat OpenShift Data Foundation 4.12 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).

Legal Notice

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

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. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD	14
3.1. CREATING A NEW BACKING STORE	14
3.2. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD USING THE MCG COMMAND LINE INTERFACE	15
3.2.1. Creating an AWS-backed backingstore	15
3.2.2. Creating an IBM COS-backed backingstore	17
3.2.3. Creating an Azure-backed backingstore	19
3.2.4. Creating a GCP-backed backingstore	21
3.2.5. Creating a local Persistent Volume-backed backingstore	23
3.3. CREATING AN S3 COMPATIBLE MULTICLOUD OBJECT GATEWAY BACKINGSTORE	25
3.4. CREATING A NEW BUCKET CLASS	26
3.5. EDITING A BUCKET CLASS	27
3.6. EDITING BACKING STORES FOR BUCKET CLASS	28
CHAPTER 4. MANAGING NAMESPACE BUCKETS	30
4.1. AMAZON S3 API ENDPOINTS FOR OBJECTS IN NAMESPACE BUCKETS	30
4.2. ADDING A NAMESPACE BUCKET USING THE MULTICLOUD OBJECT GATEWAY CLI AND YAML	31
4.2.1. Adding an AWS S3 namespace bucket using YAML	31
4.2.2. Adding an IBM COS namespace bucket using YAML	33
4.2.3. Adding an AWS S3 namespace bucket using the Multicloud Object Gateway CLI	36
4.2.4. Adding an IBM COS namespace bucket using the Multicloud Object Gateway CLI	38
4.3. ADDING A NAMESPACE BUCKET USING THE OPENSIFT CONTAINER PLATFORM USER INTERFACE	40
4.4. SHARING LEGACY APPLICATION DATA WITH CLOUD NATIVE APPLICATION USING S3 PROTOCOL	42
4.4.1. Creating a NamespaceStore to use a file system	42
4.4.2. Creating accounts with NamespaceStore filesystem configuration	43
4.4.3. Accessing legacy application data from the openshift-storage namespace	45
4.4.3.1. Changing the default SELinux label on the legacy application project to match the one in the openshift-storage project	54
4.4.3.2. Modifying the SELinux label only for the deployment config that has the pod which mounts the legacy application PVC	54
CHAPTER 5. SECURING MULTICLOUD OBJECT GATEWAY	58
5.1. CHANGING THE DEFAULT ACCOUNT CREDENTIALS TO ENSURE BETTER SECURITY IN THE MULTICLOUD OBJECT GATEWAY	58
5.1.1. Resetting the noobaa account password	58
5.1.2. Regenerating the S3 credentials for the accounts	59
5.1.3. Regenerating the S3 credentials for the OBC	61
5.2. ENABLING SECURED MODE DEPLOYMENT FOR MULTICLOUD OBJECT GATEWAY	62
CHAPTER 6. MIRRORING DATA FOR HYBRID AND MULTICLOUD BUCKETS	64
6.1. CREATING BUCKET CLASSES TO MIRROR DATA USING THE MCG COMMAND-LINE-INTERFACE	64
6.2. CREATING BUCKET CLASSES TO MIRROR DATA USING A YAML	64

CHAPTER 7. BUCKET POLICIES IN THE MULTICLOUD OBJECT GATEWAY	66
7.1. INTRODUCTION TO BUCKET POLICIES	66
7.2. USING BUCKET POLICIES IN MULTICLOUD OBJECT GATEWAY	66
7.3. CREATING A USER IN THE MULTICLOUD OBJECT GATEWAY	67
CHAPTER 8. MULTICLOUD OBJECT GATEWAY BUCKET REPLICATION	69
8.1. REPLICATING A BUCKET TO ANOTHER BUCKET	69
8.1.1. Replicating a bucket to another bucket using the MCG command-line interface	70
8.1.2. Replicating a bucket to another bucket using a YAML	70
8.2. SETTING A BUCKET CLASS REPLICATION POLICY	71
8.2.1. Setting a bucket class replication policy using the MCG command-line interface	71
8.2.2. Setting a bucket class replication policy using a YAML	72
CHAPTER 9. OBJECT BUCKET CLAIM	74
9.1. DYNAMIC OBJECT BUCKET CLAIM	74
9.2. CREATING AN OBJECT BUCKET CLAIM USING THE COMMAND LINE INTERFACE	76
9.3. CREATING AN OBJECT BUCKET CLAIM USING THE OPENSIFT WEB CONSOLE	79
9.4. ATTACHING AN OBJECT BUCKET CLAIM TO A DEPLOYMENT	80
9.5. VIEWING OBJECT BUCKETS USING THE OPENSIFT WEB CONSOLE	80
9.6. DELETING OBJECT BUCKET CLAIMS	81
CHAPTER 10. CACHING POLICY FOR OBJECT BUCKETS	82
10.1. CREATING AN AWS CACHE BUCKET	82
10.2. CREATING AN IBM COS CACHE BUCKET	84
CHAPTER 11. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE	87
11.1. AUTOMATIC SCALING OF MULTICLOUD OBJECT GATEWAY ENDPOINTS	87
11.2. SCALING THE MULTICLOUD OBJECT GATEWAY WITH STORAGE NODES	87
11.3. INCREASING CPU AND MEMORY FOR PV POOL RESOURCES	88
CHAPTER 12. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT	89

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"](#)

For example:

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

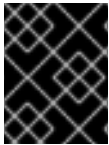
If the client application tries to access `https://<bucket-name>.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com`

<bucket-name>

is the name of the MCG bucket

For example, `https://mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com`

A DNS entry is needed for **mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com** to point to the S3 Service.



IMPORTANT

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

2.1. ACCESSING THE MULTICLOUD OBJECT GATEWAY FROM THE TERMINAL

Procedure

Run the **describe** command to view information about the 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:
  Creation Timestamp: 2019-07-29T16:22:06Z
  Generation:        1
  Resource Version:   6718822
  Self Link:          /apis/noobaa.io/v1alpha1/namespaces/openshift-storage/noobaas/noobaa
  UID:                019cfb4a-b21d-11e9-9a02-06c8de012f9e
Spec:
Status:
  Accounts:
    Admin:
      Secret Ref:
        Name:      noobaa-admin
        Namespace: openshift-storage
  Actual Image:    noobaa/noobaa-core:4.0
  Observed Generation: 1
  Phase:           Ready
  Readme:

  Welcome to NooBaa!
  -----

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

NooBaa Core Version:
NooBaa Operator Version:

Lets get started:

1. Connect to Management console:

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

```
kubectrl 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:

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

2. Test S3 client:

```
kubectrl port-forward -n openshift-storage service/s3 10443:443 &
```

1

```
NOOBAA_ACCESS_KEY=$(kubectrl get secret noobaa-admin -n openshift-storage -o json | jq -r  
'data.AWS_ACCESS_KEY_ID|@base64d')
```

2

```
NOOBAA_SECRET_KEY=$(kubectrl get secret noobaa-admin -n openshift-storage -o json | jq -r  
'data.AWS_SECRET_ACCESS_KEY|@base64d')  
alias s3='AWS_ACCESS_KEY_ID=$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-](https://a3406079515be11eaa3b70683061451e-1194613580.us-east-2.elb.amazonaws.com:443)

[2.elb.amazonaws.com:443](https://a3406079515be11eaa3b70683061451e-1194613580.us-east-2.elb.amazonaws.com:443)

Internal DNS:

<https://noobaa-mgmt.openshift-storage.svc:443>

Internal IP:

<https://172.30.235.12:443>

Node Ports:

<https://10.0.142.103:31385>

Pod Ports:

<https://10.131.0.19:8443>

serviceS3:

External DNS: 3

<https://s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com>

<https://a340f4e1315be11eaa3b70683061451e-943168195.us-east-2.elb.amazonaws.com:443>

Internal DNS:

<https://s3.openshift-storage.svc:443>

Internal IP:

<https://172.30.86.41:443>

Node Ports:

```
https://10.0.142.103:31011
Pod Ports:
https://10.131.0.19:6443
```

- 1 access key (**AWS_ACCESS_KEY_ID** value)
- 2 secret access key (**AWS_SECRET_ACCESS_KEY** value)
- 3 MCG endpoint



NOTE

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

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

Prerequisites

- Download the MCG command-line interface.

```
# subscription-manager repos --enable=rh-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]
INFO[0000] CRD Status:
INFO[0003] Exists: CustomResourceDefinition "noobaas.noobaa.io"
```

```

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

```

```
#-----#
```

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

```
#-----#
```

```

ExternalDNS : [https://noobaa-mgmt-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
https://a3406079515be11eaa3b70683061451e-1194613580.us-east-2.elb.amazonaws.com:443]

```

```
ExternalIP : []
```

```
NodePorts : [https://10.0.142.103:31385]
```

```
InternalDNS : [https://noobaa-mgmt.openshift-storage.svc:443]
```

```
InternalIP : [https://172.30.235.12:443]
```

```
PodPorts : [https://10.131.0.19:8443]
```

```
#-----#
```

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

```
#-----#
```

```
email : admin@noobaa.io
```

```
password : HKLbH1rSuVU0l/souIkSiA==
```

```
#-----#
```

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

```
#-----#
```

1

```

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

```

```

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

```

2

```

AWS_ACCESS_KEY_ID : jVmAsu9FsvRHYmfjTiHV

```

3

```

AWS_SECRET_ACCESS_KEY : E//420VNedJfATvVSmDz6FMtsSAzuBv6z180PT5c

```

```

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

```

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

```

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

```

NAME	PLACEMENT	PHASE	AGE
noobaa-default-bucket-class	{Tiers:[{Placement: BackingStores:[noobaa-default-backing-store]]}	Ready	141h35m33s

```

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

```

```

No OBC's found.

