



Red Hat OpenShift Data Foundation 4.10

ハイブリッドおよびマルチクラウドリソースの管理

Multicloud Object Gateway (NooBaa) を使用したハイブリッドクラウドまたはマルチクラウド環境でのストレージリソースを管理する方法

Red Hat OpenShift Data Foundation 4.10 ハイブリッドおよびマルチクラウドリソースの管理

Multicloud Object Gateway (NooBaa) を使用したハイブリッドクラウドまたはマルチクラウド環境でのストレージリソースを管理する方法

法律上の通知

Copyright © 2024 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.

概要

本書では、ハイブリッドクラウドまたはマルチクラウド環境でストレージリソースを管理する方法について説明します。

目次

| | |
|--|----|
| 多様性を受け入れるオープンソースの強化 | 4 |
| RED HAT ドキュメントへのフィードバック (英語のみ) | 5 |
| 第1章 MULTICLOUD OBJECT GATEWAY について | 6 |
| 第2章 アプリケーションの使用による MULTICLOUD OBJECT GATEWAY へのアクセス | 7 |
| 2.1. ターミナルから MULTICLOUD OBJECT GATEWAY へのアクセス | 8 |
| 2.2. MCG コマンドラインインターフェイスからの MULTICLOUD OBJECT GATEWAY へのアクセス | 10 |
| 第3章 MULTICLOUD OBJECT GATEWAY コンソールへのユーザーアクセスの許可 | 14 |
| 第4章 ハイブリッドまたはマルチクラウド用のストレージリソースの追加 | 16 |
| 4.1. 新規バックキングストアの作成 | 16 |
| 4.2. MCG コマンドラインインターフェイスを使用したハイブリッドまたはマルチクラウドのストレージリソースの追加 | 17 |
| 4.3. S3 と互換性のある MULTICLOUD OBJECT GATEWAY バックキングストアの作成 | 26 |
| 4.4. ユーザーインターフェイスを使用したハイブリッドおよびマルチクラウドのストレージリソースの追加 | 28 |
| 4.5. 新規バケットクラスの作成 | 29 |
| 4.6. バケットクラスの編集 | 30 |
| 4.7. バケットクラスのバックキングストアの編集 | 31 |
| 第5章 NAMESPACE バケットの管理 | 33 |
| 5.1. NAMESPACE バケットのオブジェクトの AMAZON S3 API エンドポイント | 33 |
| 5.2. MULTICLOUD OBJECT GATEWAY CLI および YAML を使用した NAMESPACE バケットの追加 | 34 |
| 5.3. OPENSIFT CONTAINER PLATFORM ユーザーインターフェイスを使用した NAMESPACE バケットの追加 | 42 |
| 5.4. S3 プロトコルを使用してレガシーアプリケーションデータをクラウドネイティブアプリケーションと共有する | 44 |
| 第6章 マルチクラウドオブジェクトゲートウェイのセキュリティを強化するために、デフォルトのアカウント資格情報を変更する | 62 |
| 第7章 ハイブリッドおよびマルチクラウドバケットのデータのミラーリング | 63 |
| 7.1. MCG コマンドラインインターフェイスを使用したデータのミラーリング用のバケットクラスの作成 | 63 |
| 7.2. YAML を使用したデータのミラーリング用のバケットクラスの作成 | 63 |
| 7.3. ユーザーインターフェイスを使用したデータミラーリングを行うためのバケットの設定 | 64 |
| 第8章 MULTICLOUD OBJECT GATEWAY のバケットポリシー | 66 |
| 8.1. バケットポリシーについて | 66 |
| 8.2. バケットポリシーの使用 | 66 |
| 8.3. MULTICLOUD OBJECT GATEWAY での AWS S3 ユーザーの作成 | 67 |
| 第9章 MULTICLOUD OBJECT GATEWAY バケットレプリケーション | 70 |
| 9.1. バケットの別のバケットへの複製 | 70 |
| 9.2. バケットクラスのレプリケーションポリシーの設定 | 72 |
| 第10章 OBJECT BUCKET CLAIM(オブジェクトバケット要求) | 75 |
| 10.1. 動的 OBJECT BUCKET CLAIM(オブジェクトバケット要求) | 75 |
| 10.2. コマンドラインインターフェイスを使用した OBJECT BUCKET CLAIM(オブジェクトバケット要求) の作成 | 77 |
| 10.3. OPENSIFT WEB コンソールを使用した OBJECT BUCKET CLAIM(オブジェクトバケット要求) の作成 | 80 |
| 10.4. OBJECT BUCKET CLAIM(オブジェクトバケット要求) のデプロイメントへの割り当て | 83 |
| 10.5. OPENSIFT WEB コンソールを使用したオブジェクトバケットの表示 | 84 |

| | |
|---|-----------|
| 10.6. OBJECT BUCKET CLAIM(オブジェクトバケット要求) の削除 | 85 |
| 第11章 オブジェクトバケットのキャッシュポリシー | 86 |
| 11.1. AWS キャッシュバケットの作成 | 86 |
| 11.2. IBM COS キャッシュバケットの作成 | 88 |
| 第12章 エンドポイントの追加による MULTICLOUD OBJECT GATEWAY パフォーマンスのスケーリング | 91 |
| 12.1. MULTICLOUD OBJECT GATEWAY エンドポイントの自動スケーリング | 91 |
| 12.2. ストレージノードを使用した MULTICLOUD OBJECT GATEWAY のスケーリング | 91 |
| 第13章 RADOS OBJECT GATEWAY S3 エンドポイントへのアクセス | 93 |

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[Red Hat CTO である Chris Wright のメッセージ](#) をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

弊社のドキュメントについてのご意見をお聞かせください。ドキュメントの改善点があれば、ぜひお知らせください。

フィードバックを送信するには、Bugzilla チケットを作成します。

1. [Bugzilla](#) の Web サイトに移動します。
2. **Component** セクションで、**documentation** を選択します。
3. **Description** フィールドに、ドキュメントの改善に向けたご提案を記入してください。ドキュメントの該当部分へのリンクも追加してください。
4. **Submit Bug** をクリックします。

第1章 MULTICLOUD OBJECT GATEWAY について

Multicloud Object Gateway (MCG) は OpenShift の軽量オブジェクトストレージサービスであり、ユーザーは必要に応じて、複数のクラスター、およびクラウドネイティブストレージを使用して、オンプレミスで小規模に開始し、その後にスケーリングできます。

第2章 アプリケーションの使用による MULTICLOUD OBJECT GATEWAY へのアクセス

AWS S3 を対象とするアプリケーションまたは AWS S3 Software Development Kit(SDK) を使用するコードを使用して、オブジェクトサービスにアクセスできます。アプリケーションは、Multicloud Object Gateway (MCG) エンドポイント、アクセスキー、およびシークレットアクセスキーを指定する必要があります。ターミナルまたは MCG CLI を使用して、この情報を取得できます。

RADOS Object Gateway (RGW) S3 エンドポイントへのアクセスについては、[Accessing the RADOS Object Gateway S3 endpoint](#) を参照してください。

前提条件

- 実行中の OpenShift Data Foundation Platform。
- MCG コマンドラインインターフェイスをダウンロードして、管理を容易にします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。

- IBM Power の場合は、次のコマンドを使用します。

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

- IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

- または、MCG パッケージを、[Download RedHat OpenShift Data Foundation](#) ページにある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

関連するエンドポイント、アクセスキー、およびシークレットアクセスキーには、以下の2つの方法でアクセスできます。

- [「ターミナルから Multicloud Object Gateway へのアクセス」](#)
- [「MCG コマンドラインインターフェイスからの Multicloud Object Gateway へのアクセス」](#)

例2.1例

仮想ホストのスタイルを使用した MCG バケットへのアクセス

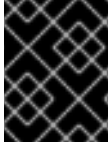
クライアントアプリケーションが `https://<bucket-name>.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com` にアクセスしようとする場合

<bucket-name>

MCG バケットの名前です。

たとえば、`https://mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com` になります。

DNS エントリーは、S3 サービスを参照するように、**`mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com`** が必要です。

**重要**

仮想ホストスタイルを使用してクライアントアプリケーションを MCG バケットを参照するように、DNS エントリーがあることを確認します。

2.1. ターミナルから MULTICLOUD OBJECT GATEWAY へのアクセス

手順

describe コマンドを実行し、アクセスキー (**`AWS_ACCESS_KEY_ID`** 値) およびシークレットアクセスキー (**`AWS_SECRET_ACCESS_KEY`** 値) を含む Multicloud Object Gateway (MCG) エンドポイントについての情報を表示します。

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

出力は以下のようになります。

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

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

Welcome to NooBaa!

NooBaa Core Version:

NooBaa Operator Version:

Lets get started:

1. Connect to Management console:

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

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

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

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

2. Test S3 client:

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

1

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

2

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

Services:

Service Mgmt:

External DNS:

```
https://noobaa-mgmt-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
```

```
https://a3406079515be11eaa3b70683061451e-1194613580.us-east-
```

2.elb.amazonaws.com:443

Internal DNS:

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

Internal IP:

```
https://172.30.235.12:443
```

Node Ports:

```
https://10.0.142.103:31385
```

Pod Ports:

```
https://10.131.0.19:8443
```

serviceS3:

External DNS: **3**

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

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

Internal DNS:

```
https://s3.openshift-storage.svc:443
```

Internal IP:

```
https://172.30.86.41:443
```

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

- 1 アクセスキー (AWS_ACCESS_KEY_ID 値)
- 2 シークレットアクセスキー (AWS_SECRET_ACCESS_KEY 値)
- 3 MCG エンドポイント

2.2. MCG コマンドラインインターフェイスからの MULTICLOUD OBJECT GATEWAY へのアクセス

前提条件

- MCG コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。

- IBM Power の場合は、次のコマンドを使用します。

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

- IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

手順

status コマンドを実行して、エンドポイント、アクセスキー、およびシークレットアクセスキーにアクセスします。

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

出力は以下のようになります。

```
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]

```

```

ExternallIP : []
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 エンドポイント
- 2 アクセスキー
- 3 シークレットアクセスキー

これで、アプリケーションに接続するための関連するエンドポイント、アクセスキー、およびシークレットアクセスキーを使用できます。

例2.2 例

AWS S3 CLI がアプリケーションである場合、以下のコマンドは OpenShift Data Foundation のバケットを一覧表示します。

```

AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY_ID>
AWS_SECRET_ACCESS_KEY=<AWS_SECRET_ACCESS_KEY>
aws --endpoint <ENDPOINT> --no-verify-ssl s3 ls

```


■

第3章 MULTICLOUD OBJECT GATEWAY コンソールへのユーザーアクセスの許可

ユーザーに Multicloud Object Gateway (MCG) コンソールへのアクセスを許可するには、ユーザーが以下の条件を満たしていることを確認してください。

- ユーザーは **cluster-admins** グループに属する。
- ユーザーは **system:cluster-admins** 仮想グループに属する。

前提条件

- 実行中の OpenShift Data Foundation Platform。

手順

1. MCG コンソールへのアクセスを有効にします。
クラスターで以下の手順を実行します。

- a. **cluster-admins** グループを作成します。

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

- b. グループを **cluster-admin** ロールにバインドします。

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

2. **cluster-admins** グループからユーザーを追加または削除して、MCG コンソールへのアクセスを制御します。

- ユーザーのセットを **cluster-admins** グループに追加するには、以下を実行します。

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

ここで、**<user-name>** は追加するユーザーの名前です。



注記

ユーザーのセットを **cluster-admins** グループに追加する場合、新たに追加されたユーザーを **cluster-admin** ロールにバインドし、OpenShift Data Foundation ダッシュボードへのアクセスを許可する必要はありません。

- ユーザーのセットを **cluster-admins** グループから削除するには、以下を実行します。

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

ここで、**<user-name>** は削除するユーザーの名前です。

検証手順

1. OpenShift Web コンソールで、Multicloud Object Gateway コンソールへのアクセスパーミッションを持つユーザーとしてログインします。

2. **Storage** → **Data Foundation** に移動します。
3. **Storage Systems** タブでストレージシステムを選択し、**Overview** → **Object** タブをクリックします。
4. **Multicloud Object Gateway** のリンクをクリックします。
5. **Allow selected permissions** をクリックします。

第4章 ハイブリッドまたはマルチクラウド用のストレージリソースの追加

4.1. 新規バックングストアの作成

以下の手順を使用して、OpenShift Data Foundation で新規のバックングストアを作成します。

前提条件

- OpenShift Data Foundation への管理者アクセス。

手順

1. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。
2. **Backing Store** タブをクリックします。
3. **Create Backing Store** をクリックします。
4. **Create New Backing Store** ページで、以下を実行します。
 - a. **Backing Store Name** を入力します。
 - b. **Provider** を選択します。
 - c. **Region** を選択します。
 - d. **Endpoint** を入力します。これは任意です。
 - e. ドロップダウンリストから **Secret** を選択するか、独自のシークレットを作成します。オプションで、**Switch to Credentials** ビューを選択すると、必要なシークレットを入力できます。
OCP シークレットの作成に関する詳細は、**Openshift Container Platform** ドキュメントの [Creating the secret](#) を参照してください。

バックングストアごとに異なるシークレットが必要です。特定のバックングストアのシークレット作成についての詳細は「[MCG コマンドラインインターフェイスを使用したハイブリッドまたはマルチクラウドのストレージリソースの追加](#)」を参照して、YAML を使用したストレージリソースの追加についての手順を実行します。



注記

このメニューは、Google Cloud およびローカル PVC 以外のすべてのプロバイダーに関連します。

- f. **Target bucket** を入力します。ターゲットバケットは、リモートクラウドサービスでホストされるコンテナストレージです。MCG に対してシステム用にこのバケットを使用できることを通知する接続を作成できます。
5. **Create Backing Store** をクリックします。

検証手順

1. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。

2. **Backing Store** タブをクリックして、すべてのバックングストアを表示します。

4.2. MCG コマンドラインインターフェイスを使用したハイブリッドまたはマルチクラウドのストレージリソースの追加

Multicloud Object Gateway (MCG) は、クラウドプロバイダーおよびクラスター全体にまたがるデータの処理を単純化します。

MCG で使用できるバックングストレージを追加します。

デプロイメントのタイプに応じて、以下のいずれかの手順を選択してバックングストレージを作成できます。

- AWS でサポートされるバックングストアを作成する方法については、「[AWS でサポートされるバックングストアの作成](#)」を参照してください。
- IBM COS でサポートされるバックングストアを作成する方法については、「[IBM COS でサポートされるバックングストアの作成](#)」を参照してください。
- Azure でサポートされるバックングストアを作成する方法については、「[Azure でサポートされるバックングストアの作成](#)」を参照してください。
- GCP でサポートされるバックングストアを作成する方法については、「[GCP でサポートされるバックングストアの作成](#)」を参照してください。
- ローカルの永続ボリュームでサポートされるバックングストアを作成する方法については、「[ローカル永続ボリュームでサポートされるバックングストアの作成](#)」を参照してください。

VMware デプロイメントの場合、「[s3 と互換性のある Multicloud Object Gateway バックングストアの作成](#)」に進み、詳細の手順を確認します。

4.2.1. AWS でサポートされるバックングストアの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

- または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/packages にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- a. **<backingstore_name>** を、バックアップストアの名前に置き換えます。
- b. **<AWS ACCESS KEY>** および **<AWS SECRET ACCESS KEY>** を、作成した AWS アクセスキー ID およびシークレットアクセスキーに置き換えます。
- c. **<bucket-name>** を既存の AWS バケット名に置き換えます。この引数は、MCG に対して、バックアップストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
出力は次のようになります。

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

YAML を使用してストレージリソースを追加することもできます。

1. 認証情報でシークレットを作成します。

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

- a. Base64 を使用して独自の AWS アクセスキー ID およびシークレットアクセスキーを指定し、エンコードし、その結果を **<AWS ACCESS KEY ID ENCODED IN BASE64>** および **<AWS SECRET ACCESS KEY ENCODED IN BASE64>** に使用する必要があります。
 - b. **<backingstore-secret-name>** を一意の名前に置き換えます。
2. 特定のバックアップストアについて以下の YAML を適用します。