```

1

endpoint

2

access key

3

secret access key

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

For 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. ADDING STORAGE RESOURCES FOR HYBRID OR MULTICLOUD

3.1. 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 → 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 3.2, "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 → Data Foundation**.
2. Click the **Backing Store** tab to view all the backing stores.

3.2. 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 3.2.1, “Creating an AWS-backed backingstore”](#)
- For creating an IBM COS-backed backingstore, see [Section 3.2.2, “Creating an IBM COS-backed backingstore”](#)
- For creating an Azure-backed backingstore, see [Section 3.2.3, “Creating an Azure-backed backingstore”](#)
- For creating a GCP-backed backingstore, see [Section 3.2.4, “Creating a GCP-backed backingstore”](#)
- For creating a local Persistent Volume-backed backingstore, see [Section 3.2.5, “Creating a local Persistent Volume-backed backingstore”](#)

For VMware deployments, skip to [Section 3.3, “Creating an s3 compatible Multicloud Object Gateway backingstore”](#) for further instructions.

3.2.1. Creating an AWS-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:

```
# 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

Using MCG command-line interface

- 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
```

<backingstore_name>

The name of the backingstore.

<AWS ACCESS KEY> and <AWS SECRET ACCESS KEY>

The AWS access key ID and secret access key you created for this purpose.

<bucket-name>

The existing AWS bucket name. This argument indicates to 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"
```

Adding storage resources using a YAML

- Create a secret with the credentials:

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

<AWS ACCESS KEY> and <AWS SECRET ACCESS KEY>

Supply and encode your own AWS access key ID and secret access key using Base64, and use the results for <AWS ACCESS KEY ID ENCODED IN BASE64> and <AWS SECRET ACCESS KEY ENCODED IN BASE64>.

<backingstore-secret-name>

The name of the backingstore secret created in the previous step.

- Apply the following YAML for a specific backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
    - noobaa.io/finalizer
  labels:
```

```

    app: noobaa
    name: bs
    namespace: openshift-storage
    spec:
      awsS3:
        secret:
          name: <backingstore-secret-name>
          namespace: openshift-storage
        targetBucket: <bucket-name>
      type: aws-s3

```

<bucket-name>

The existing AWS bucket name.

<backingstore-secret-name>

The name of the backingstore secret created in the previous step.

3.2.2. 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 example,

- 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/packages

NOTE

Choose the correct Product Variant according to your architecture.

Procedure

Using command-line interface

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
```

<backingstore_name>

The name of the backingstore.

<IBM ACCESS KEY>, <IBM SECRET ACCESS KEY>, and <IBM COS ENDPOINT>

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.

<bucket-name>

An existing IBM bucket name. This argument indicates MCG about the 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"
```

Adding storage resources using an YAML

1. Create a secret with the credentials:

```
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>
```

<IBM COS ACCESS KEY ID ENCODED IN BASE64> and <IBM COS SECRET ACCESS KEY ENCODED IN BASE64>

Provide and encode your own IBM COS access key ID and secret access key using Base64, and use the results in place of these attributes respectively.

<backingstore-secret-name>

The name of the backingstore secret.

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

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
    - noobaa.io/finalizer
labels:
```

```

    app: noobaa
    name: bs
    namespace: openshift-storage
spec:
  ibmCos:
    endpoint: <endpoint>
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBucket: <bucket-name>
    type: ibm-cos

```

<bucket-name>

an existing IBM COS bucket name. This argument indicates to MCG about the bucket to use as a target bucket for its backingstore, and subsequently, data storage and administration.

<endpoint>

A regional endpoint that corresponds to the location of the existing IBM bucket name. This argument indicates to MCG about the endpoint to use for its backingstore, and subsequently, data storage and administration.

<backingstore-secret-name>

The name of the secret created in the previous step.

3.2.3. Creating an Azure-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:

```
# 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

Using the MCG command-line interface

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

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

<backingstore_name>

The name of the backingstore.

<AZURE ACCOUNT KEY> and <AZURE ACCOUNT NAME>

An AZURE account key and account name you created for this purpose.

<blob container name>

An existing Azure blob container name. This argument indicates to MCG about the bucket to use as a target bucket for its backingstore, and subsequently, data storage and administration.

The output will be similar to the following:

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

Adding storage resources using a YAML

- Create a secret with the credentials:

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

<AZURE ACCOUNT NAME ENCODED IN BASE64> and <AZURE ACCOUNT KEY ENCODED IN BASE64>

Supply and encode your own Azure Account Name and Account Key using Base64, and use the results in place of these attributes respectively.

<backingstore-secret-name>

A unique name of backingstore secret.

- Apply the following YAML for a specific backing store:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
    - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
```



```
spec:
  azureBlob:
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBlobContainer: <blob-container-name>
  type: azure-blob
```

<blob-container-name>

An existing Azure blob container name. This argument indicates to the MCG about the bucket to use as a target bucket for its backingstore, and subsequently, data storage and administration.

<backingstore-secret-name>

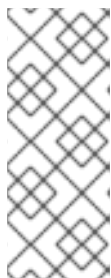
with the name of the secret created in the previous step.

3.2.4. 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:

```
# 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

Using the MCG command-line interface

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

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

<backingstore_name>

Name of the backingstore.

<PATH TO GCP PRIVATE KEY JSON FILE>

A path to your GCP private key created for this purpose.

<GCP bucket name>

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:

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

Adding storage resources using a YAML

1. Create a secret with the credentials:

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

<GCP PRIVATE KEY ENCODED IN BASE64>

Provide and encode your own GCP service account private key using Base64, and use the results for this attribute.

<backingstore-secret-name>

A unique name of the backingstore secret.

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

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

<target bucket>

An existing Google storage bucket. This argument indicates to the MCG about the bucket to use as a target bucket for its backing store, and subsequently, data storage dfdand administration.

<backingstore-secret-name>

The name of the secret created in the previous step.

3.2.5. Creating a local Persistent Volume-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 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/packages



NOTE

Choose the correct Product Variant according to your architecture.

Procedure

Adding storage resources using the MCG command-line interface

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



NOTE

This command must be run from within the **openshift-storage** namespace.

```
$ noobaa -n openshift-storage backingstore create pv-pool <backingstore_name> --num-
volumes <NUMBER OF VOLUMES> --pv-size-gb <VOLUME SIZE> --request-cpu <CPU
REQUEST> --request-memory <MEMORY REQUEST> --limit-cpu <CPU LIMIT> --limit-
memory <MEMORY LIMIT> --storage-class <LOCAL STORAGE CLASS>
```

Adding storage resources using YAML

- 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>
        cpu: <CPU REQUEST>
        memory: <MEMORY REQUEST>
      limits:
        cpu: <CPU LIMIT>
        memory: <MEMORY LIMIT>
    storageClass: <LOCAL STORAGE CLASS>
  type: pv-pool
```

<backingstore_name>

The name of the backingstore.

<NUMBER OF VOLUMES>

The number of volumes you would like to create. Note that increasing the number of volumes scales up the storage.

<VOLUME SIZE>

Required size in GB of each volume.

<CPU REQUEST>

Guaranteed amount of CPU requested in CPU unit **m**.

<MEMORY REQUEST>

Guaranteed amount of memory requested.

<CPU LIMIT>

Maximum amount of CPU that can be consumed in CPU unit **m**.

<MEMORY LIMIT>

Maximum amount of memory that can be consumed.

<LOCAL STORAGE CLASS>

The local storage class name, recommended to use **ocs-storagecluster-ceph-rbd**.
The output will be similar to the following:

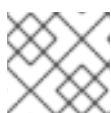
```
INFO[0001] Exists: NooBaa "noobaa"
INFO[0002] Exists: BackingStore "local-mcg-storage"
```

3.3. 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:



NOTE

This command must be run from within the **openshift-storage** namespace.

```
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:

```
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](#).

3.4. 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 → 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](#).

**NOTE**

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

- d. Review and confirm Bucket Class settings.
- e. Click **Create Bucket Class**.

Verification steps

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

3.5. 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 → 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**.


3.6. EDITING BACKING STORES FOR BUCKET CLASS

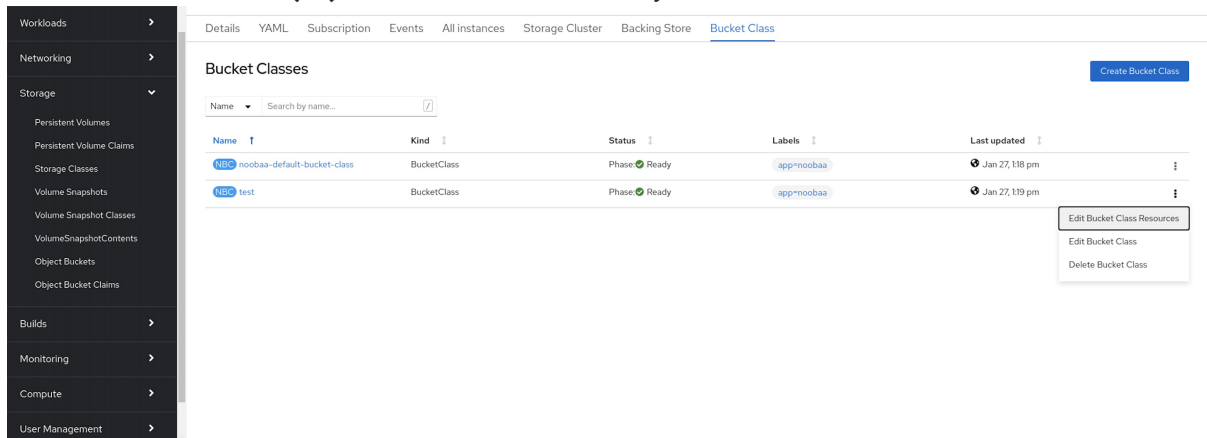
Use the following procedure to edit an existing Multicloud Object Gateway (MCG) bucket class to change the underlying backing stores used in a bucket class.

Prerequisites

- Administrator access to OpenShift Web Console.
- A bucket class.
- Backing stores.

Procedure

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

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

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

Tier 1- Backing Stores (Spread)

Select at least 2 Resources resources *

☒ Name

Name	Target Bucket	Type	Region
<input checked="" type="checkbox"/> aws-s3-main	my-aws	AWS-S3	Eu-east-1a
<input checked="" type="checkbox"/> bucket-main-azure	bucket-main	Azure Blob	Us-east-1b
<input type="checkbox"/> archive-bucket	buck-1	S3 Compatible	Us-east-1a

2 Backing Stores selected

Tier 2- Backing Stores (Mirror)

Select at least 2 Resources resources *

☒ Name

Name	Target Bucket	Type	Region
<input checked="" type="checkbox"/> archive-bucket	buck-1	S3 Compatible	Us-east-1a
<input checked="" type="checkbox"/> data-bucket	bucket-main	Azure Blob	Us-east-1b
<input checked="" type="checkbox"/> buck-2	buck-1	S3 Compatible	Us-east-1a

2 Backing Stores selected

Save Cancel

6. Click **Save**.

CHAPTER 4. MANAGING NAMESPACE BUCKETS

Namespace buckets let you connect data repositories on different providers together, so that 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.

4.1. 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.

Additional resources

- [Amazon S3 REST API Reference](#)
- [Amazon S3 CLI Reference](#)

4.2. 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](#)

4.2.1. Adding an AWS S3 namespace bucket using YAML

Prerequisites

- OpenShift Container Platform with OpenShift Data Foundation operator installed.
- Access to the Multicloud Object Gateway (MCG).
For information, see Chapter 2, [Accessing the Multicloud Object Gateway with your applications](#).

Procedure

1. Create a secret with the credentials:

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

where **<namespacestore-secret-name>** is a unique NamespaceStore name.

You must provide 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>**.

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:

—

```

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

```

<resource-name>

The name you want to give to the resource.

<namespacestore-secret-name>

The secret created in the previous step.

<namespace-secret>

The namespace where the secret can be found.

<target-bucket>

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>

```

<my-bucket-class>

The unique namespace bucket class name.

<resource>

The name of a single NamespaceStore 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>

```

<my-bucket-class>

A unique bucket class name.

<write-resource>

The name of a single NamespaceStore that defines the **write** target of the namespace bucket.

<read-resources>

A list of the names of the NamespaceStores that defines the **read** targets of the namespace bucket.

4. Create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in the earlier step using the following 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>

```

<resource-name>

The name you want to give to the resource.

<my-bucket>

The name you want to give to the bucket.

<my-bucket-class>

The bucket class created in the previous step.

After 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 in the same namespace as that of the OBC.

4.2.2. Adding an IBM COS namespace bucket using YAML

Prerequisites

- OpenShift Container Platform with OpenShift Data Foundation operator installed.
- 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:

```
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>
```

<namespacestore-secret-name>

A unique NamespaceStore name.

You must provide 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>**.

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:

```
apiVersion: noobaa.io/v1alpha1
kind: 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>
  type: ibm-cos
```

<IBM COS ENDPOINT>

The appropriate IBM COS endpoint.

<namespacestore-secret-name>

The secret created in the previous step.

<namespace-secret>

The namespace where the secret can be found.

<target-bucket>

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**.
 - The 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>
```

<my-bucket-class>

The unique namespace bucket class name.

<resource>

The name of a single NamespaceStore that defines the **read** and **write** target of the namespace bucket.

- The 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>
```

<my-bucket-class>

The unique bucket class name.

<write-resource>

The name of a single NamespaceStore that defines the write target of the namespace bucket.

<read-resources>

A list of the NamespaceStores names that defines the **read** targets of the namespace bucket.

4. To create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in the previous step, apply the following 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>
```

<resource-name>

The name you want to give to the resource.

<my-bucket>

The name you want to give to the bucket.

<my-bucket-class>

The bucket class created in the previous step.

After 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 in the same namespace as that of the OBC.

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

Prerequisites

- Openshift Container Platform with OpenShift Data Foundation operator installed.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, [Accessing the Multicloud Object Gateway with your applications](#).
- 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 subscription manager. For instance, 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.

**NOTE**

Choose the correct Product Variant according to your architecture.

Procedure

1. In the MCG command-line interface, 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.

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

<namespacestore>

The name of the NamespaceStore.

<AWS ACCESS KEY> and <AWS SECRET ACCESS KEY>

The AWS access key ID and secret access key you created for this purpose.

<bucket-name>

The 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 can be either **single** or **multi**.

- 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
```

<resource-name>

The name you want to give the resource.

<my-bucket-class>

A unique bucket class name.

<resource>

A single namespace-store that defines the **read** and **write** target of the namespace bucket.

- 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
```

<resource-name>

The name you want to give the resource.

<my-bucket-class>

A unique bucket class name.

<write-resource>

A single namespace-store that defines the **write** target of the namespace bucket.

<read-resources>s

A list of namespace-stores separated by commas that defines the **read** targets of the namespace bucket.

3. Create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in the previous step.

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

<bucket-name>

A bucket name of your choice.

<custom-bucket-class>

The name of the bucket class created in the previous step.

After the OBC is provisioned by the operator, a bucket is created in the MCG, and the operator creates a **Secret** and a **ConfigMap** with the same name and in the same namespace as that of the OBC.

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

Prerequisites

- Openshift Container Platform with OpenShift Data Foundation operator installed.
- Access to the Multicloud Object Gateway (MCG), see Chapter 2, [Accessing the Multicloud Object Gateway with your applications](#).
- 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 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.

**NOTE**

Choose the correct Product Variant according to your architecture.

Procedure

- In the MCG command-line interface, 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.

```
$ 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
```

<namespacestore>

The name of the NamespaceStore.

<IBM ACCESS KEY>, <IBM SECRET ACCESS KEY>, <IBM COS ENDPOINT>

An IBM access key ID, secret access key, and the appropriate regional endpoint that corresponds to the location of the existing IBM bucket.

<bucket-name>

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.

- 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**.

- 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
```

<resource-name>

The name you want to give the resource.

<my-bucket-class>

A unique bucket class name.

<resource>

A single NamespaceStore that defines the **read** and **write** target of the namespace bucket.