```
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>** を既存の AWS バケット名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
- <backingstore-secret-name>** を直前の手順で作成したシークレットの名前に置き換えます。

4.2.2. IBM COS でサポートされるバックングストアの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

```

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

```



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、以下のようになります。

- IBM Power の場合は、次のコマンドを使用します。

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

- IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

- または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/packages にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

- MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- a. **<backingstore_name>** を、バックイングストアの名前に置き換えます。
- b. **<IBM ACCESS KEY>**, **<IBM SECRET ACCESS KEY>**, **<IBM COS ENDPOINT>** を IBM アクセスキー ID、シークレットアクセスキー、および既存の IBM バケットの場所に対応する地域のエンドポイントに置き換えます。
IBM クラウドで上記のキーを生成するには、ターゲットバケットのサービス認証情報を作成する際に HMAC 認証情報を含める必要があります。
- c. **<bucket-name>** を既存の IBM バケット名に置き換えます。この引数は、MCG に対して、バックイングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
出力は次のようになります。

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

YAML を使用してストレージリソースを追加することもできます。

1. 認証情報でシークレットを作成します。

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

- a. Base64 を使用して独自の IBM COS アクセスキー ID およびシークレットアクセスキーを指定し、エンコードし、その結果を **<IBM COS ACCESS KEY ID ENCODED IN BASE64>** および **<IBM COS SECRET ACCESS KEY ENCODED IN BASE64>** に使用する必要があります。
 - b. **<backingstore-secret-name>** を一意の名前に置き換えます。
2. 特定のバックイングストアについて以下の YAML を適用します。

```
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
```

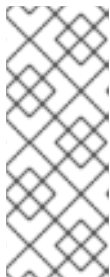
- <bucket-name>** を既存の IBM COS バケット名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
- <endpoint>** を、既存の IBM バケット名の場所に対応する地域のエンドポイントに置き換えます。この引数は、Multicloud Object Gateway に対して、バックングストア、およびその後のデータストレージおよび管理に使用するエンドポイントについて指示します。
- <backingstore-secret-name>** を直前の手順で作成したシークレットの名前に置き換えます。

4.2.3. Azure でサポートされるバックングストアの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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

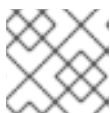


注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

- または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/packages にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

- MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- <backingstore_name>** を、バックングストアの名前に置き換えます。

- b. **<AZURE ACCOUNT KEY>** および **<AZURE ACCOUNT NAME>** は、この目的のために作成した AZURE アカウントキーおよびアカウント名に置き換えます。
- c. **<blob container name>** を既存の Azure blob コンテナ名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
出力は次のようになります。

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

YAML を使用してストレージリソースを追加することもできます。

1. 認証情報でシークレットを作成します。

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

- a. Base64 を使用して独自の Azure アカウント名およびアカウントキーを指定し、エンコードし、その結果を **<AZURE ACCOUNT NAME ENCODED IN BASE64>** および **<AZURE ACCOUNT KEY ENCODED IN BASE64>** に使用する必要があります。
 - b. **<backingstore-secret-name>** を一意の名前に置き換えます。
2. 特定のバックングストアについて以下の YAML を適用します。

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

- a. **<blob-container-name>** を既存の Azure blob コンテナ名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。

- b. **<backingstore-secret-name>** を直前の手順で作成したシークレットの名前に置き換えます。

4.2.4. GCP でサポートされるバックングストアの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

- または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/packages にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- a. **<backingstore_name>** を、バックングストアの名前に置き換えます。
- b. **<PATH TO GCP PRIVATE KEY JSON FILE>** を、この目的で作成された GCP プライベートキーへのパスに置き換えます。
- c. **<GCP bucket name>** を、既存の GCP オブジェクトストレージバケット名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。出力は次のようになります。

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

YAML を使用してストレージリソースを追加することもできます。

1. 認証情報でシークレットを作成します。

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

- a. Base64 を使用して独自の GCP サービスアカウントプライベートキー ID を指定し、エンコードし、その結果を **<GCP PRIVATE KEY ENCODED IN BASE64>** の場所で使用する必要があります。
 - b. **<backingstore-secret-name>** を一意の名前に置き換えます。
2. 特定のバックングストアについて以下の YAML を適用します。

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

- a. **<target bucket>** を、既存の Google ストレージバケットに置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
- b. **<backingstore-secret-name>** を直前の手順で作成したシークレットの名前に置き換えます。

4.2.5. ローカル永続ボリュームでサポートされるバックングストアの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

- または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/packages にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. MCG コマンドラインインターフェイスから、以下のコマンドを実行します。



注記

このコマンドは、**openshift-storage** namespace 内から実行する必要があります。

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

- a. **<backingstore_name>** を、バックアップストアの名前に置き換えます。
- b. **<NUMBER OF VOLUMES>** を、作成するボリューム数に置き換えます。ボリュームの数を増やすと、ストレージが拡大することに注意してください。
- c. **<VOLUME SIZE>** を、各ボリュームに必要なサイズ (GB 単位) に置き換えます。
- d. **<LOCAL STORAGE CLASS>** をローカルストレージクラスに置き換えます。これは、**ocs-storagecluster-ceph-rbd** を使用する際に推奨されます。出力は次のようになります。

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

YAML を使用してストレージリソースを追加することもできます。

1. 特定のバックアップストアについて以下の YAML を適用します。

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

```

app: noobaa
name: <backingstore_name>
namespace: openshift-storage
spec:
  pvPool:
    numVolumes: <NUMBER OF VOLUMES>
    resources:
      requests:
        storage: <VOLUME SIZE>
        storageClass: <LOCAL STORAGE CLASS>
    type: pv-pool

```

- a. **<backingstore_name>** を、バックングストアの名前に置き換えます。
- b. **<NUMBER OF VOLUMES>** を、作成するボリューム数に置き換えます。ボリュームの数を増やすと、ストレージが拡大することに注意してください。
- c. **<VOLUME SIZE>** を、各ボリュームに必要なサイズ (GB 単位) に置き換えます。文字 **G** はそのままにする必要があることに注意してください。
- d. **<LOCAL STORAGE CLASS>** をローカルストレージクラスに置き換えます。これは、**ocs-storagecluster-ceph-rbd** を使用する際に推奨されます。

4.3. S3 と互換性のある MULTICLOUD OBJECT GATEWAY バックングストアの作成

Multicloud Object Gateway (MCG) は、任意の S3 と互換性のあるオブジェクトストレージをバックングストアとして使用できます (例: Red Hat Ceph Storage の RADOS Object Gateway (RGW))。以下の手順では、Red Hat Ceph Storage の RGW 用の S3 と互換性のある MCG バックングストアを作成する方法を説明します。RGW がデプロイされると、OpenShift Data Foundation operator は MCG の S3 と互換性のあるバックングストアを自動的に作成することに注意してください。

手順

1. MCG コマンドラインインターフェイスから、以下のコマンドを実行します。



注記

このコマンドは、**openshift-storage** namespace 内から実行する必要があります。

```

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

```

- a. **<RGW ACCESS KEY>** および **<RGW SECRET KEY>** を取得するには、RGW ユーザーシークレット名を使用して以下のコマンドを実行します。

```

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

```

- b. Base64 からアクセスキー ID とアクセスキーをデコードし、それらのキーを保持します。

- c. **<RGW USER ACCESS KEY>** と **<RGW USER SECRET ACCESS KEY>** を、直前の手順でデコードした適切なデータに置き換えます。
- d. **<bucket-name>** を既存の RGW バケット名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
- e. **<RGW endpoint>** を取得するには、[RADOS Object Gateway S3 エンドポイントへのアクセス](#) を参照してください。
出力は次のようになります。