- 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
```

<resource-name>

The name you want to give the resource.

<my-bucket-class>

A unique bucket class name.

<write-resource>

A single NamespaceStore that defines the **write** target of the namespace bucket.

<read-resources>

A comma-separated list of NamespaceStores that defines the **read** targets of the namespace bucket.

3. Create a bucket using an Object Bucket Class (OBC) resource that uses the bucket class defined in the earlier step.

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

<bucket-name>

A bucket name of your choice.

<custom-bucket-class>

The name of the bucket class created in the previous step.

After 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 in the same namespace as that of the OBC.

4.3. ADDING A NAMESPACE BUCKET USING THE OPENSIFT CONTAINER PLATFORM USER INTERFACE

You can add namespace buckets using the OpenShift Container Platform user interface. For 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 → 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 that 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. (Optional) Add description.
 - d. Click **Next**.
5. Choose a namespace policy type for your namespace bucket, and then click **Next**.
6. Select the target resources.
 - 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 → 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 the earlier step 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 steps

- 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.

4.4. SHARING LEGACY APPLICATION DATA WITH CLOUD NATIVE APPLICATION USING S3 PROTOCOL

Many legacy applications use file systems to share data sets. You can access and share the legacy data in the file system by using the S3 operations. To share data you need to do the following:

- Export the pre-existing file system datasets, that is, RWX volume such as Ceph FileSystem (CephFS) or create a new file system datasets using the S3 protocol.
- Access file system datasets from both file system and S3 protocol.
- Configure S3 accounts and map them to the existing or a new file system unique identifiers (UIDs) and group identifiers (GIDs).

4.4.1. Creating a NamespaceStore to use a file system

Prerequisites

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

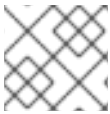
Procedure

1. Log into the OpenShift Web Console.
2. Click **Storage → Data Foundation**.
3. Click the **NamespaceStore** tab to create NamespaceStore resources to be used in the namespace bucket.
4. Click **Create namespacestore**.
5. Enter a name for the NamespaceStore.

6. Choose **Filesystem** as the provider.
7. Choose the Persistent volume claim.
8. Enter a folder name.
If the folder name exists, then that folder is used to create the NamespaceStore or else a folder with that name is created.
9. Click **Create**.
10. Verify the NamespaceStore is in the Ready state.

4.4.2. Creating accounts with NamespaceStore filesystem configuration

You can either create a new account with NamespaceStore filesystem configuration or convert an existing normal account into a NamespaceStore filesystem account by editing the YAML.



NOTE

You cannot remove a NamespaceStore filesystem configuration from an account.

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
```

Procedure

- Create a new account with NamespaceStore filesystem configuration using the MCG command-line interface.

```
$ noobaa account create <noobaa-account-name> [flags]
```

For example:

```
$ noobaa account create testaccount --full_permission --nsfs_account_config --gid 10001 --uid 10001 --default_resource fs_namespacestore
```

allow_bucket_create	Indicates whether the account is allowed to create new buckets. Supported values are true or false . Default value is true .
allowed_buckets	A comma separated list of bucket names to which the user is allowed to have access and management rights.
default_resource	The NamespaceStore resource on which the new buckets will be created when using the S3 CreateBucket operation. The NamespaceStore must be backed by an RWX (ReadWriteMany) persistent volume claim (PVC).
full_permission	Indicates whether the account should be allowed full permission or not. Supported values are true or false . Default value is false .

new_buckets_path	The filesystem path where directories corresponding to new buckets will be created. The path is inside the filesystem of NamespaceStore filesystem PVCs where new directories are created to act as the filesystem mapping of newly created object bucket classes.
nsfs_account_config	A mandatory field that indicates if the account is used for NamespaceStore filesystem.
nsfs_only	Indicates whether the account is used only for NamespaceStore filesystem or not. Supported values are true or false . Default value is false . If it is set to 'true', it limits you from accessing other types of buckets.
uid	The user ID of the filesystem to which the MCG account will be mapped and it is used to access and manage data on the filesystem
gid	The group ID of the filesystem to which the MCG account will be mapped and it is used to access and manage data on the filesystem

The MCG system sends a response with the account configuration and its S3 credentials:

```
# NooBaaAccount spec:
allow_bucket_creation: true
Allowed_buckets:
  full_permission: true
  permission_list: []
default_resource: noobaa-default-namespace-store
Nsfs_account_config:
  gid: 10001
  new_buckets_path: /
  nsfs_only: true
  uid: 10001
INFO[0006] Exists: Secret "noobaa-account-testaccount"
Connection info:
  AWS_ACCESS_KEY_ID    : <aws-access-key-id>
  AWS_SECRET_ACCESS_KEY : <aws-secret-access-key>
```

You can list all the custom resource definition (CRD) based accounts by using the following command:

```
$ noobaa account list
NAME          ALLOWED_BUCKETS  DEFAULT_RESOURCE          PHASE  AGE
testaccount  [*]             noobaa-default-backing-store  Ready  1m17s
```

If you are interested in a particular account, you can read its custom resource definition (CRD) directly by the account name:

```
$ oc get noobaaaccount/testaccount -o yaml
spec:
  allow_bucket_creation: true
  allowed_buckets:
    full_permission: true
```



```

    permission_list: []
    default_resource: noobaa-default-namespace-store
    nsfs_account_config:
      gid: 10001
      new_buckets_path: /
      nsfs_only: true
      uid: 10001

```

4.4.3. Accessing legacy application data from the openshift-storage namespace

When using the Multicloud Object Gateway (MCG) NamespaceStore filesystem (NSFS) feature, you need to have the Persistent Volume Claim (PVC) where the data resides in the **openshift-storage** namespace. In almost all cases, the data you need to access is not in the **openshift-storage** namespace, but in the namespace that the legacy application uses.

In order to access data stored in another namespace, you need to create a PVC in the **openshift-storage** namespace that points to the same CephFS volume that the legacy application uses.

Procedure

1. Display the application namespace with **scc**:

```
$ oc get ns <application_namespace> -o yaml | grep scc
```

<application_namespace>

Specify the name of the application namespace.
For example:

```

$ oc get ns testnamespace -o yaml | grep scc

openshift.io/sa.scc.mcs: s0:c26,c5
openshift.io/sa.scc.supplemental-groups: 1000660000/10000
openshift.io/sa.scc.uid-range: 1000660000/10000

```

2. Navigate into the application namespace:

```
$ oc project <application_namespace>
```

For example:

```
$ oc project testnamespace
```

3. Ensure that a ReadWriteMany (RWX) PVC is mounted on the pod that you want to consume from the noobaa S3 endpoint using the MCG NSFS feature:

```

$ oc get pvc

NAME                                STATUS VOLUME
CAPACITY ACCESS MODES STORAGECLASS  AGE
cephfs-write-workload-generator-no-cache-pv-claim Bound pvc-aa58fb91-c3d2-475b-bbee-68452a613e1a
10Gi   RWX             ocs-storagecluster-cephfs 12s

```

```
$ oc get pod
```

NAME	READY	STATUS	RESTARTS	AGE
cephfs-write-workload-generator-no-cache-1-cv892	1/1	Running	0	11s

4. Check the mount point of the Persistent Volume (PV) inside your pod.

- a. Get the volume name of the PV from the pod:

```
$ oc get pods <pod_name> -o jsonpath='{.spec.volumes[]}'
```

<pod_name>

Specify the name of the pod.

For example:

```
$ oc get pods cephfs-write-workload-generator-no-cache-1-cv892 -o
jsonpath='{.spec.volumes[]}'

{"name":"app-persistent-storage","persistentVolumeClaim":{"claimName":"cephfs-
write-workload-generator-no-cache-pv-claim"}}
```

In this example, the name of the volume for the PVC is **cephfs-write-workload-generator-no-cache-pv-claim**.

- b. List all the mounts in the pod, and check for the mount point of the volume that you identified in the previous step:

```
$ oc get pods <pod_name> -o jsonpath='{.spec.containers[].volumeMounts}'
```

For example:

```
$ oc get pods cephfs-write-workload-generator-no-cache-1-cv892 -o
jsonpath='{.spec.containers[].volumeMounts}'

[{"mountPath":"/mnt/pv","name":"app-persistent-storage"},
{"mountPath":"/var/run/secrets/kubernetes.io/serviceaccount","name":"kube-api-access-
8tnc5","readOnly":true}]
```

5. Confirm the mount point of the RWX PV in your pod:

```
$ oc exec -it <pod_name> -- df <mount_path>
```

<mount_path>

Specify the path to the mount point that you identified in the previous step.

For example:

```
$ oc exec -it cephfs-write-workload-generator-no-cache-1-cv892 -- df /mnt/pv

main
Filesystem
1K-blocks Used Available Use% Mounted on
```

```
172.30.202.87:6789,172.30.120.254:6789,172.30.77.247:6789:/volumes/csi/csi-vol-cc416d9e-dbf3-11ec-b286-0a580a810213/edcfe4d5-bdcb-4b8e-8824-8a03ad94d67c10485760 0 10485760 0% /mnt/pv
```

6. Ensure that the UID and SELinux labels are the same as the ones that the legacy namespace uses:

```
$ oc exec -it <pod_name> -- ls -latrZ <mount_path>
```

For example:

```
$ oc exec -it cephfs-write-workload-generator-no-cache-1-cv892 -- ls -latrZ /mnt/pv/

total 567
drwxrwxrwx. 3 root      root system_u:object_r:container_file_t:s0:c26,c5   2 May 25 06:35 .
-rw-r--r--. 1 1000660000 root system_u:object_r:container_file_t:s0:c26,c5 580138 May 25 06:35 fs_write_cephfs-write-workload-generator-no-cache-1-cv892-data.log
drwxrwxrwx. 3 root      root system_u:object_r:container_file_t:s0:c26,c5   30 May 25 06:35 ..
```

7. Get the information of the legacy application RWX PV that you want to make accessible from the **openshift-storage** namespace:

```
$ oc get pv | grep <pv_name>
```

<pv_name>

Specify the name of the PV.