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

YAML を使用してバックングストアを作成することもできます。

1. **CephObjectStore** ユーザーを作成します。これにより、RGW 認証情報が含まれるシークレットも作成されます。

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

- a. **<RGW-Username>** と **<Display-name>** を、一意のユーザー名および表示名に置き換えます。
2. 以下の YAML を S3 と互換性のあるバックングストアについて適用します。

```
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. **<backingstore-secret-name>** を、直前の手順で **CephObjectStore** で作成したシークレットの名前に置き換えます。

- b. **<bucket-name>** を既存の RGW バケット名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
- c. **<RGW endpoint>** を取得するには、[RADOS Object Gateway S3 エンドポイントへのアクセス](#) を参照してください。

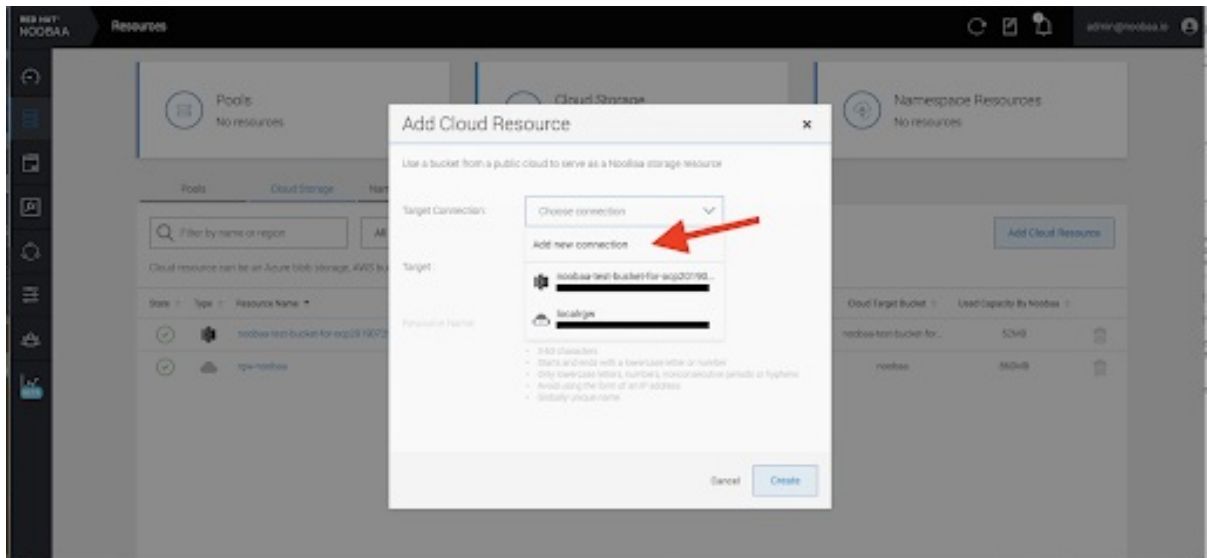
4.4. ユーザーインターフェイスを使用したハイブリッドおよびマルチクラウドのストレージリソースの追加

手順

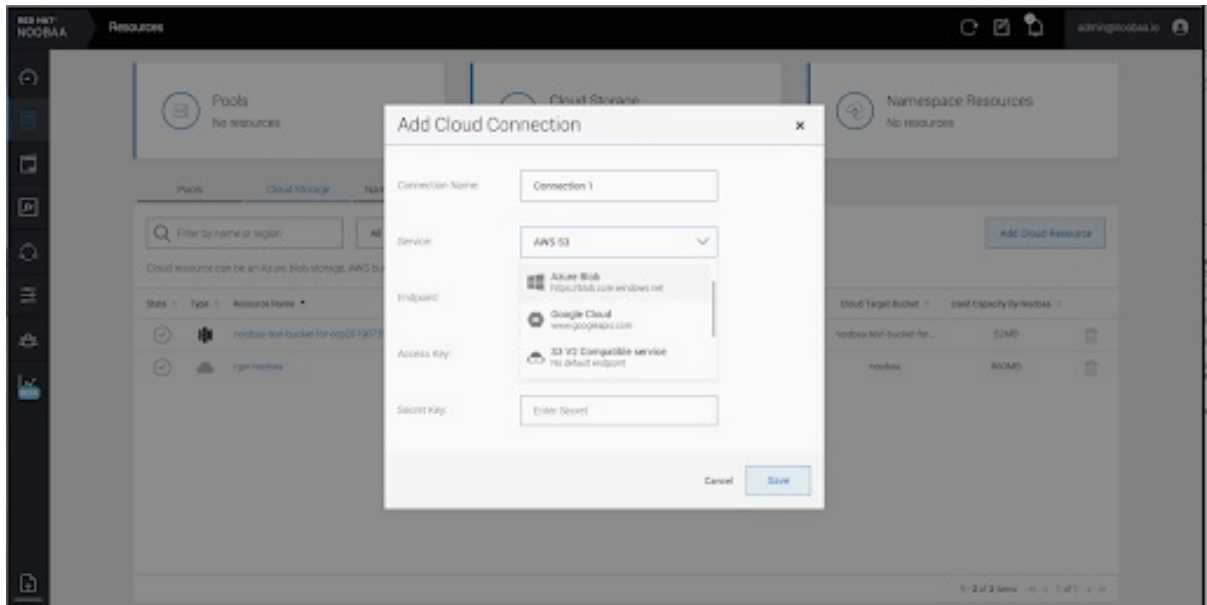
1. OpenShift Web コンソールで、**Storage** → **Data Foundation** をクリックします。
2. **Storage Systems** タブでストレージシステムを選択し、**Overview** → **Object** タブをクリックします。
3. **Multicloud Object Gateway** のリンクをクリックします。
1. 以下に強調表示されているように左側にある **Resources** タブを選択します。設定する一覧から、**Add Cloud Resource** を選択します。



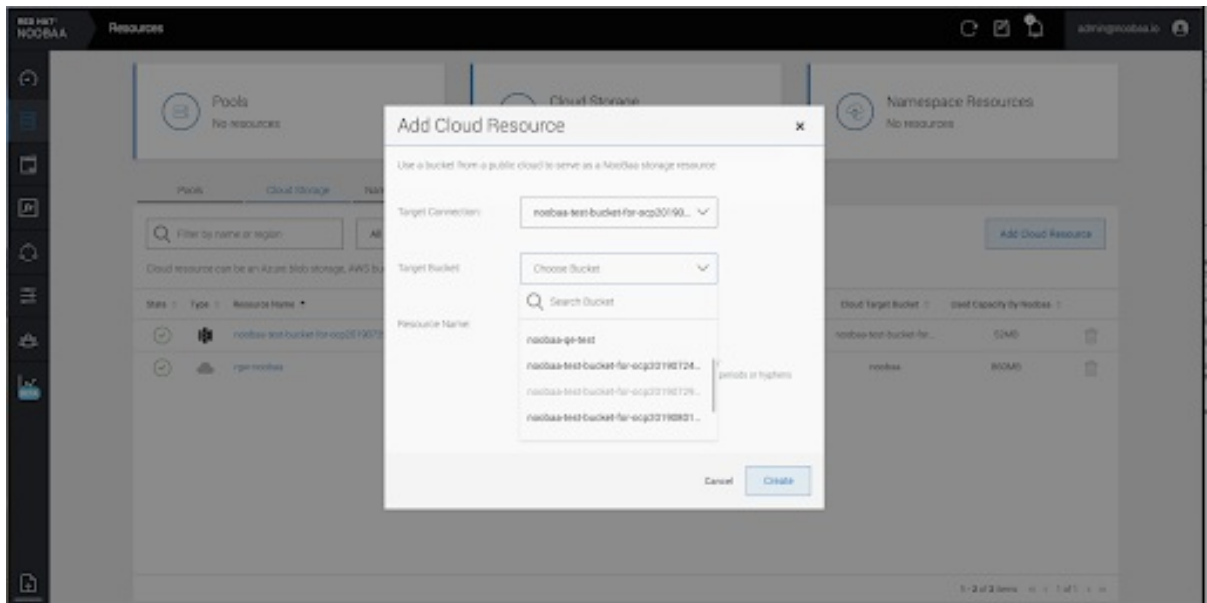
2. **Add new connection** を選択します。



3. 関連するネイティブクラウドプロバイダーまたは S3 互換オプションを選択し、詳細を入力します。



4. 新規に作成された接続を選択し、これを既存バケットにマップします。



5. これらの手順を繰り返して、必要な数のバッキングストアを作成します。



注記

NooBaa UI で作成されたリソースは、OpenShift UI または MCG CLI では使用できません。

4.5. 新規バケットクラスの作成

バケットクラスは、OBC (Object Bucket Class) の階層ポリシーおよびデータ配置を定義するバケットのクラスを表す CRD です。

以下の手順を使用して、OpenShift Data Foundation でバケットクラスを作成します。

手順

1. OpenShift Web コンソールで、**Storage** → **Data Foundation** をクリックします。

2. **Bucket Class** タブをクリックします。
 3. **Create Bucket Class** をクリックします。
 4. Create new Bucket Class ページで、以下を実行します。
 - a. バケットクラスタイプを選択し、バケットクラス名を入力します。
 - i. **BucketClass** タイプを選択します。以下のいずれかのオプションを選択します。
 - **Standard**: データは Multicloud Object Gateway (MCG) に使用され、重複排除、圧縮、および暗号化されます。
 - **Namespace**: データは、重複排除、圧縮、または暗号化を実行せずに NamespaceStores に保存されます。デフォルトでは、**Standard** が選択されます。
 - ii. **Bucket Class Name** 名を入力します。
 - iii. **Next** をクリックします。
 - b. **Placement Policy** で **Tier 1 - Policy Type** を選択し、**Next** をクリックします。要件に応じて、いずれかのオプションを選択できます。
 - **Spread** により、選択したリソース全体にデータを分散できます。
 - **Mirror** により、選択したリソース全体でデータを完全に複製できます。
 - **Add Tier** をクリックし、別のポリシー階層を追加します。
 - c. **Tier 1 - Policy Type** で **Spread** を選択した場合は、利用可能な一覧から1つ以上の **Backing Store** リソースを選択してから、**Next** をクリックします。また、[新しいバックイングストアを作成](#) することも可能です。
-
- 注記**
- 直前の手順で Policy Type に Mirror を選択する場合は、2つ以上のバックイングストアを選択する必要があります。

- d. Bucket Class 設定を確認し、確認します。
- e. **Create Bucket Class** をクリックします。

検証手順

1. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。
2. **Bucket Class** タブをクリックし、新しい Bucket Class を検索します。

4.6. バケットクラスの編集

以下の手順に従って、OpenShift Web コンソールの **edit** ボタンをクリックし、YAML ファイルを使用してバケットクラスコンポーネントを編集します。

前提条件

- OpenShift Web コンソールへの管理者アクセス。

手順

1. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。
2. **Bucket Class** タブをクリックします。
3. 編集する Bucket クラスの横にあるアクションメニュー (⋮) をクリックします。
4. **Edit Bucket Class** をクリックします。
5. **YAML** ファイルにリダイレクトされ、このファイルで必要な変更を加え、**Save** をクリックします。

4.7. バケットクラスのバックキングストアの編集

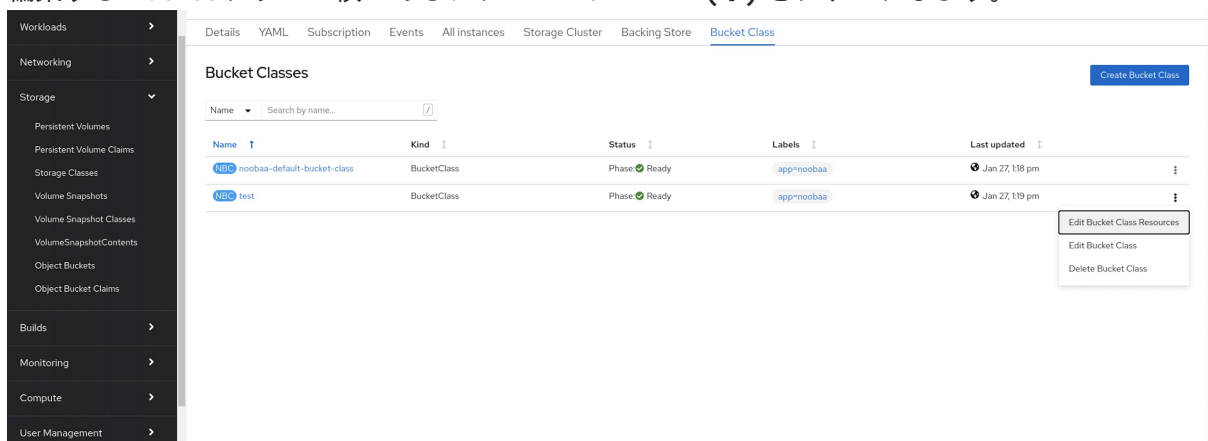
以下の手順を使用して、既存の Multicloud Object Gateway (MCG) バケットクラスを編集し、バケットクラスで使用される基礎となるバックキングストアを変更します。

前提条件

- OpenShift Web コンソールへの管理者アクセス。
- バケットクラス。
- バックキングストア。

手順

1. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。
2. **Bucket Class** タブをクリックします。
3. 編集する Bucket クラスの横にあるアクションメニュー (⋮) をクリックします。



4. **Edit Bucket Class Resources** をクリックします。
5. **Edit Bucket Class Resources** ページで、バックキングストアをバケットクラスに追加するか、バケットクラスからバックキングストアを削除してバケットクラスリソースを編集します。1つまたは2つの層を使用して作成されたバケットクラスリソースや、異なる配置ポリシーが指定されたバケットクラスリソースを編集することもできます。
 - バックキングストアをバケットクラスに追加するには、バックキングストアの名前を選択しま

す。

- バケットクラスからバックングストアを削除するには、バックングストアの名前を消去します。

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 *

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

2 Backing Stores selected

Tier 2 - Backing Stores (Mirror)

Select at least 2 Resources resources *

| <input checked="" type="checkbox"/> | Name | Target Bucket | Type | Region |
|-------------------------------------|----------------|---------------|---------------|------------|
| <input checked="" type="checkbox"/> | archive-bucket | buck-1 | S3 Compitable | Us-east-la |
| <input checked="" type="checkbox"/> | data-bucket | bucket-main | Azure Blob | Us-east-lb |
| <input checked="" type="checkbox"/> | buck-2 | buck-1 | S3 Compitable | Us-east-la |

2 Backing Stores selected

Save Cancel

6. Save をクリックします。

第5章 NAMESPACE バケットの管理

namespace バケットを使用すると、異なるプロバイダーのデータリポジトリを接続できるため、単一の統合ビューを使用してすべてのデータと対話できます。各プロバイダーに関連付けられたオブジェクトバケットを namespace バケットに追加し、namespace バケット経由でデータにアクセスし、一度にすべてのオブジェクトバケットを表示します。これにより、他の複数のストレージプロバイダーから読み込む間に、希望するストレージプロバイダーへの書き込みを行うことができ、新規ストレージプロバイダーへの移行コストが大幅に削減されます。



注記

namespace バケットは、このバケットの書き込みターゲットが利用可能で機能している場合にのみ使用できます。

5.1. NAMESPACE バケットのオブジェクトの AMAZON S3 API エンドポイント

Amazon Simple Storage Service (S3) API を使用して namespace バケットのオブジェクトと対話できます。

Red Hat OpenShift Data Foundation 4.6 以降では、以下の namespace バケット操作をサポートします。

- [ListObjectVersions](#)
- [ListObjects](#)
- [PutObject](#)
- [CopyObject](#)
- [ListParts](#)
- [CreateMultipartUpload](#)
- [CompleteMultipartUpload](#)
- [UploadPart](#)
- [UploadPartCopy](#)
- [AbortMultipartUpload](#)
- [GetObjectAcl](#)
- [GetObject](#)
- [HeadObject](#)
- [DeleteObject](#)
- [DeleteObjects](#)

これらの操作および使用方法に関する最新情報は、Amazon S3 API リファレンスのドキュメントを参照してください。

関連情報

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

5.2. MULTICLOUD OBJECT GATEWAY CLI および YAML を使用した NAMESPACE バケットの追加

namespace バケットの詳細は、[namespace バケットの管理](#) を参照してください。

デプロイメントのタイプに応じて、また YAML または Multicloud Object Gateway (MCG) CLI を使用するかどうかに応じて、以下の手順のいずれかを選択して namespace バケットを追加します。

- [YAML を使用した AWS S3 namespace バケットの追加](#)
- [YAML を使用した IBM COS namespace バケットの追加](#)
- [Multicloud Object Gateway CLI を使用した AWS S3 namespace バケットの追加](#)
- [Multicloud Object Gateway CLI を使用した IBM COS namespace バケットの追加](#)

5.2.1. YAML を使用した AWS S3 namespace バケットの追加

前提条件

- 実行中の OpenShift Data Foundation Platform。
- Multicloud Object Gateway (MCG) へのアクセスについては、第 2 章の [Accessing the Multicloud Object Gateway with your applications](#) を参照してください。

手順

1. 認証情報でシークレットを作成します。

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

- a. Base64 を使用して独自の AWS アクセスキー ID およびシークレットアクセスキーを指定し、エンコードし、その結果を **<AWS ACCESS KEY ID ENCODED IN BASE64>** および **<AWS SECRET ACCESS KEY ENCODED IN BASE64>** に使用する必要があります。
 - b. **<namespacestore-secret-name>** を一意の名前に置き換えます。
2. OpenShift カスタムリソース定義 (CRD) を使用して NamespaceStore リソースを作成します。NamespaceStore は、MCG namespace バケットでデータの読み取りおよび書き込みターゲットとして使用される基礎となるストレージを表します。NamespaceStore リソースを作成するには、以下の YAML を適用します。

```

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

```

- a. **<resource-name>** をリソースに指定する名前に置き換えます。
 - b. **<namespacestore-secret-name>** を手順1で作成したシークレットに置き換えます。
 - c. **<namespace-secret>** を、シークレットが含まれる namespace に置き換えます。
 - d. **<target-bucket>** を NamespaceStore 用に作成したターゲットバケットに置き換えます。
3. namespace バケットの namespace ポリシーを定義する namespace バケットクラスを作成します。namespace ポリシーには、**single** または **multi** のタイプが必要です。
- タイプ **single** の namespace ポリシーには、以下の設定が必要です。

```

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>** を一意のバケットクラス名に置き換えます。
 - **<resource>** を namespace バケットの読み取りおよび書き込みターゲットを定義する単一の namespace-store の名前に置き換えます。
- タイプが **multi** の namespace ポリシーには、以下の設定が必要です。

```

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>** を一意のバケットクラス名に置き換えます。
 - **<write-resource>** を、namespace バケットの書き込みターゲットを定義する単一の namespace-store の名前に置き換えます。
 - **<read-resources>** を、namespace バケットの読み取りターゲットを定義する namespace-stores の名前の一覧に置き換えます。
4. 以下のコマンドを実行して、手順 2 に定義されたバケットクラスを使用する Object Bucket Class (OBC) リソースを使用してバケットを作成します。

```

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>

```

- a. **<resource-name>** をリソースに指定する名前に置き換えます。
- b. **<my-bucket>** をバケットに指定する名前に置き換えます。
- c. **<my-bucket-class>** を直前の手順で作成したバケットクラスに置き換えます。

OBC が Operator によってプロビジョニングされると、バケットが MCG で作成され、Operator は OBC の同じ namespace 上に同じ名前でシークレットおよび ConfigMap を作成します。

5.2.2. YAML を使用した IBM COS namespace バケットの追加

前提条件

- 実行中の OpenShift Data Foundation Platform。
- Multicloud Object Gateway (MCG) へのアクセスについては、第 2 章の [Accessing the Multicloud Object Gateway with your applications](#) を参照してください。

手順

1. 認証情報でシークレットを作成します。


```

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

```

- a. Base64 を使用して独自の IBM COS アクセスキー ID およびシークレットアクセスキーを指定し、エンコードし、その結果を **<IBM COS ACCESS KEY ID ENCODED IN BASE64>** および **<IBM COS SECRET ACCESS KEY ENCODED IN BASE64>** に使用する必要があります。
 - b. **<namespacestore-secret-name>** を一意の名前に置き換えます。
2. OpenShift カスタムリソース定義 (CRD) を使用して NamespaceStore リソースを作成します。NamespaceStore は、MCG namespace バケットでデータの読み取りおよび書き込みターゲットとして使用される基礎となるストレージを表します。NamespaceStore リソースを作成するには、以下の YAML を適用します。

```

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

```

- a. **<IBM COS ENDPOINT>** を適切な IBM COS エンドポイントに置き換えます。
 - b. **<namespacestore-secret-name>** を手順 1 で作成したシークレットに置き換えます。
 - c. **<namespace-secret>** を、シークレットが含まれる namespace に置き換えます。
 - d. **<target-bucket>** を NamespaceStore 用に作成したターゲットバケットに置き換えます。
3. namespace バケットの namespace ポリシーを定義する namespace バケットクラスを作成します。namespace ポリシーには、**single** または **multi** のタイプが必要です。
- タイプ **single** の namespace ポリシーには、以下の設定が必要です。