For example:

```
$ oc get pv | grep pvc-aa58fb91-c3d2-475b-bbee-68452a613e1a

pvc-aa58fb91-c3d2-475b-bbee-68452a613e1a 10Gi      RWX          Delete
Bound testnamespace/cephfs-write-workload-generator-no-cache-pv-claim ocs-storagecluster-cephfs 47s
```

8. Ensure that the PVC from the legacy application is accessible from the **openshift-storage** namespace so that one or more noobaa-endpoint pods can access the PVC.
 - a. Find the values of the **subvolumePath** and **volumeHandle** from the **volumeAttributes**. You can get these values from the YAML description of the legacy application PV:

```
$ oc get pv <pv_name> -o yaml
```

For example:

```
$ oc get pv pvc-aa58fb91-c3d2-475b-bbee-68452a613e1a -o yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: openshift-storage.cephfs.csi.ceph.com
```

```

creationTimestamp: "2022-05-25T06:27:49Z"
finalizers:
- kubernetes.io/pv-protection
name: pvc-aa58fb91-c3d2-475b-bbee-68452a613e1a
resourceVersion: "177458"
uid: 683fa87b-5192-4ccf-af2f-68c6bcf8f500
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: cephfs-write-workload-generator-no-cache-pv-claim
    namespace: testnamespace
    resourceVersion: "177453"
    uid: aa58fb91-c3d2-475b-bbee-68452a613e1a
  csi:
    controllerExpandSecretRef:
      name: rook-csi-cephfs-provisioner
      namespace: openshift-storage
    driver: openshift-storage.cephfs.csi.ceph.com
    nodeStageSecretRef:
      name: rook-csi-cephfs-node
      namespace: openshift-storage
    volumeAttributes:
      clusterID: openshift-storage
      fsName: ocs-storagecluster-cephfilesystem
      storage.kubernetes.io/csiProvisionerIdentity: 1653458225664-8081-openshift-
storage.cephfs.csi.ceph.com
      subvolumeName: csi-vol-cc416d9e-dbf3-11ec-b286-0a580a810213
      subvolumePath: /volumes/csi/csi-vol-cc416d9e-dbf3-11ec-b286-
0a580a810213/edcfe4d5-bdcb-4b8e-8824-8a03ad94d67c
      volumeHandle: 0001-0011-openshift-storage-0000000000000001-cc416d9e-dbf3-
11ec-b286-0a580a810213
    persistentVolumeReclaimPolicy: Delete
    storageClassName: ocs-storagecluster-cephfs
    volumeMode: Filesystem
status:
  phase: Bound

```

- b. Use the **subvolumePath** and **volumeHandle** values that you identified in the previous step to create a new PV and PVC object in the **openshift-storage** namespace that points to the same CephFS volume as the legacy application PV:

Example YAML file:

```

$ cat << EOF >> pv-openshift-storage.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cephfs-pv-legacy-openshift-storage
spec:
  storageClassName: ""
  accessModes:
  - ReadWriteMany

```

```

capacity:
  storage: 10Gi ❶
csi:
  driver: openshift-storage.cephfs.csi.ceph.com
  nodeStageSecretRef:
    name: rook-csi-cephfs-node
    namespace: openshift-storage
  volumeAttributes:
    # Volume Attributes can be copied from the Source testnamespace PV
    "clusterID": "openshift-storage"
    "fsName": "ocs-storagecluster-cephfilesystem"
    "staticVolume": "true"
    # rootpath is the subvolumePath: you copied from the Source testnamespace PV
    "rootPath": /volumes/csi/csi-vol-cc416d9e-dbf3-11ec-b286-0a580a810213/edcfe4d5-
bdc4-4b8e-8824-8a03ad94d67c
    volumeHandle: 0001-0011-openshift-storage-0000000000000001-cc416d9e-dbf3-
11ec-b286-0a580a810213-clone ❷
    persistentVolumeReclaimPolicy: Retain
    volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cephfs-pvc-legacy
  namespace: openshift-storage
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi ❸
    volumeMode: Filesystem
    # volumeName should be same as PV name
    volumeName: cephfs-pv-legacy-openshift-storage
EOF

```

- ❶ The storage capacity of the PV that you are creating in the **openshift-storage** namespace must be the same as the original PV.
- ❷ The volume handle for the target PV that you create in **openshift-storage** needs to have a different handle than the original application PV, for example, add **-clone** at the end of the volume handle.
- ❸ The storage capacity of the PVC that you are creating in the **openshift-storage** namespace must be the same as the original PVC.

- c. Create the PV and PVC in the **openshift-storage** namespace using the YAML file specified in the previous step:

```
$ oc create -f <YAML_file>
```

<YAML_file>

Specify the name of the YAML file.

For example:

```
$ oc create -f pv-openshift-storage.yaml

persistentvolume/cephfs-pv-legacy-openshift-storage created
persistentvolumeclaim/cephfs-pvc-legacy created
```

- d. Ensure that the PVC is available in the **openshift-storage** namespace:

```
$ oc get pvc -n openshift-storage
```

NAME	STATUS	VOLUME	CAPACITY
ACCESS MODES	STORAGECLASS	AGE	
cephfs-pvc-legacy	Bound	cephfs-pv-legacy-openshift-storage	10Gi
RWX	14s		

- e. Navigate into the **openshift-storage** project:

```
$ oc project openshift-storage
```

Now using project "openshift-storage" on server "https://api.cluster-5f6ng.5f6ng.sandbox65.opentlc.com:6443".

- f. Create the NSFS namespacestore:

```
$ noobaa namespacestore create nsfs <nsfs_namespacestore> --pvc-
name='<cephfs_pvc_name>' --fs-backend='CEPH_FS'
```

<nsfs_namespacestore>

Specify the name of the NSFS namespacestore.

<cephfs_pvc_name>

Specify the name of the CephFS PVC in the **openshift-storage** namespace.

For example:

```
$ noobaa namespacestore create nsfs legacy-namespace --pvc-name='cephfs-pvc-
legacy' --fs-backend='CEPH_FS'
```

- g. Ensure that the noobaa-endpoint pod restarts and that it successfully mounts the PVC at the NSFS namespacestore, for example, **/nsfs/legacy-namespace** mountpoint:

```
$ oc exec -it <noobaa_endpoint_pod_name> -- df -h /nsfs/<nsfs_namespacestore>
```

<noobaa_endpoint_pod_name>

Specify the name of the noobaa-endpoint pod.

For example:

```
$ oc exec -it noobaa-endpoint-5875f467f5-546c6 -- df -h /nsfs/legacy-namespace
```

Filesystem	Size	Used	Avail	Use%	Mounted on

```
172.30.202.87:6789,172.30.120.254:6789,172.30.77.247:6789:/volumes/csi/csi-vol-cc416d9e-dbf3-11ec-b286-0a580a810213/edcfe4d5-bdcb-4b8e-8824-8a03ad94d67c
10G  0 10G  0% /nsfs/legacy-namespace
```

- h. Create a MCG user account:

```
$ noobaa account create <user_account> --full_permission --allow_bucket_create=true -
--new_buckets_path="/" --nsfs_only=true --nsfs_account_config=true --gid <gid_number>
--uid <uid_number> --default_resource='legacy-namespace'
```

<user_account>

Specify the name of the MCG user account.

<gid_number>

Specify the GID number.

<uid_number>

Specify the UID number.



IMPORTANT

Use the same **UID** and **GID** as that of the legacy application. You can find it from the previous output.

For example:

```
$ noobaa account create leguser --full_permission --allow_bucket_create=true --
new_buckets_path="/" --nsfs_only=true --nsfs_account_config=true --gid 0 --uid
1000660000 --default_resource='legacy-namespace'
```

- i. Create a MCG bucket.

- i. Create a dedicated folder for S3 inside the NSFS share on the CephFS PV and PVC of the legacy application pod:

```
$ oc exec -it <pod_name> -- mkdir <mount_path>/nsfs
```

For example:

```
$ oc exec -it cephfs-write-workload-generator-no-cache-1-cv892 -- mkdir
/mnt/pv/nsfs
```

- ii. Create the MCG bucket using the **nsfs/** path:

```
$ noobaa api bucket_api create_bucket '{
  "name": "<bucket_name>",
  "namespace":{
    "write_resource": { "resource": "<nsfs_namespacestore>", "path": "nsfs/" },
    "read_resources": [ { "resource": "<nsfs_namespacestore>", "path": "nsfs/" } ]
  }
}'
```

For example:

```
$ noobaa api bucket_api create_bucket '{
  "name": "legacy-bucket",
  "namespace": {
    "write_resource": { "resource": "legacy-namespace", "path": "nsfs/" },
    "read_resources": [ { "resource": "legacy-namespace", "path": "nsfs/" } ]
  }
}'
```

- j. Check the SELinux labels of the folders residing in the PVCs in the legacy application and **openshift-storage** namespaces:

```
$ oc exec -it <noobaa_endpoint_pod_name> -n openshift-storage -- ls -ltrZ
/nsfs/<nsfs_namespacstore>
```