```

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>** を一意のバケットクラス名に置き換えます。
- **<resource>** を namespace バケットの読み取りおよび書き込みターゲットを定義する単一の namespace-store の名前に置き換えます。
- タイプが **multi** の namespace ポリシーには、以下の設定が必要です。

```

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>** を一意のバケットクラス名に置き換えます。
 - **<write-resource>** を、namespace バケットの書き込みターゲットを定義する単一の namespace-store の名前に置き換えます。
 - **<read-resources>** を、namespace バケットの読み取りターゲットを定義する namespace-stores の名前の一覧に置き換えます。
4. 以下のコマンドを実行して、手順 2 に定義されたバケットクラスを使用する Object Bucket Class (OBC) リソースを使用してバケットを作成します。

```

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>

```

- a. **<resource-name>** をリソースに指定する名前に置き換えます。
- b. **<my-bucket>** をバケットに指定する名前に置き換えます。
- c. **<my-bucket-class>** を直前の手順で作成したバケットクラスに置き換えます。

OBC が Operator によってプロビジョニングされると、バケットが MCG で作成され、Operator は OBC の同じ namespace 上に同じ名前でシークレットおよび ConfigMap を作成します。

5.2.3. Multicloud Object Gateway CLI を使用した AWS S3 namespace バケットの追加

前提条件

- 実行中の OpenShift Data Foundation Platform。
- Multicloud Object Gateway (MCG) へのアクセスについては、第 2 章の [Accessing the Multicloud Object Gateway with your applications](#) を参照してください。
- MCG コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. NamespaceStore リソースを作成します。NamespaceStore は、MCG namespace バケットでデータの読み取りおよび書き込みターゲットとして使用される基礎となるストレージを表します。MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- a. **<namespacestore>** を NamespaceStore の名前に置き換えます。
- b. **<AWS ACCESS KEY>** および **<AWS SECRET ACCESS KEY>** を、作成した AWS アクセスキー ID およびシークレットアクセスキーに置き換えます。

- c. **<bucket-name>** を既存の AWS バケット名に置き換えます。この引数は、MCG に対して、バックストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
2. namespace バケットの namespace ポリシーを定義する namespace バケットクラスを作成します。namespace ポリシーには、**single** または **multi** のタイプが必要です。
 - 以下のコマンドを実行して、タイプ **single** の namespace ポリシーで namespace バケットクラスを作成します。

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

- **<resource-name>** をリソースに指定する名前に置き換えます。
 - **<my-bucket-class>** を一意のバケットクラス名に置き換えます。
 - **<resource>** を namespace バケットの読み取りおよび書き込みターゲットを定義する単一の namespace-store に置き換えます。
- 以下のコマンドを実行して、タイプ **multi** の namespace ポリシーで namespace バケットクラスを作成します。

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

- **<resource-name>** をリソースに指定する名前に置き換えます。
 - **<my-bucket-class>** を一意のバケットクラス名に置き換えます。
 - **<write-resource>** を、namespace バケットの書き込みターゲットを定義する単一の namespace-store に置き換えます。
 - **<read-resources>** を、namespace バケットの読み取りターゲットを定義する、コマンドで区切られた namespace-stores の一覧に置き換えます。
3. 以下のコマンドを実行して、手順 2 に定義されたバケットクラスを使用する Object Bucket Class (OBC) リソースを使用してバケットを作成します。

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

- a. **<bucket-name>** を、選択するバケット名に置き換えます。
- b. **<custom-bucket-class>** を、手順 2 で作成したバケットクラスの名前に置き換えます。

OBC が Operator によってプロビジョニングされると、バケットが MCG で作成され、Operator は OBC の同じ namespace 上に同じ名前でもシークレットおよび ConfigMap を作成します。

5.2.4. Multicloud Object Gateway CLI を使用した IBM COS namespace バケットの追加

前提条件

- 実行中の OpenShift Data Foundation Platform。

- Multicloud Object Gateway (MCG) へのアクセスについては、第2章の [Accessing the Multicloud Object Gateway with your applications](#) を参照してください。
- MCG コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。

- IBM Power の場合は、次のコマンドを使用します。

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

- IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/package にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. NamespaceStore リソースを作成します。NamespaceStore は、MCG namespace バケットでデータの読み取りおよび書き込みターゲットとして使用される基礎となるストレージを表します。MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- a. **<namespacestore>** を NamespaceStore の名前に置き換えます。
 - b. **<IBM ACCESS KEY>**, **<IBM SECRET ACCESS KEY>**, **<IBM COS ENDPOINT>** を IBM アクセスキー ID、シークレットアクセスキー、および既存の IBM バケットの場所に対応する地域のエンドポイントに置き換えます。
 - c. **<bucket-name>** を既存の IBM バケット名に置き換えます。この引数は、MCG に対して、バックストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
2. namespace バケットの namespace ポリシーを定義する namespace バケットクラスを作成します。namespace ポリシーには、**single** または **multi** のタイプが必要です。
 - 以下のコマンドを実行して、タイプ **single** の namespace ポリシーで namespace バケットクラスを作成します。

■

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

- **<resource-name>** をリソースに指定する名前に置き換えます。
- **<my-bucket-class>** を一意のバケットクラス名に置き換えます。
- **<resource>** を namespace バケットの読み取りおよび書き込みターゲットを定義する単一の namespace-store に置き換えます。
- 以下のコマンドを実行して、タイプ **multi** の namespace ポリシーで namespace バケットクラスを作成します。

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

- **<resource-name>** をリソースに指定する名前に置き換えます。
 - **<my-bucket-class>** を一意のバケットクラス名に置き換えます。
 - **<write-resource>** を、namespace バケットの書き込みターゲットを定義する単一の namespace-store に置き換えます。
 - **<read-resources>** を、namespace バケットの読み取りターゲットを定義する、コマンドで区切られた namespace-stores の一覧に置き換えます。
3. 以下のコマンドを実行して、手順 2 に定義されたバケットクラスを使用する Object Bucket Class (OBC) リソースを使用してバケットを作成します。

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

- a. **<bucket-name>** を、選択するバケット名に置き換えます。
- b. **<custom-bucket-class>** を、手順 2 で作成したバケットクラスの名前に置き換えます。

OBC が Operator によってプロビジョニングされると、バケットが MCG で作成され、Operator は OBC の同じ namespace 上に同じ名前ですークレットおよび ConfigMap を作成します。

5.3. OPENSIFT CONTAINER PLATFORM ユーザーインターフェイスを使用した NAMESPACE バケットの追加

OpenShift Data Foundation 4.8 以降のリリースでは、namespace バケットは OpenShift Container Platform ユーザーインターフェイスを使用して追加できます。namespace バケットの詳細は、[namespace バケットの管理](#) を参照してください。

前提条件

- OpenShift Data Foundation Operator を使用した OpenShift Container Platform のインストール
- Multicloud Object Gateway (MCG) へのアクセス。

手順

1. OpenShift Web コンソールにログインします。
2. Storage → Data Foundation をクリックします。
3. **Namespace Store** タブをクリックして、namespace バケットで使用される **namespacestore** リソースを作成します。
 - a. **Create namespace store** をクリックします。
 - b. namespacestore 名を入力します。
 - c. プロバイダーを選択します。
 - d. リージョンを選択します。
 - e. 既存のシークレットを選択するか、**Switch to credentials** をクリックして、シークレットキーおよびシークレットアクセスキーを入力してシークレットを作成します。
 - f. ターゲットバケットを選択します。
 - g. **Create** をクリックします。
 - h. namespacestore が **Ready** 状態にあることを確認します。
 - i. 必要なリソースが得られるまで、これらの手順を繰り返します。
4. **Bucket Class** タブ → **Create a new Bucket Class** をクリックします。
 - a. **Namespace** ラジオボタンを選択します。
 - b. Bucket Class 名を入力します。
 - c. 説明 (オプション) を追加します。
 - d. **Next** をクリックします。
5. namespace バケットの namespace ポリシータイプを選択し、**Next** をクリックします。
6. ターゲットリソースを選択します。
 - Namespace ポリシータイプが **Single** の場合、読み取りリソースを選択する必要があります。
 - namespace ポリシータイプが **Multi** の場合、読み取りリソースおよび書き込みリソースを選択する必要があります。
 - namespace ポリシータイプが **Cache** の場合は、namespace バケットの読み取りおよび書き込みターゲットを定義する Hub namespace ストアを選択する必要があります。
7. **Next** をクリックします。
8. 新しいバケットクラスを確認してから **Create Bucketclass** をクリックします。
9. **BucketClass** ページで、新たに作成されたリソースが **Created** フェーズにあることを確認します。
10. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。

11. **Status** カードで **Storage System** をクリックし、表示されるポップアップからストレージシステムリンクをクリックします。
12. **Object** タブで、**Multicloud Object Gateway** → **Buckets** → **Namespace Buckets** タブをクリックします。
13. **Create Namespace Bucket** をクリックします。
 - a. **Choose Name** タブで、namespace バケットの **Name** を指定し、**Next** をクリックします。
 - b. **Set Placement** タブで、以下を実行します。
 - i. **Read Policy** で、namespace バケットがデータの読み取りに使用する、ステップ 5 で作成した各 namespace リソースのチェックボックスを選択します。
 - ii. 使用している namespace ポリシタイプが **Multi** の場合、**Write Policy** の場合は、namespace バケットがデータを書き込む namespace リソースを指定します。
 - iii. **Next** をクリックします。
 - c. **Create** をクリックします。

検証

- namespace バケットが **State** 列の緑色のチェックマークと、予想される読み取りリソースの数、および予想される書き込みリソース名と共に一覧表示されていることを確認します。

5.4. S3 プロトコルを使用してレガシーアプリケーションデータをクラウドネイティブアプリケーションと共有する

多くのレガシーアプリケーションは、ファイルシステムを使用してデータセットを共有します。S3 操作を使用して、ファイルシステム内のレガシーデータにアクセスして共有できます。データを共有するには、次のことを行う必要があります。

- 既存のファイルシステムデータセット、つまり Ceph File System (CephFS) などの RWX ボリュームをエクスポートするか、S3 プロトコルを使用して新しいファイルシステムデータセットを作成します。
- ファイルシステムと S3 プロトコルの両方からファイルシステムデータセットにアクセスします。
- S3 アカウントを設定し、それらを既存または新規のファイルシステムの一意的識別子 (UID) とグループ識別子 (GID) にマップします。

5.4.1. ファイルシステムを使用するための NamespaceStore の作成

前提条件

- OpenShift Data Foundation Operator を使用した OpenShift Container Platform のインストール
- Multicloud Object Gateway (MCG) へのアクセス。

手順

1. OpenShift Web コンソールにログインします。
2. **Storage** → **Data Foundation** をクリックします。
3. **Namespace Store** タブをクリックして、namespace バケットで使用される NamespaceStore リソースを作成します。
4. **Create namespaces** をクリックします。
5. NamespaceStore の名前を入力します。
6. プロバイダーとして**ファイルシステム**を選択します。
7. 永続ボリュームクレームを選択します。
8. フォルダー名を入力します。
フォルダー名が存在する場合は、そのフォルダーを使用して NamespaceStore を作成するか、その名前のフォルダーを作成します。
9. **Create** をクリックします。
10. namespacestore が Ready 状態にあることを確認します。

5.4.2. NamespaceStore ファイルシステム設定を使用したアカウントの作成

NamespaceStore ファイルシステム設定を使用して新しいアカウントを作成するか、YAML を編集して既存の通常のアカウントを NamespaceStore ファイルシステムアカウントに変換できます。



注記

NamespaceStore ファイルシステム設定をアカウントから削除することはできません。

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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

手順

- MCG コマンドラインインターフェイスを使用して、NamespaceStore ファイルシステム設定で新しいアカウントを作成します。

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

以下に例を示します。

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

| | |
|---------------------|---|
| allow_bucket_create | アカウントが新しいバケットの作成を許可されているかどうかを示します。サポートされている値は true または false です。デフォルト値は true です。 |
|---------------------|---|

| | |
|---------------------|---|
| allowed_buckets | ユーザーがアクセス権と管理権を持つことを許可されているバケット名のコンマ区切りのリスト。 |
| default_resource | S3 CreateBucket 操作を使用するときに新しいバケットが作成される NamespaceStore リソース。NamespaceStore は、RWX (ReadWriteMany) 永続ボリューム要求 (PVC) によってサポートされている必要があります。 |
| full_permission | アカウントに完全な許可を許可するかどうかを示します。サポートされている値は true または false です。デフォルト値は false です。 |
| new_buckets_path | 新しいバケットに対応するディレクトリーが作成されるファイルシステムパス。パスは、NamespaceStore ファイルシステム PVC のファイルシステム内にあり、新しく作成されたオブジェクトバケットクラスのファイルシステムマッピングとして機能する新しいディレクトリーが作成されます。 |
| nsfs_account_config | アカウントが NamespaceStore ファイルシステムに使用されているかどうかを示す必須フィールド。 |
| nsfs_only | アカウントが NamespaceStore ファイルシステムにのみ使用されるかどうかを示します。サポートされている値は true または false です。デフォルト値は false です。true に設定すると、他のタイプのバケットへのアクセスが制限されます。 |
| uid | MCG アカウントがマップされるファイルシステムのユーザー ID であり、ファイルシステム上のデータにアクセスして管理するために使用されます。 |
| gid | MCG アカウントがマップされるファイルシステムのグループ ID であり、ファイルシステム上のデータにアクセスして管理するために使用されます。 |

MCG システムは、アカウント設定とその S3 クレデンシャルを含む応答を送信します。

```
# 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>
```

次のコマンドを使用して、すべてのカスタムリソース定義 (CRD) ベースのアカウントを一覧表示できます。

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

特定のアカウントに関心がある場合は、そのカスタムリソース定義 (CRD) をアカウント名で直接読み取ることができます。

```
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
```

5.4.3. openshift-storage namespace からレガシーアプリケーションデータにアクセスする

Multicloud Object Gateway (MCG) NamespaceStore ファイルシステム (NSFS) 機能を使用する場合、データが **openshift-storage** namespace に存在する 永続ボリューム要求 (PVC) が必要です。ほとんどすべての場合、アクセスする必要のあるデータは、**openshift-storage** namespace ではなく、レガシーアプリケーションが使用する namespace にあります。

別の namespace に保存されているデータにアクセスするには、レガシーアプリケーションが使用するのと同じ CephFS ボリュームを指す PVC を **openshift-storage** namespace に作成する必要があります。

手順

1. **scc** を使用してアプリケーションの namespace を表示します。

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

<application_namespace>

アプリケーションの namespace の名前を指定します。

例5.1 例

```
$ oc get ns testnamespace -o yaml | grep scc
```

例5.2 出力例

```
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. アプリケーションの namespace に移動します。