For example:

```
$ oc exec -it noobaa-endpoint-5875f467f5-546c6 -n openshift-storage -- ls -ltrZ
/nsfs/legacy-namespace
```

```
total 567
drwxrwxrwx. 3 root    root system_u:object_r:container_file_t:s0:c0,c26   2 May 25
06:35 .
-rw-r--r--. 1 1000660000 root system_u:object_r:container_file_t:s0:c0,c26 580138 May
25 06:35 fs_write_cephfs-write-workload-generator-no-cache-1-cv892-data.log
drwxrwxrwx. 3 root    root system_u:object_r:container_file_t:s0:c0,c26   30 May 25
06:35 ..
```

```
$ oc exec -it <pod_name> -- ls -ltrZ <mount_path>
```

For example:

```
$ oc exec -it cephfs-write-workload-generator-no-cache-1-cv892 -- ls -ltrZ /mnt/pv/

total 567
drwxrwxrwx. 3 root    root system_u:object_r:container_file_t:s0:c26,c5   2 May 25
06:35 .
-rw-r--r--. 1 1000660000 root system_u:object_r:container_file_t:s0:c26,c5 580138 May
25 06:35 fs_write_cephfs-write-workload-generator-no-cache-1-cv892-data.log
drwxrwxrwx. 3 root    root system_u:object_r:container_file_t:s0:c26,c5   30 May 25
06:35 ..
```

In these examples, you can see that the SELinux labels are not the same which results in permission denied or access issues.

9. Ensure that the legacy application and **openshift-storage** pods use the same SELinux labels on the files.

You can do this in one of the following ways:

- [Section 4.4.3.1, “Changing the default SELinux label on the legacy application project to match the one in the openshift-storage project”](#).

- [Section 4.4.3.2, “Modifying the SELinux label only for the deployment config that has the pod which mounts the legacy application PVC”](#).

10. Delete the NSFS namespacestore:

a. Delete the MCG bucket:

```
$ noobaa bucket delete <bucket_name>
```

For example:

```
$ noobaa bucket delete legacy-bucket
```

b. Delete the MCG user account:

```
$ noobaa account delete <user_account>
```

For example:

```
$ noobaa account delete leguser
```

c. Delete the NSFS namespacestore:

```
$ noobaa namespacestore delete <nsfs_namespacestore>
```

For example:

```
$ noobaa namespacestore delete legacy-namespace
```

11. Delete the PV and PVC:



IMPORTANT

Before you delete the PV and PVC, ensure that the PV has a retain policy configured.

```
$ oc delete pv <cephfs_pv_name>
```

```
$ oc delete pvc <cephfs_pvc_name>
```

<cephfs_pv_name>

Specify the CephFS PV name of the legacy application.

<cephfs_pvc_name>

Specify the CephFS PVC name of the legacy application.

For example:

```
$ oc delete pv cephfs-pv-legacy-openshift-storage
```

```
$ oc delete pvc cephfs-pvc-legacy
```

4.4.3.1. Changing the default SELinux label on the legacy application project to match the one in the openshift-storage project

1. Display the current **openshift-storage** namespace with **sa.scc.mcs**:

```
$ oc get ns openshift-storage -o yaml | grep sa.scc.mcs
openshift.io/sa.scc.mcs: s0:c26,c0
```

2. Edit the legacy application namespace, and modify the **sa.scc.mcs** with the value from the **sa.scc.mcs** of the **openshift-storage** namespace:

```
$ oc edit ns <application_namespace>
```

For example:

```
$ oc edit ns testnamespace
```

```
$ oc get ns <application_namespace> -o yaml | grep sa.scc.mcs
```

For example:

```
$ oc get ns testnamespace -o yaml | grep sa.scc.mcs
openshift.io/sa.scc.mcs: s0:c26,c0
```

3. Restart the legacy application pod. A relabel of all the files take place and now the SELinux labels match with the **openshift-storage** deployment.

4.4.3.2. Modifying the SELinux label only for the deployment config that has the pod which mounts the legacy application PVC

1. Create a new **scc** with the **MustRunAs** and **seLinuxOptions** options, with the Multi Category Security (MCS) that the **openshift-storage** project uses.

Example YAML file:

```
$ cat << EOF >> scc.yaml
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups:
- system:authenticated
kind: SecurityContextConstraints
metadata:
```

```

  annotations:
    name: restricted-pvselinux
  priority: null
  readOnlyRootFilesystem: false
  requiredDropCapabilities:
  - KILL
  - MKNOD
  - SETUID
  - SETGID
  runAsUser:
    type: MustRunAsRange
  seLinuxContext:
    seLinuxOptions:
      level: s0:c26,c0
    type: MustRunAs
  supplementalGroups:
    type: RunAsAny
  users: []
  volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
EOF

```

```
$ oc create -f scc.yaml
```

2. Create a service account for the deployment and add it to the newly created **scc**.

- a. Create a service account:

```
$ oc create serviceaccount <service_account_name>
```

<service_account_name>`

Specify the name of the service account.

For example:

```
$ oc create serviceaccount testnamespacesa
```

- b. Add the service account to the newly created **scc**:

```
$ oc adm policy add-scc-to-user restricted-pvselinux -z <service_account_name>
```

For example:

```
$ oc adm policy add-scc-to-user restricted-pvselinux -z testnamespacesa
```

3. Patch the legacy application deployment so that it uses the newly created service account. This allows you to specify the SELinux label in the deployment:

```
$ oc patch dc/<pod_name> '{"spec":{"template":{"spec":{"serviceAccountName":
"<service_account_name>"}}}}'
```

For example:

```
$ oc patch dc/cephfs-write-workload-generator-no-cache --patch '{"spec":{"template":{"spec":
{"serviceAccountName": "testnamespacesa"}}}}'
```

4. Edit the deployment to specify the security context to use at the SELinux label in the deployment configuration:

```
$ oc edit dc <pod_name> -n <application_namespace>
```

Add the following lines:

```
spec:
  template:
    metadata:
      securityContext:
        seLinuxOptions:
          Level: <security_context_value>
```

<security_context_value>

You can find this value when you execute the command to create a dedicated folder for S3 inside the NSFS share, on the CephFS PV and PVC of the legacy application pod.

For example:

```
$ oc edit dc cephfs-write-workload-generator-no-cache -n testnamespace
```

```
spec:
  template:
    metadata:
      securityContext:
        seLinuxOptions:
          level: s0:c26,c0
```

5. Ensure that the security context to be used at the SELinux label in the deployment configuration is specified correctly:

```
$ oc get dc <pod_name> -n <application_namespace> -o yaml | grep -A 2 securityContext
```

For example"

```
$ oc get dc cephfs-write-workload-generator-no-cache -n testnamespace -o yaml | grep -A 2
securityContext
```

```
securityContext:
  seLinuxOptions:
    level: s0:c26,c0
```

The legacy application is restarted and begins using the same SELinux labels as the **openshift-storage** namespace.

CHAPTER 5. SECURING MULTICLOUD OBJECT GATEWAY

5.1. CHANGING THE DEFAULT ACCOUNT CREDENTIALS TO ENSURE BETTER SECURITY IN THE MULTICLOUD OBJECT GATEWAY

Change and rotate your Multicloud Object Gateway (MCG) account credentials using the command-line interface to prevent issues with applications, and to ensure better account security.

5.1.1. Resetting the noobaa account password

Prerequisites

- A running OpenShift Data Foundation cluster.
- Download the Multicloud Object Gateway (MCG) command-line interface for easier management. For instructions, see [Accessing the Multicloud Object Gateway with your applications](#).

Procedure

- To reset the noobaa account password, run the following command:

```
$ noobaa account passwd <noobaa_account_name> [options]
```

```
$ noobaa account passwd
FATA[0000] Missing expected arguments: <noobaa_account_name>
```

Options:

--new-password="": New Password for authentication - the best practice is to omit this flag, in that case the CLI will prompt to prompt and read it securely from the terminal to avoid leaking secrets in the shell history

--old-password="": Old Password for authentication - the best practice is to omit this flag, in that case the CLI will prompt to prompt and read it securely from the terminal to avoid leaking secrets in the shell history

--retype-new-password="": Retype new Password for authentication - the best practice is to omit this flag, in that case the CLI will prompt to prompt and read it securely from the terminal to avoid leaking secrets in the shell history

Usage:

```
noobaa account passwd <noobaa-account-name> [flags] [options]
```

Usage:

```
noobaa account passwd <noobaa-account-name> [flags] [options]
```

Use "noobaa options" for a list of global command-line options (applies to all commands).