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

例5.3 例

```
$ oc project testnamespace
```

3. MCG NSFS 機能を使用して、noobaa S3 エンドポイントから消費する Pod に ReadWriteMany (RWX) PVC がマウントされていることを確認します。

```
$ oc get pvc
```

例5.4 出力例

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

```
$ oc get pod
```

例5.5 出力例

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

4. Pod 内の永続ボリューム (PV) のマウントポイントを確認します。

- a. Pod から PV のボリューム名を取得します。

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

<pod_name>

pod の名前を指定します。

例5.6 例

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

例5.7 出力例

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

この例では、PVC のボリュームの名前は **cephfs-write-workload-generator-no-cache-pv-claim** です。

- b. Pod 内のすべてのマウントを一覧表示し、前の手順で特定したボリュームのマウントポイントを確認します。

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

例5.8 例

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

例5.9 出力例

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

5. Pod 内の RWX PV のマウントポイントを確認します。

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

<mount_path>

前の手順で特定したマウントポイントへのパスを指定します。

例5.10 例

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

例5.11 出力例

```
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-8824-8a03ad94d67c
10485760 0 10485760 0% /mnt/pv
```

6. UID および SELinux ラベルが、レガシー namespace が使用するものと同じであることを確認してください。

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

例5.12 例

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

例5.13 出力例

```
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. **openshift-storage** namespace からアクセス可能にするレガシーアプリケーション RWX PV の情報を取得します。

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

<pv_name>

PV の名前を指定します。

例5.14 例

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

例5.15 出力例

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

8. 1つ以上の noobaa-endpoint Pod が PVC にアクセスできるように、レガシーアプリケーションの PVC が **openshift-storage** namespace からアクセス可能であることを確認します。
 - a. **volumeAttributes** から **subvolumePath** と **volumeHandle** の値を検索します。これらの値は、レガシーアプリケーション PV の YAML 記述から取得できます。

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

例5.16 例

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

例5.17 出力例

```

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. 前のステップで特定した **subvolumePath** と **volumeHandle** の値を使用して、レガシーアプリケーション PV と同じ CephFS ボリュームを指す **openshift-storage** namespace に新しい PV および PVC オブジェクトを作成します。

例5.18 サンプル YAML ファイル

```

$ cat << EOF >> pv-openshift-storage.yaml
apiVersion: v1

```

```

kind: PersistentVolume
metadata:
  name: cephfs-pv-legacy-openshift-storage
spec:
  storageClassName: ""
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi ①
  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-bdcb-4b8e-8824-8a03ad94d67c
      volumeHandle: 0001-0011-openshift-storage-0000000000000001-cc416d9e-dbf3-
11ec-b286-0a580a810213-clone ②
    persistentVolumeReclaimPolicy: Retain
    volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cephfs-pvc-legacy
  namespace: openshift-storage
spec:
  storageClassName: ""
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi ③
  volumeMode: Filesystem
  # volumeName should be same as PV name
  volumeName: cephfs-pv-legacy-openshift-storage
EOF

```

- ① **openshift-storage** namespace で作成する PV のストレージ容量は、元の PV と同じである必要があります。
- ② **openshift-storage** で作成するターゲット PV のボリュームハンドルには、元のアプリケーション PV とは異なるハンドルが必要です。たとえば、ボリュームハンドルの末尾に **-clone** を追加します。
- ③ **openshift-storage** namespace で作成する PVC のストレージ容量は、元の PVC と同じである必要があります。

- c. 前のステップで指定した YAML ファイルを使用して、**openshift-storage** namespace に PV と PVC を作成します。

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

<YAML_file>

YAML ファイルの名前を指定します。

例5.19 例

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

例5.20 出力例

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

- d. PVC が **openshift-storage** namespace で使用可能であることを確認します。

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

例5.21 出力例

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

- e. **openshift-storage** プロジェクトに移動します。

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

例5.22 出力例

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

- f. NSFS namespace ストアを作成します。

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

<nsfs_namespacestore>

NSFS namespace ストアの名前を指定します。

<cephfs_pvc_name>

openshift-storage namespace で CephFSPVC の名前を指定します。

例5.23 例

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

- g. noobaa-endpointPod が再起動し、NSFS namespace ストア (**/nsfs/legacy-namespace** mountpoint など) に PVC が正常にマウントされていることを確認します。

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

<noobaa_endpoint_pod_name>

noobaa エンドポイント Pod の名前を指定します。

例5.24 例

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

例5.25 出力例

```
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. MCG ユーザーアカウントを作成します。

```
$ 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>

MCG ユーザーアカウントの名前を指定します。

<gid_number>

GID 番号を指定します。

<uid_number>

UID 番号を指定します。

例5.26 例



重要

レガシーアプリケーションと同じ **UID** と **GID** を使用します。前の出力から見つけることができます。

```
$ 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. MCG バケットを作成します。

i. レガシーアプリケーション Pod の CephFS PV および PVC の NSFS 共有内に S3 専用のフォルダーを作成します。

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

例5.27 例

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

ii. **nsfs/** パスを使用して MCG バケットを作成します。

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

例5.28 例

```
$ 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. レガシーアプリケーションおよび **openshift-storage** namespace の PVC にあるフォルダーの SELinux ラベルを確認します。

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

例5.29 例

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

例5.30 出力例

```
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 -latrZ <mount_path>
```

例5.31 例

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

例5.32 出力例

```
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 ..
```

これらの例では、SELinux ラベルが同じではないため、アクセス許可が拒否されたり、アクセスの問題が発生したりすることがわかります。

9. レガシーアプリケーションと **openshift-storage** Pod がファイルで同じ SELinux ラベルを使用していることを確認します。

これは、次の 2 つの方法のいずれかで実行できます。

- 「レガシーアプリケーションプロジェクトのデフォルトの SELinux ラベルを、openshift-storage プロジェクトのラベルと一致するように変更します」
- 「レガシーアプリケーション PVC をマウントする Pod を持つデプロイメント設定に対してのみ SELinux ラベルを変更する」

10. NSFS namespace ストアを削除します。

- a. MCG バケットを削除します。

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

例5.33 例

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

b. MCG ユーザーアカウントを削除します。

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

例5.34 例

```
$ noobaa account delete leguser
```

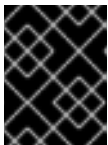
c. NSFS namespace ストアを削除します。

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

例5.35 例

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

11. PV と PVC を削除します。



重要

PV と PVC を削除する前に、PV に保持ポリシーが設定されていることを確認してください。

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

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

<cephfs_pv_name>

レガシーアプリケーションの CephFS PV 名を指定します。

<cephfs_pvc_name>

レガシーアプリケーションの CephFS PVC 名を指定します。

例5.36 例

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

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

5.4.3.1. レガシーアプリケーションプロジェクトのデフォルトの SELinux ラベルを、openshift-storage プロジェクトのラベルと一致するように変更します

1. 現在の **openshift-storage** namespace を **sa.scc.mcs** で表示します。

```
$ oc get ns openshift-storage -o yaml | grep sa.scc.mcs
```

例5.37 出力例

```
openshift.io/sa.scc.mcs: s0:c26,c0
```

- レガシーアプリケーションの namespace を編集し、**openshift-storage** namespace の **sa.scc.mcs** の値で **sa.scc.mcs** を変更します。

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

例5.38 例

```
$ oc edit ns testnamespace
```

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

例5.39 例

```
$ oc get ns testnamespace -o yaml | grep sa.scc.mcs
```

例5.40 出力例

```
openshift.io/sa.scc.mcs: s0:c26,c0
```

- レガシーアプリケーション Pod を再起動します。すべてのファイルの再ラベル付けが行われ、SELinux ラベルが **openshift-storage** デプロイメントと一致するようになりました。

5.4.3.2. レガシーアプリケーション PVC をマウントする Pod を持つデプロイメント設定に対してのみ SELinux ラベルを変更する

- MustRunAs** および **seLinuxOptions** オプションを使用して、**openshift-storage** プロジェクトが使用するマルチカテゴリーセキュリティー (MCS) を使用して新しい **scc** を作成します。

例5.41 サンプル YAML ファイル

```
$ 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. デプロイメント用のサービスアカウントを作成し、新しく作成した **scc** に追加します。

a. サービスアカウントを作成します。

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

<service_account_name>

サービスアカウントの名前を指定します。

例5.42 例

```
$ oc create serviceaccount testnamespacesa
```

b. 新しく作成した **scc** にサービスアカウントを追加します。

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

例5.43 例

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

- 新しく作成されたサービスアカウントを使用するように、レガシーアプリケーションのデプロイメントにパッチを適用します。これで、デプロイメントで SELinux ラベルを指定できるようになりました。

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

例5.44 例

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

- デプロイメントを編集して、デプロイメント設定の SELinux ラベルで使用するセキュリティーコンテキストを指定します。

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

以下の行を追加します。

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

<security_context_value>

この値は、レガシーアプリケーション Pod の CephFS PV および PVC で、NSFS 共有内に S3 専用のフォルダーを作成するコマンドを実行するときに見つかります。

例5.45 例

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

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

- デプロイメント設定の SELinux ラベルで使用するセキュリティーコンテキストが正しく指定されていることを確認してください。

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


例5.46 例

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

例5.47 出力例

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

レガシーアプリケーションが再起動され、**openshift-storage** namespace と同じ SELinux ラベルの使用が開始されます。

第6章 マルチクラウドオブジェクトゲートウェイのセキュリティーを強化するために、デフォルトのアカウント資格情報を変更する

コマンドラインインターフェイスを使用して Multicloud Object Gateway (MCG) アカウントの資格情報を変更およびローテーションし、アプリケーションの問題を防ぎ、アカウントのセキュリティーを強化します。

デフォルトの MCG アカウントの認証情報を変更する方法の詳細は、Red Hat ナレッジベースのソリューション [How to change the default account credentials to ensure better security in the Multicloud Object Gateway?](#) を参照してください。

第7章 ハイブリッドおよびマルチクラウドバケットのデータのミラーリング

Multicloud Object Gateway (MCG) は、クラウドプロバイダーおよびクラスター全体にまたがるデータの処理を単純化します。

前提条件

- まず、MCG で使用できるバックングストレージを追加する必要があります。[4章ハイブリッドまたはマルチクラウド用のストレージリソースの追加](#)を参照してください。

次に、データ管理ポリシー (ミラーリング) を反映するバケットクラスを作成します。

手順

ミラーリングデータは、以下の3つの方法で設定できます。

- 「MCG コマンドラインインターフェイスを使用したデータのミラーリング用のバケットクラスの作成」
- 「YAML を使用したデータのミラーリング用のバケットクラスの作成」
- 「ユーザーインターフェイスを使用したデータミラーリングを行うためのバケットの設定」

7.1. MCG コマンドラインインターフェイスを使用したデータのミラーリング用のバケットクラスの作成

1. Multicloud Object Gateway (MCG) コマンドラインインターフェイスから以下のコマンドを実行し、ミラーリングポリシーでバケットクラスを作成します。

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

2. 新たに作成されたバケットクラスを新規のバケット要求に設定し、2つのロケーション間でミラーリングされる新規バケットを生成します。

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

7.2. YAML を使用したデータのミラーリング用のバケットクラスの作成

1. 以下のYAMLを適用します。このYAMLは、ローカルCephストレージとAWS間でデータをミラーリングするハイブリッドの例です。

```
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
```

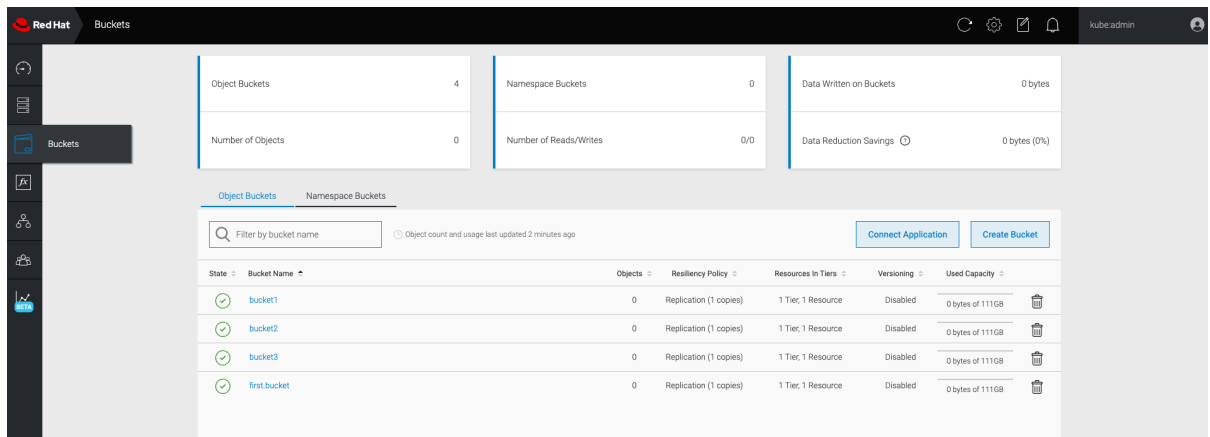
- 以下の行を標準の Object Bucket Claim (オブジェクトバケット要求、OBC) に追加します。

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

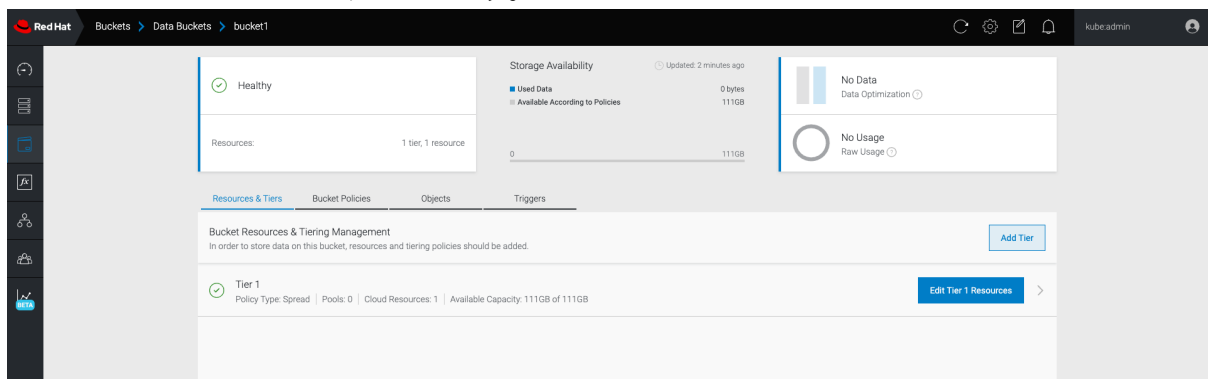
OBC についての詳細は、[10章 Object Bucket Claim\(オブジェクトバケット要求\)](#) を参照してください。

7.3. ユーザーインターフェイスを使用したデータミラーリングを行うためのバケットの設定

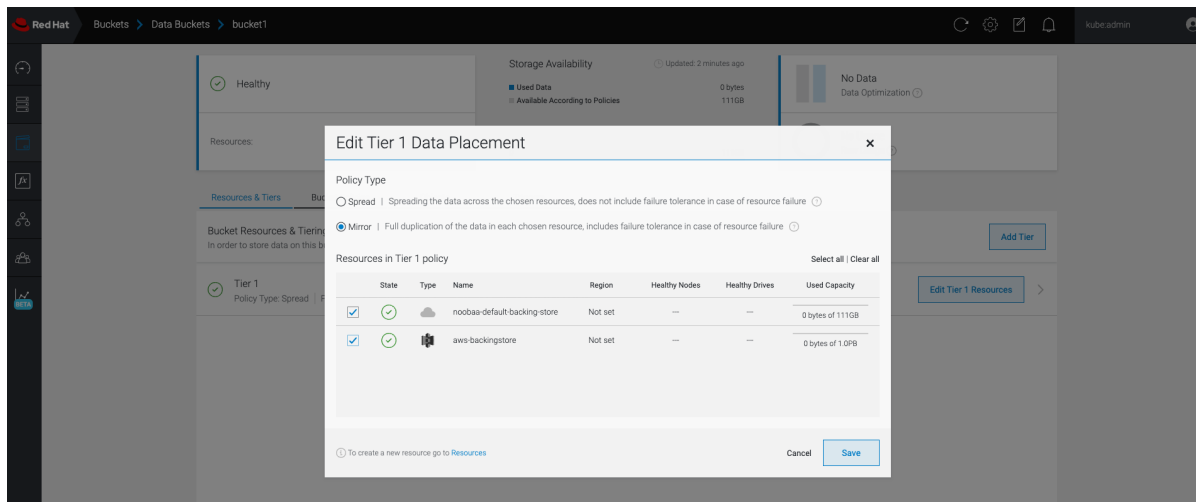
- OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。
- Status** カードで **Storage System** をクリックし、表示されるポップアップからストレージシステムリンクをクリックします。
- Object** タブで、**Multicloud Object Gateway** リンクをクリックします。
- NooBaa** ページの左側にある **buckets** アイコンをクリックします。バケットの一覧が表示されます。



- 更新するバケットをクリックします。
- Edit Tier 1 Resources** をクリックします。



7. **Mirror** を選択し、このバケットに使用する関連リソースを確認します。次の例では、RGW にある **noobaa-default-backing-store** と AWS にある **AWS-backingstore** の間のデータがミラーリングされます。



8. **Save** をクリックします。



注記

NooBaa UI で作成されたリソースは、OpenShift UI または Multicloud Object Gateway (MCG) CLI では使用できません。

第8章 MULTICLOUD OBJECT GATEWAY のバケットポリシー

OpenShift Data Foundation は AWS S3 バケットポリシーをサポートします。バケットポリシーにより、ユーザーにバケットとそれらのオブジェクトのアクセスパーミッションを付与することができます。

8.1. バケットポリシーについて

バケットポリシーは、AWS S3 バケットおよびオブジェクトにパーミッションを付与するために利用できるアクセスポリシーオプションです。バケットポリシーは JSON ベースのアクセスポリシー言語を使用します。アクセスポリシー言語についての詳細は、[AWS Access Policy Language Overview](#) を参照してください。

8.2. バケットポリシーの使用

前提条件

- 実行中の OpenShift Data Foundation Platform。
- Multicloud Object Gateway (MCG) へのアクセス。[2章アプリケーションの使用による Multicloud Object Gateway へのアクセス](#) を参照してください。

手順

MCG でバケットポリシーを使用するには、以下を実行します。

1. JSON 形式でバケットポリシーを作成します。以下の例を参照してください。

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

アクセスパーミッションに関して、バケットポリシーには数多くの利用可能な要素がありません。

これらの要素の詳細と、それらを使用してアクセスパーミッションを制御する方法の例は、[AWS Access Policy Language Overview](#) を参照してください。

バケットポリシーの他の例については、[AWS Bucket Policy Examples](#) を参照してください。

S3 ユーザーの作成方法については、「[Multicloud Object Gateway での AWS S3 ユーザーの作成](#)」を参照してください。

2. AWS S3 クライアントを使用して **put-bucket-policy** コマンドを使用してバケットポリシーを S3 バケットに適用します。

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

- a. **ENDPOINT** を S3 エンドポイントに置き換えます。
- b. **MyBucket** を、ポリシーを設定するバケットに置き換えます。
- c. **BucketPolicy** をバケットポリシー JSON ファイルに置き換えます。
- d. デフォルトの自己署名証明書を使用している場合は、**--no-verify-ssl** を追加します。以下に例を示します。

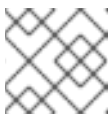
```
# aws --endpoint https://s3-openshift-storage.apps.gogo44.noobaa.org --no-verify-ssl s3api put-bucket-policy -bucket MyBucket --policy file://BucketPolicy
```

put-bucket-policy コマンドについての詳細は、[AWS CLI Command Reference for put-bucket-policy](#) を参照してください。



注記

主となる要素では、リソース (バケットなど) へのアクセスを許可または拒否されるユーザーを指定します。現在、NooBaa アカウントのみがプリンシパルとして使用できます。Object Bucket Claim (オブジェクトバケット要求) の場合、NooBaa はアカウント **obc-account.<generated bucket name>@noobaa.io** を自動的に作成します。



注記

バケットポリシー条件はサポートされていません。

8.3. MULTICLOUD OBJECT GATEWAY での AWS S3 ユーザーの作成

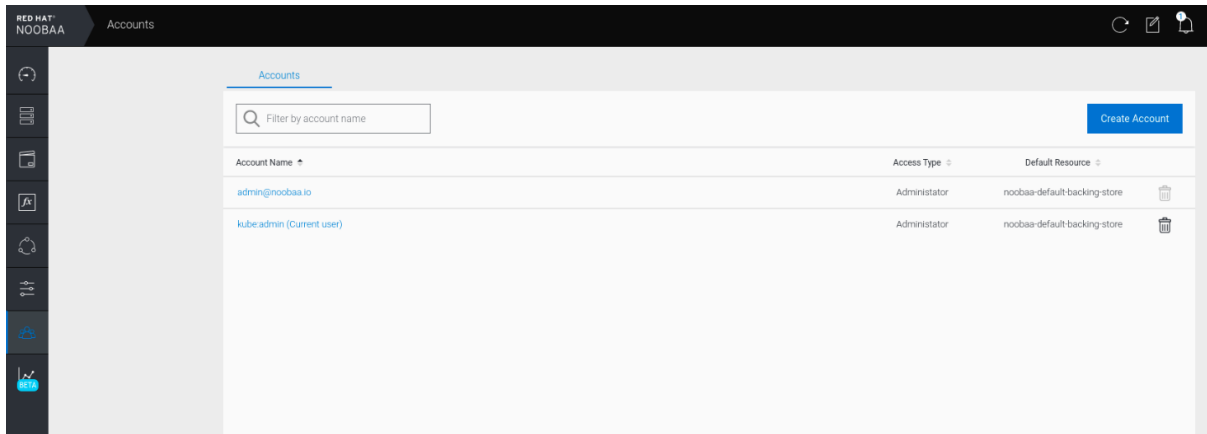
前提条件

- 実行中の OpenShift Data Foundation Platform。
- Multicloud Object Gateway (MCG) へのアクセス。[2章アプリケーションの使用による Multicloud Object Gateway へのアクセス](#) を参照してください。

手順

1. OpenShift Web コンソールで、**Storage → Data Foundation** をクリックします。
2. **Status** カードで **Storage System** をクリックし、表示されるポップアップからストレージシステムリンクをクリックします。
3. **Object** タブで、**Multicloud Object Gateway** リンクをクリックします。

4. Accounts タブで、Create Account をクリックします。



5. S3 Access Only を選択し、Account Name を指定します (例: john.doe@example.com)。Next をクリックします。

Create Account ×

1 Account Details
2 S3 Access

Access Type:

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

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

Account Name:

3 - 32 characters

Cancel Next

6. S3 default placement を選択します (例: noobaa-default-backing-store)。Buckets Permissions を選択します。特定のバケットまたはすべてのバケットを選択できます。Create をクリックします。

Create Account ✕

✔ Account Details 2 S3 Access

S3 default placement: ? ▼

Buckets Permissions: ▼

Include any future buckets

Allow new bucket creation: ? Enabled

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

第9章 MULTICLOUD OBJECT GATEWAY バケットレプリケーション

1つの Multicloud Object Gateway(MCG) バケットから別の MCG バケットへのデータレプリケーションは、より高い復元力とより優れたコラボレーションオプションを提供します。これらのバケットは、サポート対象のストレージソリューション (S3、Azure など) でサポートされるデータバケットまたは namespace バケットのいずれかになります。

レプリケーションポリシーは、レプリケーションルールの一覧で設定されます。各ルールは宛先バケットを定義し、オブジェクトキーの接頭辞に基づいてフィルターを指定できます。2番目のバケットで補完的なレプリケーションポリシーを設定すると、双方向レプリケーションが実行されます。

前提条件

- 実行中の OpenShift Data Foundation Platform。
- Multicloud Object Gateway へのアクセスについては、Accessing the Multicloud Object Gateway with your applications を参照してください。
- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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



重要

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。たとえば、IBM Power の場合は、次のコマンドを使用します。

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

- または、**mcg** パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/packages にある OpenShift Data Foundation RPM からインストールできます。



重要

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。



注記

特定の MCG 機能は特定の MCG バージョンでのみ利用でき、適切な MCG CLI ツールのバージョンを使用して MCG の機能を完全に活用する必要があります。

バケットを複製するには、[バケットの別のバケットへの複製](#) を参照してください。

バケットクラスレプリケーションポリシーを設定するには、[バケットクラスのレプリケーションポリシーの設定](#) を参照してください。

9.1. バケットの別のバケットへの複製

バケットレプリケーションポリシーは、次の2つの方法で設定できます。

- MCG コマンドラインインターフェイスを使用したバケットの別のバケットへの複製。
- YAML を使用したバケットの別のバケットへの複製

9.1.1. MCG コマンドラインインターフェイスを使用してバケットの別のバケットへの複製

Multicloud Object Gateway (MCG) バケットに特定のレプリケーションポリシーが必要なアプリケーションは、Object Bucket Claim (OBC) を作成し、JSON ファイルで **replication policy** パラメーターを定義できます。

手順

- MCG コマンドラインインターフェイスから以下のコマンドを実行し、特定のレプリケーションポリシーで OBC を作成します。

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

<bucket-claim-name>

バケットクレームの名前を指定します。

/path/to/json-file.json

レプリケーションポリシーを定義する JSON ファイルへのパスです。

JSON ファイルの例:

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

"prefix"

これは任意になります。複製する必要があるのはオブジェクトキーの接頭辞であり、たとえば {"prefix": ""} のように、空のままにすることもできます。

例9.1 例

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

9.1.2. YAML を使用してバケットを別のバケットに複製

Multicloud Object Gateway (MCG) データバケットを必要とするアプリケーションは、特定のレプリケーションポリシーを持つことができます。また、Object Bucket Claim (OBC) を作成し、**spec.additionalConfig.replication-policy** パラメーターを OBC に追加できます。

手順

- 以下の YAML を適用します。

```
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>

バケットクレームの名前を指定します。

<desired-namespace>

namespace を指定します。

<desired-bucket-name>

バケット名の接頭辞を指定します。

"rule_id"

ルールの ID 番号を指定します (例: {"rule_id": "rule-1"}).

"destination_bucket"

宛先バケットの名前を指定します (例: {"destination_bucket": "first.bucket"}).

"prefix"

これは任意になります。複製する必要があるのはオブジェクトキーの接頭辞であり、たとえば {"prefix": ""} のように、空のままにすることもできます。

追加情報

- OBC についての詳細は、[Object Bucket Claim](#) を参照してください。

9.2. バケットクラスのレプリケーションポリシーの設定

特定のバケットクラスで作成されたすべてのバケットに自動的に適用されるレプリケーションポリシーを設定することができます。これは、以下の2つの方法で実行できます。

- [MCG コマンドラインインターフェイスを使用したバケットクラスのレプリケーションポリシーの設定](#)。
- [YAML を使用したバケットクラスのレプリケーションポリシーの設定](#)。

9.2.1. MCG コマンドラインインターフェイスを使用したバケットクラスのレプリケーションポリシーの設定

マルチクラウドオブジェクトゲートウェイ (MCG) バケットクラスに特定のレプリケーションポリシーが必要なアプリケーションは、**bucketclass** を作成し、JSON ファイルで **replication-policy** パラメーターを定義できます。

次の2種類のバケットクラスにバケットクラスレプリケーションポリシーを設定できます。

- Placement
- Namespace

手順

- MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

<bucketclass-name>

バケットクラスの名前を指定します。

<backingstores>

バックングストアの名前を指定します。複数のバックングストアをコンマで区切って渡すことができます。

/path/to/json-file.json

レプリケーションポリシーを定義する JSON ファイルへのパスです。

JSON ファイルの例:

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

"prefix"

これは任意になります。複製する必要があるのはオブジェクトキーの接頭辞であり、たとえば {"prefix": ""} のように、空のままにすることもできます。

例9.2 例

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

この例では、JSON ファイルで定義された特定のレプリケーションポリシーを使用して配置バケットクラスを作成します。

9.2.2. YAML を使用したバケットクラスのレプリケーションポリシーの設定

マルチクラウドオブジェクトゲートウェイ (MCG) バケットクラスに特定のレプリケーションポリシーが必要なアプリケーションは、**spec.replicationPolicy** フィールドを使用してバケットクラスを作成できます。

手順

1. 以下の YAML を適用します。

```
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>"}]}
```

このYAMLは、配置バケットクラスを作成する例です。バケットにアップロードされた各オブジェクトバケットクレーム(OBC)オブジェクトは、接頭辞に基づいてフィルタリングされ、**first.bucket**に複製されます。

<desired-app-label>

アプリのラベルを指定します。

<desired-bucketclass-name>

バケットクラス名を指定します。

<desired-namespace>

バケットクラスが作成される namespace を指定します。

<backingstore>

バックングストアの名前を指定します。複数のバックングストアを通過することが可能です。

"rule_id"

ルールのID番号を指定します(例: `{"rule_id": "rule-1"}`)。

"destination_bucket"

宛先バケットの名前を指定します(例: `{"destination_bucket": "first.bucket"}`)。

"prefix"

これは任意になります。複製する必要があるのはオブジェクトキーの接頭辞であり、たとえば `{"prefix": ""}` のように、空のままにすることもできます。

第10章 OBJECT BUCKET CLAIM(オブジェクトバケット要求)

Object Bucket Claim(オブジェクトバケット要求)は、ワークロードの S3 と互換性のあるバケットバックエンドを要求するために使用できます。

Object Bucket Claim(オブジェクトバケット要求)は3つの方法で作成できます。

- 「動的 Object Bucket Claim(オブジェクトバケット要求)」
- 「コマンドラインインターフェイスを使用した Object Bucket Claim(オブジェクトバケット要求)の作成」
- 「OpenShift Web コンソールを使用した Object Bucket Claim(オブジェクトバケット要求)の作成」