Example:

```
$ noobaa account passwd admin@noobaa.io
```

Example output:

```
Enter old-password: [got 24 characters]
Enter new-password: [got 7 characters]
Enter retype-new-password: [got 7 characters]
INFO[0017] Exists: Secret "noobaa-admin"
INFO[0017] Exists: NooBaa "noobaa"
INFO[0017] Exists: Service "noobaa-mgmt"
INFO[0017] Exists: Secret "noobaa-operator"
INFO[0017] Exists: Secret "noobaa-admin"
INFO[0017] ➔ RPC: account.reset_password() Request: {Email:admin@noobaa.io
VerificationPassword:* Password:*}
WARN[0017] RPC: GetConnection creating connection to wss://localhost:58460/rpc/
0xc000402ae0
INFO[0017] RPC: Connecting websocket (0xc000402ae0) &{RPC:0xc000501a40
Address:wss://localhost:58460/rpc/ State:init WS:<nil> PendingRequests:map[]
NextRequestID:0
Lock:{state:1 sema:0} ReconnectDelay:0s cancelPings:<nil>}
INFO[0017] RPC: Connected websocket (0xc000402ae0) &{RPC:0xc000501a40
Address:wss://localhost:58460/rpc/ State:init WS:<nil> PendingRequests:map[]
NextRequestID:0
Lock:{state:1 sema:0} ReconnectDelay:0s cancelPings:<nil>}
INFO[0020] RPC: account.reset_password() Response OK: took 2907.1ms
INFO[0020] Updated: "noobaa-admin"
INFO[0020] Successfully reset the password for the account "admin@noobaa.io"
```

IMPORTANT

To access the admin account credentials run the **noobaa status** command from the terminal:

```
-----
- Mgmt Credentials -
-----

email   : admin@noobaa.io
password : ***
```

5.1.2. Regenerating the S3 credentials for the accounts

Prerequisites

- A running OpenShift Data Foundation cluster.
- Download the Multicloud Object Gateway (MCG) command-line interface for easier management. For instructions, see [Accessing the Multicloud Object Gateway with your applications](#).

Procedure

1. Get the account name.
For listing the accounts, run the following command:

```
$ noobaa account list
```

Example output:

```
NAME          ALLOWED_BUCKETS  DEFAULT_RESOURCE  PHASE  AGE
account-test  [*]             noobaa-default-backing-store  Ready  14m17s
test2        [first.bucket]  noobaa-default-backing-store  Ready  3m12s
```

Alternatively, run the **oc get noobaaaccount** command from the terminal:

```
$ oc get noobaaaccount
```

Example output:

```
NAME      PHASE  AGE
account-test  Ready  15m
test2       Ready  3m59s
```

- To regenerate the noobaa account S3 credentials, run the following command:

```
$ noobaa account regenerate <noobaa_account_name> [options]
```

```
$ noobaa account regenerate
FATA[0000] Missing expected arguments: <noobaa-account-name>
```

Usage:

```
noobaa account regenerate <noobaa-account-name> [flags] [options]
```

Use "noobaa options" for a list of global command-line options (applies to all commands).

- Once you run the **noobaa account regenerate** command it will prompt a warning that says **"This will invalidate all connections between S3 clients and NooBaa which are connected using the current credentials."**, and ask for confirmation:

Example:

```
$ noobaa account regenerate account-test
```

Example output:

```
INFO[0000] You are about to regenerate an account's security credentials.
INFO[0000] This will invalidate all connections between S3 clients and NooBaa which are
connected using the current credentials.
INFO[0000] are you sure? y/n
```

- On approving, it will regenerate the credentials and eventually print them:

```
INFO[0015] Exists: Secret "noobaa-account-account-test"
Connection info:
AWS_ACCESS_KEY_ID    : ***
AWS_SECRET_ACCESS_KEY : ***
```


5.1.3. Regenerating the S3 credentials for the OBC

Prerequisites

- A running OpenShift Data Foundation cluster.
- Download the Multicloud Object Gateway (MCG) command-line interface for easier management. For instructions, see [Accessing the Multicloud Object Gateway with your applications](#).

Procedure

1. To get the OBC name, run the following command:

```
$ noobaa obc list
```

Example output:

```

NAMESPACE NAME      BUCKET-NAME                                STORAGE-CLASS
BUCKET-CLASS      PHASE
default  obc-test  obc-test-35800e50-8978-461f-b7e0-7793080e26ba  default.noobaa.io
noobaa-default-bucket-class  Bound

```

Alternatively, run the **oc get obc** command from the terminal:

```
$ oc get obc
```

Example output:

```

NAME      STORAGE-CLASS  PHASE  AGE
obc-test  default.noobaa.io  Bound  38s

```

2. To regenerate the noobaa OBC S3 credentials, run the following command:

```
$ noobaa obc regenerate <bucket_claim_name> [options]
```

```
$ noobaa obc regenerate
FATA[0000] Missing expected arguments: <bucket-claim-name>
```

Usage:

```
noobaa obc regenerate <bucket-claim-name> [flags] [options]
```

Use "noobaa options" for a list of global command-line options (applies to all commands).

3. Once you run the **noobaa obc regenerate** command it will prompt a warning that says **"This will invalidate all connections between the S3 clients and noobaa which are connected using the current credentials."**, and ask for confirmation:

Example:

```
$ noobaa obc regenerate obc-test
```

Example output:

```
-
```

```
INFO[0000] You are about to regenerate an OBC's security credentials.
INFO[0000] This will invalidate all connections between S3 clients and NooBaa which are
connected using the current credentials.
INFO[0000] are you sure? y/n
```

4. On approving, it will regenerate the credentials and eventually print them:

```
INFO[0022] RPC: bucket.read_bucket() Response OK: took 95.4ms

ObjectBucketClaim info:
Phase           : Bound
ObjectBucketClaim : kubectl get -n default objectbucketclaim obc-test
ConfigMap        : kubectl get -n default configmap obc-test
Secret           : kubectl get -n default secret obc-test
ObjectBucket      : kubectl get objectbucket obc-default-obc-test
StorageClass      : kubectl get storageclass default.noobaa.io
BucketClass       : kubectl get -n default bucketclass noobaa-default-bucket-class

Connection info:
BUCKET_HOST       : s3.default.svc
BUCKET_NAME       : obc-test-35800e50-8978-461f-b7e0-7793080e26ba
BUCKET_PORT       : 443
AWS_ACCESS_KEY_ID : ***
AWS_SECRET_ACCESS_KEY : ***

Shell commands:
AWS S3 Alias       : alias s3='AWS_ACCESS_KEY_ID=***
AWS_SECRET_ACCESS_KEY=*** aws s3 --no-verify-ssl --endpoint-url ***'

Bucket status:
Name               : obc-test-35800e50-8978-461f-b7e0-7793080e26ba
Type               : REGULAR
Mode               : OPTIMAL
ResiliencyStatus   : OPTIMAL
QuotaStatus        : QUOTA_NOT_SET
Num Objects        : 0
Data Size          : 0.000 B
Data Size Reduced  : 0.000 B
Data Space Avail   : 13.261 GB
Num Objects Avail  : 9007199254740991
```

5.2. ENABLING SECURED MODE DEPLOYMENT FOR MULTICLOUD OBJECT GATEWAY

You can specify a range of IP addresses that should be allowed to reach the Multicloud Object Gateway (MCG) load balancer services to enable secure mode deployment. This helps to control the IP addresses that can access the MCG services.

Prerequisites

- A running OpenShift Data Foundation cluster.
- In case of a bare metal deployment, ensure that the load balancer controller supports setting the **loadBalancerSourceRanges** attribute in the Kubernetes services.

Procedure

- Edit the NooBaa custom resource (CR) to specify the range of IP addresses that can access the MCG services after deploying OpenShift Data Foundation.

```
$ oc edit noobaa -n openshift-storage noobaa
```

noobaa

The NooBaa CR type that controls the NooBaa system deployment.

noobaa

The name of the NooBaa CR.

For example:

```
...
spec:
  ...
  loadBalancerSourceSubnets:
    s3: ["10.0.0.0/16", "192.168.10.0/32"]
    sts:
      - "10.0.0.0/16"
      - "192.168.10.0/32"
  ...
```

loadBalancerSourceSubnets

A new field that can be added under **spec** in the NooBaa CR to specify the IP addresses that should have access to the NooBaa services.

In this example, all the IP addresses that are in the subnet 10.0.0.0/16 or 192.168.10.0/32 will be able to access MCG S3 and security token service (STS) while the other IP addresses are not allowed to access.

Verification steps

- To verify if the specified IP addresses are set, in the OpenShift Web Console, run the following command and check if the output matches with the IP addresses provided to MCG:

```
$ oc get svc -n openshift-storage <s3 | sts> -o=go-template='{{
.spec.loadBalancerSourceRanges }}'
```

CHAPTER 6. MIRRORING DATA FOR HYBRID AND MULTICLOUD BUCKETS

You can use the simplified process of the Multicloud Object Gateway (MCG) to span data across cloud providers and clusters. Before you create a bucket class that reflects the data management policy and mirroring, you must add a backing storage that can be used by the MCG. For information, see Chapter 4, [Chapter 3, *Adding storage resources for hybrid or Multicloud*](#).

You can set up mirroring data by using the OpenShift UI, YAML or MCG command-line interface.

See the following sections:

- [Section 6.2, *Section 6.1, "Creating bucket classes to mirror data using the MCG command-line-interface"*](#)
- [Section 6.3, *Section 6.2, "Creating bucket classes to mirror data using a YAML"*](#)

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

Prerequisites

- Ensure to download Multicloud Object Gateway (MCG) command-line interface.