Object Bucket Claim(オブジェクトバケット要求)は、新しいアクセスキーおよびシークレットアクセスキーを含む、バケットのパーミッションのある NooBaa の新しいバケットとアプリケーションアカウントを作成します。アプリケーションアカウントは単一バケットにのみアクセスでき、デフォルトで新しいバケットを作成することはできません。

10.1. 動的 OBJECT BUCKET CLAIM(オブジェクトバケット要求)

永続ボリュームと同様に、Object Bucket Claim (OBC) の詳細をアプリケーションの YAML に追加し、設定マップおよびシークレットで利用可能なオブジェクトサービスエンドポイント、アクセスキー、およびシークレットアクセスキーを取得できます。この情報をアプリケーションの環境変数に動的に読み込むことは容易に実行できます。



注記

Multicloud Object Gateway エンドポイントは、OpenShift が自己署名証明書を使用する場合にのみ、自己署名証明書を使用します。OpenShift で署名付き証明書を使用すると、Multicloud Object Gateway エンドポイント証明書が署名付き証明書に自動的に置き換えられます。ブラウザーを介してエンドポイントにアクセスし、Multicloud Object Gateway で現在使用されている証明書を取得します。詳細は、[アプリケーションの使用による Multicloud Object Gateway へのアクセス](#) を参照してください。

手順

1. 以下の行をアプリケーション YAML に追加します。

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

これらの行は OBC 自体になります。

- a. **<obc-name>** を、一意の OBC の名前に置き換えます。
 - b. **<obc-bucket-name>** を、OBC の一意のバケット名に置き換えます。
2. YAML ファイルにさらに行を追加して、OBC の使用を自動化できます。以下の例はバケット要

求の結果のマッピングです。これは、データを含む設定マップおよび認証情報のあるシークレットです。この特定のジョブは NooBaa からオブジェクトバケットを要求し、バケットとアカウントを作成します。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - image: <your application image>
        name: test
        env:
        - name: BUCKET_NAME
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_NAME
        - name: BUCKET_HOST
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_HOST
        - name: BUCKET_PORT
          valueFrom:
            configMapKeyRef:
              name: <obc-name>
              key: BUCKET_PORT
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: <obc-name>
              key: AWS_ACCESS_KEY_ID
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: <obc-name>
              key: AWS_SECRET_ACCESS_KEY
```

- a. **<obc-name>** のすべてのインスタンスを、OBC の名前に置き換えます。
 - b. **<your application image>** をアプリケーションイメージに置き換えます。
3. 更新された YAML ファイルを適用します。

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

<yaml.file> を YAML ファイルの名前に置き換えます。

4. 新しい設定マップを表示するには、以下を実行します。

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


obc-name を OBC の名前に置き換えます。

出力には、以下の環境変数が表示されることが予想されます。

- **BUCKET_HOST**: アプリケーションで使用するエンドポイント
- **BUCKET_PORT**: アプリケーションで利用できるポート
 - ポートは **BUCKET_HOST** に関連します。たとえば、**BUCKET_HOST** が <https://my.example.com> で、**BUCKET_PORT** が 443 の場合、オブジェクトサービスのエンドポイントは <https://my.example.com:443> になります。
- **BUCKET_NAME**: 要求されるか、生成されるバケット名
- **AWS_ACCESS_KEY_ID**: 認証情報の一部であるアクセスキー
- **AWS_SECRET_ACCESS_KEY**: 認証情報の一部であるシークレットのアクセスキー

重要

AWS_ACCESS_KEY_ID と **AWS_SECRET_ACCESS_KEY** を取得します。名前は、AWS S3 と互換性があるように使用されます。S3 操作の実行中、特に Multicloud Object Gateway (MCG) バケットから読み取り、書き込み、または一覧表示する場合は、キーを指定する必要があります。キーは Base64 でエンコードされています。キーを使用する前に、キーをデコードしてください。

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

<obc_name>

オブジェクトバケットクレームの名前を指定します。

10.2. コマンドラインインターフェイスを使用した OBJECT BUCKET CLAIM(オブジェクトバケット要求)の作成

コマンドラインインターフェイスを使用して Object Bucket Claim (OBC) を作成する場合、設定マップとシークレットを取得します。これらには、アプリケーションがオブジェクトストレージサービスを使用するために必要なすべての情報が含まれます。

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。

- IBM Power の場合は、次のコマンドを使用します。

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

- IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

手順

1. コマンドラインインターフェイスを使用して、新規バケットおよび認証情報の詳細を生成します。以下のコマンドを実行します。

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

<obc-name> を一意の OBC 名に置き換えます (例: **myappobc**)。

さらに、**--app-namespace** オプションを使用して、OBC 設定マップおよびシークレットが作成される namespace を指定できます (例: **myapp-namespace**)。

出力例:

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

MCG コマンドラインインターフェイスが必要な設定を作成し、新規 OBC について OpenShift に通知します。

2. 以下のコマンドを実行して OBC を表示します。

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

出力例:

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

3. 以下のコマンドを実行して、新規 OBC の YAML ファイルを表示します。

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

出力例:

```
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. **openshift-storage** namespace 内で、設定マップおよびシークレットを見つけ、この OBC を使用することができます。CM とシークレットの名前はこの OBC の名前と同じです。以下のコマンドを実行してシークレットを表示します。

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

出力例:

```

Example output:
apiVersion: v1
data:
  AWS_ACCESS_KEY_ID: c0M0R2xVanF3ODR3bHBkVW94cmY=
  AWS_SECRET_ACCESS_KEY:
  Wi9kcFluSWxHRzIWaFlzNk1hc0xma2JXcjM1MVhqa051SIBleXpmOQ==
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

```

シークレットは S3 アクセス認証情報を提供します。

5. 以下のコマンドを実行して設定マップを表示します。

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

出力例:

```
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
```

設定マップには、アプリケーションの S3 エンドポイント情報が含まれます。

10.3. OPENSIFT WEB コンソールを使用した OBJECT BUCKET CLAIM(オブジェクトバケット要求)の作成

OpenShift Web コンソールを使用して Object Bucket Claim (オブジェクトバケット要求) を作成できません。

前提条件

- OpenShift Web コンソールへの管理者アクセス。
- アプリケーションが OBC と通信できるようにするには、configmap およびシークレットを使用する必要があります。これに関する詳細情報は、「[動的 Object Bucket Claim\(オブジェクトバケット要求\)](#)」を参照してください。

手順

1. OpenShift Web コンソールにログインします。
2. 左側のナビゲーションバーで **Storage → Object Bucket Claims → Create Object Bucket Claim** をクリックします。
 - a. Object Bucket Claim(オブジェクトバケット要求) の名前を入力し、ドロップダウンメニューから、内部または外部かのデプロイメントに応じて適切なストレージクラスとバケットクラスを選択します。

内部モード

Project: openshift-storage ▼

Create Object Bucket Claim [Edit YAML](#)

Object Bucket Claim Name

If not provided, a generic name will be generated.

Storage Class *

Select storage class ▼

Select storage class

No default storage class

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

デプロイメント後に作成された以下のストレージクラスを使用できます。

- **ocs-storagecluster-ceph-rgw** は Ceph Object Gateway (RGW) を使用します。
- **openshift-storage.noobaa.io** は Multicloud Object Gateway (MCG) を使用します。

外部モード

Project: openshift-storage ▼

Create Object Bucket Claim

[Edit YAML](#)

Object Bucket Claim Name

If not provided, a generic name will be generated.

Storage Class *

No default storage class

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

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

デプロイメント後に作成された以下のストレージクラスを使用できます。

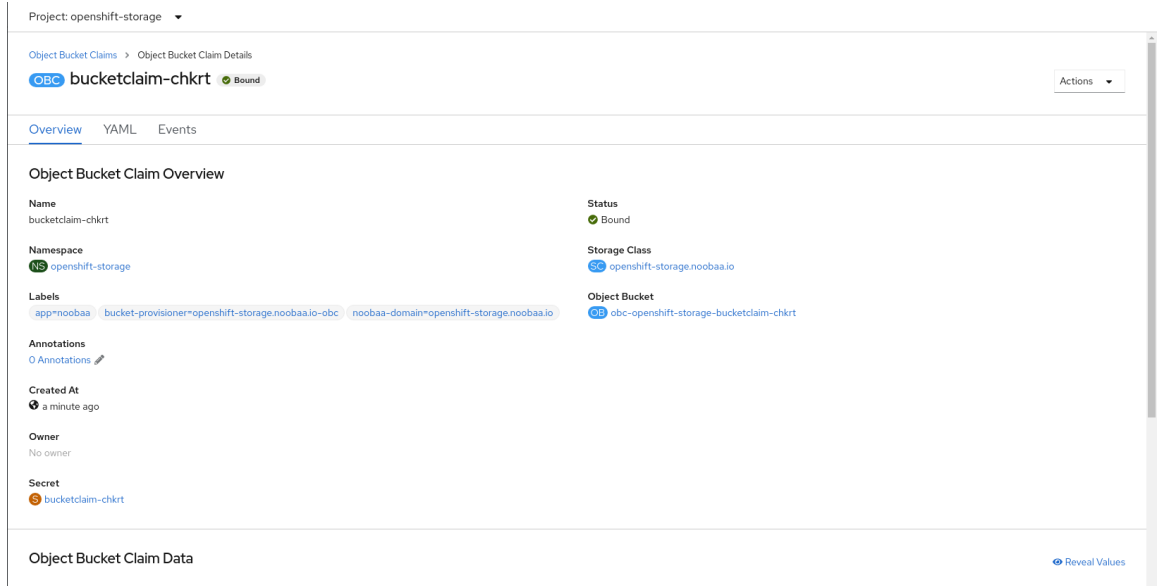
- **ocs-external-storagecluster-ceph-rgw** は RGW を使用します。
- **openshift-storage.noobaa.io** は MCG を使用します。



注記

RGW OBC ストレージクラスは、OpenShift Data Foundation バージョン 4.5 の新規インストールでのみ利用できます。これは、以前の OpenShift Data Foundation リリースからアップグレードされたクラスターには適用されません。

- b. **Create** をクリックします。
OBC を作成すると、その詳細ページにリダイレクトされます。



関連情報

- [10章 Object Bucket Claim\(オブジェクトバケット要求\)](#)

10.4. OBJECT BUCKET CLAIM(オブジェクトバケット要求)のデプロイメントへの割り当て

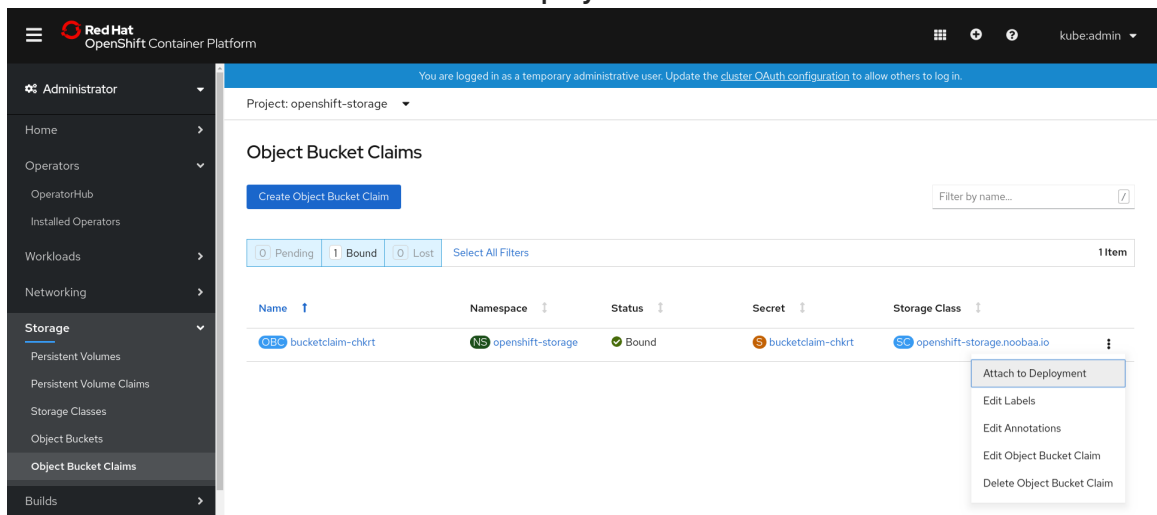
Object Bucket Claim(オブジェクトバケット要求、OBC)は作成後に、特定のデプロイメントに割り当てることができます。

前提条件

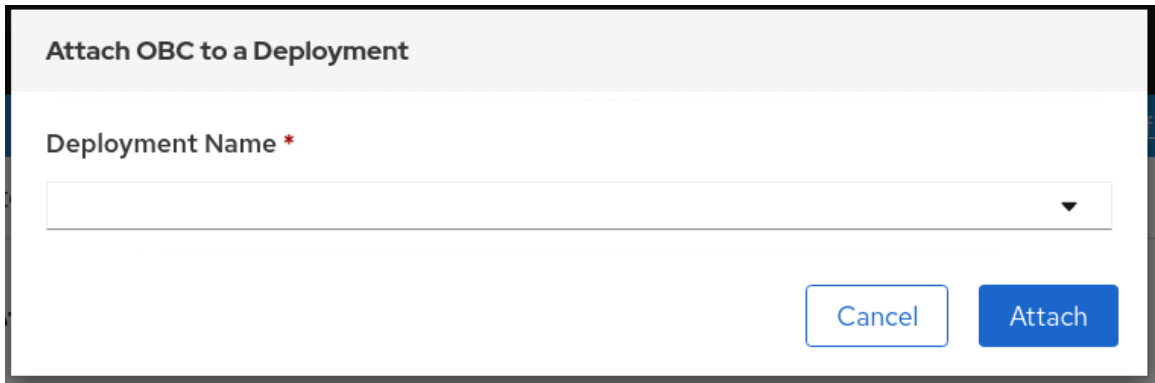
- OpenShift Web コンソールへの管理者アクセス。

手順

1. 左側のナビゲーションバーで **Storage** → **Object Bucket Claims** をクリックします。
2. 作成した OBC の横にあるアクションメニュー (⋮) をクリックします。
 - a. ドロップダウンメニューで、**Attach to Deployment** を選択します。



- b. Deployment Name 一覧から必要なデプロイメントを選択し、**Attach** をクリックします。



関連情報

- [10章 Object Bucket Claim\(オブジェクトバケット要求\)](#)

10.5. OPENSIFT WEB コンソールを使用したオブジェクトバケットの表示

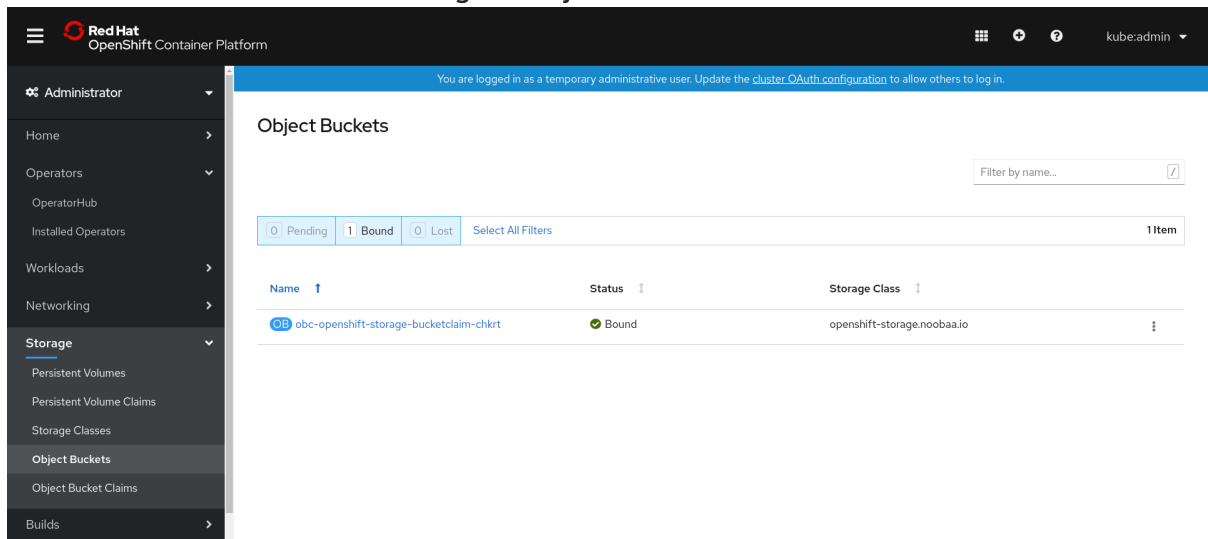
OpenShift Web コンソールを使用して、Object Bucket Claim(オブジェクトバケット要求、OBC)用に作成されたオブジェクトバケットの詳細を表示できます。

前提条件

- OpenShift Web コンソールへの管理者アクセス。

手順

1. OpenShift Web コンソールにログインします。
2. 左側のナビゲーションバーで **Storage** → **Object Buckets** をクリックします。



また、特定の OBC の詳細ページに移動し、**Resource** リンクをクリックして、その OBC のオブジェクトバケットを表示します。

3. 詳細を表示するオブジェクトバケットを選択します。Object Bucket Details ページに移動します。

関連情報

- [10章 Object Bucket Claim\(オブジェクトバケット要求\)](#)

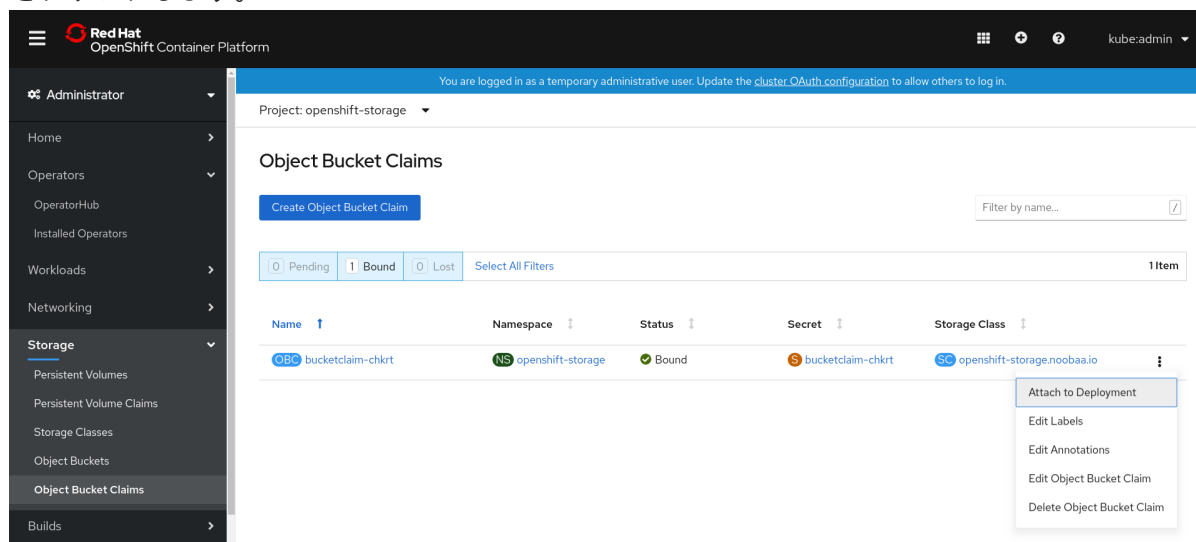
10.6. OBJECT BUCKET CLAIM(オブジェクトバケット要求) の削除

前提条件

- OpenShift Web コンソールへの管理者アクセス。

手順

1. 左側のナビゲーションバーで **Storage** → **Object Bucket Claims** をクリックします。
2. 削除する Object Bucket Claim(オブジェクトバケット要求) の横にあるアクションメニュー (⋮) をクリックします。



- a. **Delete Object Bucket Claim** を選択します。
- b. **Delete** をクリックします。

関連情報

- [10章 Object Bucket Claim\(オブジェクトバケット要求\)](#)

第11章 オブジェクトバケットのキャッシュポリシー

キャッシュバケットは、ハブのターゲットとキャッシュターゲットが指定された namespace バケットです。ハブのターゲットは、S3 と互換性のある大規模なオブジェクトストレージバケットです。キャッシュのバケットは、ローカルの Multicloud Object Gateway (MCG) バケットです。AWS バケットまたは IBM COS バケットをキャッシュするキャッシュバケットを作成できます。

- [AWS S3](#)
- [IBM COS](#)

11.1. AWS キャッシュバケットの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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

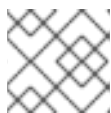


注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/package にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. NamespaceStore リソースを作成します。NamespaceStore は、MCG namespace バケットでデータの読み取りおよび書き込みターゲットとして使用される基礎となるストレージを表します。MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

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

- a. **<namespacestore>** を namespacestore の名前に置き換えます。
- b. **<AWS ACCESS KEY>** および **<AWS SECRET ACCESS KEY>** を、作成した AWS アクセスキー ID およびシークレットアクセスキーに置き換えます。
- c. **<bucket-name>** を既存の AWS バケット名に置き換えます。この引数は、MCG に対して、バックストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。

YAML を適用してストレージリソースを追加することもできます。まず、認証情報を使用してシークレットを作成します。

```
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>
```

Base64 を使用して独自の AWS アクセスキー ID およびシークレットアクセスキーを指定し、エンコードし、その結果を **<AWS ACCESS KEY ID ENCODED IN BASE64>** および **<AWS SECRET ACCESS KEY ENCODED IN BASE64>** に使用する必要があります。

<namespacestore-secret-name> を一意の名前に置き換えます。

次に、以下の YAML を適用します。

```
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. **<namespacestore>** を一意の名前に置き換えます。
 - e. **<namespacestore-secret-name>** を、直前の手順で作成されたシークレットに置き換えます。
 - f. **<namespace-secret>** を、直前の手順でシークレットを作成するために使用された namespace に置き換えます。
 - g. **<target-bucket>** を namespacestore 用に作成した AWS S3 バケットに置き換えます。
2. 以下のコマンドを実行してバケットクラスを作成します。

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

- a. **<my-cache-bucket-class>** を一意のバケットクラス名に置き換えます。

- b. **<backing-store>** を関連するバックングストアに置き換えます。コンマで区切られた1つ以上のバックングストアを一覧表示できます。
 - c. **<namespacestore>** を、直前の手順で作成された namespacestore に置き換えます。
3. 以下のコマンドを実行して、手順2に定義されたバケットクラスを使用する Object Bucket Claim (OBC) リソースを使用してバケットを作成します。

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

- a. **<my-bucket-claim>** を一意の名前に置き換えます。
- b. **<custom-bucket-class>** を、手順2で作成したバケットクラスの名前に置き換えます。

11.2. IBM COS キャッシュバケットの作成

前提条件

- Multicloud Object Gateway (MCG) コマンドラインインターフェイスをダウンロードします。

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



注記

サブスクリプションマネージャーを使用してリポジトリを有効にするための適切なアーキテクチャーを指定します。

- IBM Power の場合は、次のコマンドを使用します。

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

- IBM Z インフラストラクチャーの場合は、以下のコマンドを使用します。

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

または、MCG パッケージを、https://access.redhat.com/downloads/content/547/ver=4/rhel--8/4/x86_64/package にある OpenShift Data Foundation RPM からインストールできます。



注記

お使いのアーキテクチャーに応じて、正しい製品バリエーションを選択します。

手順

1. NamespaceStore リソースを作成します。NamespaceStore は、MCG namespace バケットでデータの読み取りおよび書き込みターゲットとして使用される基礎となるストレージを表します。MCG コマンドラインインターフェイスから、以下のコマンドを実行します。

```
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. **<namespacestore>** を NamespaceStore の名前に置き換えます。
- b. **<IBM ACCESS KEY>**, **<IBM SECRET ACCESS KEY>**, **<IBM COS ENDPOINT>** を IBM アクセスキー ID、シークレットアクセスキー、および既存の IBM バケットの場所に対応する地域のエンドポイントに置き換えます。
- c. **<bucket-name>** を既存の IBM バケット名に置き換えます。この引数は、MCG に対して、バックングストア、およびその後のデータストレージおよび管理のためのターゲットバケットとして使用するバケットについて指示します。
YAML を適用してストレージリソースを追加することもできます。まず、認証情報を使用してシークレットを作成します。

```
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>
```

Base64 を使用して独自の IBM COS アクセスキー ID およびシークレットアクセスキーを指定し、エンコードし、その結果を **<IBM COS ACCESS KEY ID ENCODED IN BASE64>** および **<IBM COS SECRET ACCESS KEY ENCODED IN BASE64>** に使用する必要があります。

<namespacestore-secret-name> を一意の名前に置き換えます。

次に、以下の YAML を適用します。

```
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. **<namespacestore>** を一意の名前に置き換えます。
- e. **<IBM COS ENDPOINT>** を適切な IBM COS エンドポイントに置き換えます。
- f. **<backingstore-secret-name>** を、直前の手順で作成されたシークレットに置き換えます。

- g. **<namespace-secret>** を、直前の手順でシークレットを作成するために使用された namespace に置き換えます。
 - h. **<target-bucket>** を namespacestore 用に作成した AWS S3 バケットに置き換えます。
2. 以下のコマンドを実行してバケットクラスを作成します。

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

- a. **<my-bucket-class>** を一意のバケットクラス名に置き換えます。
 - b. **<backing-store>** を関連するバックングストアに置き換えます。コンマで区切られた1つ以上のバックングストアを一覧表示できます。
 - c. **<namespacestore>** を、直前の手順で作成された namespacestore に置き換えます。
3. 以下のコマンドを実行して、手順2に定義されたバケットクラスを使用する Object Bucket Class リソースを使用してバケットを作成します。

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

- a. **<my-bucket-claim>** を一意の名前に置き換えます。
- b. **<custom-bucket-class>** を、手順2で作成したバケットクラスの名前に置き換えます。

第12章 エンドポイントの追加による MULTICLOUD OBJECT GATEWAY パフォーマンスのスケールリング

Multicloud Object Gateway (MCG) のパフォーマンスは環境によって異なる場合があります。特定のアプリケーションでは、高速なパフォーマンスを必要とする場合があります、これは S3 エンドポイントをスケールリングして簡単に対応できます。

MCG リソースプールは、デフォルトで有効にされる 2 種類のサービスを提供する NooBaa デーモンコンテナのグループです。

- ストレージサービス
- S3 エンドポイントサービス

S3 エンドポイントサービス

S3 エンドポイントは、すべての MCG がデフォルトで提供するサービスであり、これは Multicloud Object Gateway (MCG) で負荷の高いデータ消費タスクの大部分を処理します。エンドポイントサービスは、インラインのデータチャンク、重複排除、圧縮、および暗号化を処理し、(MCG) からのデータ配置の指示を受け入れます。

12.1. MULTICLOUD OBJECT GATEWAY エンドポイントの自動スケールリング

MultiCloud Object Gateway (MCG) の S3 サービスの負荷が増減すると、MCG エンドポイントの数が自動的にスケールリングされます。OpenShift Data Foundation クラスタは、アクティブな MCG エンドポイントを 1 つ使用してデプロイされます。デフォルトでは、MCG エンドポイント Pod はそれぞれ、CPU 1 つ、メモリー要求 2 Gi、要求に一致する制限で設定されます。エンドポイントの CPU 負荷が一貫した期間、使用率 80% のしきい値を超えると、2 番目のエンドポイントがデプロイされ、最初のエンドポイントの負荷を軽減します。両方のエンドポイントの平均 CPU 負荷が、一貫した期間 80% のしきい値を下回ると、エンドポイントの 1 つが削除されます。この機能により、MCG のパフォーマンスおよび保守性が向上します。

12.2. ストレージノードを使用した MULTICLOUD OBJECT GATEWAY のスケールリング

前提条件

- Multicloud Object Gateway (MCG) にアクセスできる OpenShift Container Platform で実行中の OpenShift Data Foundation クラスタ。

MCG のストレージノードは 1 つ以上の永続ボリューム (PV) に割り当てられた NooBaa デーモンコンテナであり、ローカルオブジェクトサービスデータストレージに使用されます。NooBaa デーモンは Kubernetes ノードにデプロイできます。これは、StatefulSet Pod で設定される Kubernetes プールを作成して実行できます。

手順

1. OpenShift Web Console にログインします。
2. MCG ユーザーインターフェイスから **Overview** → **Add Storage Resources** をクリックします。
3. ウィンドウから **Deploy Kubernetes Pool** をクリックします。

4. **Create Pool**手順で、今後インストールされるノードのターゲットプールを作成します。
5. **Configure** 手順で、要求される Pod 数と各 PV のサイズを設定します。新規 Pod ごとに、1つの PV が作成されます。
6. **Review** 手順で、新規プールの詳細を検索し、ローカルまたは外部デプロイメントのいずれかの使用するデプロイメント方法を選択します。ローカルデプロイメントが選択されている場合、Kubernetes ノードはクラスター内にデプロイされます。外部デプロイメントが選択されている場合、外部で実行するための YAML ファイルが提供されます。
7. すべてのノードは最初の手順で選択したプールに割り当てられ、**Resources** → **Storage resources** → **Resource name** の下で確認できます。

第13章 RADOS OBJECT GATEWAY S3 エンドポイントへのアクセス

ユーザーは、RADOS Object Gateway (RGW) エンドポイントに直接アクセスできます。

Red Hat OpenShift Data Foundation の以前のバージョンでは、RGW パブリックルートを作成するために RGW サービスを手動で公開する必要がありました。OpenShift Data Foundation バージョン 4.7 以降、RGW ルートはデフォルトで作成され、**rook-ceph-rgw-ocs-storagecluster-cephobjectstore** という名前が付けられています。