Procedure

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 to generate 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:  
  - <backing-store-1>  
  - <backing-store-2>  
placement: Mirror
```

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

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

For more information about OBCs, see [Chapter 9, Object Bucket Claim](#).

CHAPTER 7. BUCKET POLICIES IN THE MULTICLOUD OBJECT GATEWAY

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. INTRODUCTION TO 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 IN 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

To use bucket policies in the MCG:

1. Create the bucket policy in JSON format.
For example:

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

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.

Additional resources

- 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](#).

7.3. CREATING A USER IN THE MULTICLOUD OBJECT GATEWAY

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.

Procedure

Execute the following command to create an MCG user account:

```
noobaa account create <noobaa-account-name> [--allow_bucket_create=true] [--allowed_buckets=[]]
[--default_resource=""] [--full_permission=false]
```

<noobaa-account-name>

Specify the name of the new MCG user account.

--allow_bucket_create

Allows the user to create new buckets.

--allowed_buckets

Sets the user's allowed bucket list (use commas or multiple flags).

--default_resource

Sets the default resource. The new buckets are created on this default resource (including the future ones).

--full_permission

Allows this account to access all existing and future buckets.

**IMPORTANT**

You need to provide permission to access at least one bucket or full permission to access all the buckets.

CHAPTER 8. MULTICLOUD OBJECT GATEWAY BUCKET REPLICATION

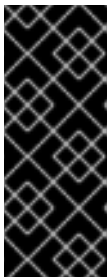
Data replication from one Multicloud Object Gateway (MCG) bucket to another MCG bucket provides higher resiliency and better collaboration options. These buckets can be either data buckets or namespace buckets backed by any supported storage solution (S3, Azure, etc.).

A 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 the second bucket results in bidirectional replication.

Prerequisites

- A running OpenShift Data Foundation Platform.
- Access to the Multicloud Object Gateway, see [link:Accessing the Multicloud Object Gateway with your applications](#).
- 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
```

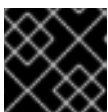


IMPORTANT

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

```
# 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:

```
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:

+

```
[[{"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": ""}**.

For example:

```
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:

```
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": "<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
- Namespace

Procedure

- 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:

```
[{"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": ""}`.

For 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:

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: <desired-app-label>
    name: <desired-bucketclass-name>
    namespace: <desired-namespace>
spec:
  placementPolicy:
    tiers:
      - backingstores:
          - <backingstore>
        placement: Spread
    replicationPolicy: [{"rule_id": "<rule id>", "destination_bucket": "first.bucket", "filter": {"prefix":
"<object name prefix>"}}]
```

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": ""}`.

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.



NOTE

The Multicloud Object Gateway endpoints uses self-signed certificates only if OpenShift uses self-signed certificates. Using signed certificates in OpenShift automatically replaces the Multicloud Object Gateway endpoints certificates with signed certificates. Get the certificate currently used by Multicloud Object Gateway by accessing the endpoint via the browser. See [Accessing the Multicloud Object Gateway with your applications](#) for more information.

Procedure

1. Add the following lines to your application YAML:

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

These lines are the 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. To automate the use of the OBC add more lines to the YAML file.
For example:

```
apiVersion: batch/v1
kind: Job
```

```

metadata:
  name: testjob
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - image: <your application image>
        name: test
        env:
        - name: BUCKET_NAME
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_NAME
        - name: BUCKET_HOST
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_HOST
        - name: BUCKET_PORT
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_PORT
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: <obc-name>
              key: AWS_ACCESS_KEY_ID
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: <obc-name>
              key: AWS_SECRET_ACCESS_KEY

```

The example 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.

- 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:

```
# 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:

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

IMPORTANT

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

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

<obc_name>

Specify the name of the object bucket claim.

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**.

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**.

For example:

```
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
```

For example:

```
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
```

For example:

```
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:

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

For example:

```

apiVersion: v1
data:
  AWS_ACCESS_KEY_ID: c0M0R2xVanF3ODR3bHBkVW94cmY=
  AWS_SECRET_ACCESS_KEY:
Wi9kcFluSWxHRzIWaFlzNk1hc0xma2JXcjM1MVhq051SIBleXpmOQ==
kind: Secret
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
    - objectbucket.io/finalizer
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
name: test21obc
namespace: openshift-storage
ownerReferences:
  - apiVersion: objectbucket.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ObjectBucketClaim
    name: test21obc
    uid: 64f04cba-f662-11e9-bc3c-0295250841af
resourceVersion: "40751"
selfLink: /api/v1/namespaces/openshift-storage/secrets/test21obc
uid: 65117c1c-f662-11e9-9094-0a5305de57bb
type: Opaque

```

The secret gives you the S3 access credentials.

5. Run the following command to view the configuration map:

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

For example:

```
apiVersion: v1
data:
  BUCKET_HOST: 10.0.171.35
  BUCKET_NAME: test21obc-933348a6-e267-4f82-82f1-e59bf4fe3bb4
  BUCKET_PORT: "31242"
  BUCKET_REGION: ""
  BUCKET_SUBREGION: ""
kind: ConfigMap
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
    - objectbucket.io/finalizer
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  ownerReferences:
    - apiVersion: objectbucket.io/v1alpha1
      blockOwnerDeletion: true
      controller: true
      kind: ObjectBucketClaim
      name: test21obc
      uid: 64f04cba-f662-11e9-bc3c-0295250841af
  resourceVersion: "40752"
  selfLink: /api/v1/namespaces/openshift-storage/configmaps/test21obc
  uid: 651c6501-f662-11e9-9094-0a5305de57bb
```

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

9.3. CREATING AN OBJECT BUCKET CLAIM USING THE OPENSIFT WEB CONSOLE

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

Prerequisites

- Administrative access to the OpenShift Web Console.
- In order for your applications to communicate with the OBC, you need to use the configmap and secret. For more information about this, see [Section 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.

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**.

9.5. VIEWING OBJECT BUCKETS USING THE OPENSIFT WEB CONSOLE

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

Prerequisites

- Administrative access to the OpenShift Web Console.

Procedure


1. Log into the OpenShift Web Console.
2. On the left navigation bar, click **Storage → Object Buckets**.
Optional: 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 of which you want to see the details. Once selected you are navigated to the **Object Bucket Details** page.

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**.

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.



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:

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

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

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

Then apply the following YAML:

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

- d. Replace **<namespacestore>** with a unique name.
 - e. Replace **<namespacestore-secret-name>** with the secret created in the previous step.
 - f. Replace **<namespace-secret>** with the namespace used to create the secret in the previous step.
 - g. Replace **<target-bucket>** with the AWS S3 bucket you created for the namespacestore.
2. Run the following command to create a bucket class:

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

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

- b. Replace **<backing-store>** with the relevant backing store. You can list one or more backingstores separated by commas in this field.
 - c. Replace **<namespacestore>** with the namespacestore created in the previous step.
3. Run the following command to create a bucket using an Object Bucket Claim (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.



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 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:

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

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

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

Then apply the following YAML:

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

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

- e. Replace **<IBM COS ENDPOINT>** with the appropriate IBM COS endpoint.
 - f. Replace **<backingstore-secret-name>** with the secret created in the previous step.
 - g. Replace **<namespace-secret>** with the namespace used to create the secret in the previous step.
 - h. Replace **<target-bucket>** with the AWS S3 bucket you created for the namespacestore.
2. Run the following command to create a bucket class:

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

- a. Replace **<my-bucket-class>** with a unique bucket class name.
 - b. Replace **<backing-store>** with the relevant backing store. You can list one or more backingstores separated by commas in this field.
 - c. Replace **<namespacestore>** with the namespacestore created in the previous step.
3. Run the following command to create a bucket using an Object Bucket Claim resource that uses the bucket class defined in step 2.

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

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

CHAPTER 11. SCALING MULTICLOUD OBJECT GATEWAY PERFORMANCE

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**.

11.3. INCREASING CPU AND MEMORY FOR PV POOL RESOURCES

MCG default configuration supports low resource consumption. However, when you need to increase CPU and memory to accommodate specific workloads and to increase MCG performance for the workloads, it is possible to configure the required values for CPU and memory in the OpenShift Web Console.

Procedure

1. In the OpenShift Web Console, click **Installed operators → ODF Operator**.
2. Click on the **Backingstore** tab.
3. Select the new **backingstore**.
4. Scroll down and click **Edit PV pool resources**.
5. In the edit window that appears, edit the value of **Mem**, **CPU**, and **Vol size** based on the requirement.
6. Click **Save**.

Verification steps

- To verify, you can check the resource values of the PV pool pods.

CHAPTER 12. ACCESSING THE RADOS OBJECT GATEWAY S3 ENDPOINT

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

In previous versions of Red Hat OpenShift Data Foundation, RGW service needed to be manually exposed to create RGW public route. As of OpenShift Data Foundation version 4.7, the RGW route is created by default and is named **rook-ceph-rgw-ocs-storagecluster-cephobjectstore**.