Red Hat OpenShift Data Foundation 4.11

Red Hat OpenShift Data Foundation architecture

Overview of OpenShift Data Foundation architecture and the roles that the components and services perform.

Last Updated: 2024-01-17

Red Hat OpenShift Data Foundation 4.11 Red Hat OpenShift Data Foundation architecture

Overview of OpenShift Data Foundation architecture and the roles that the components and services perform.

Legal Notice

Abstract

This document provides an overview of the OpenShift Data Foundation architecture.

Table of Contents

PREFACE

This document provides an overview of the OpenShift Data Foundation architecture.

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

To give feedback, create a Bugzilla ticket:

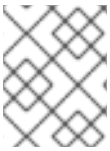
1. Go to the Bugzilla website.
2. In the **Component** section, choose **documentation**.
3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
4. Click **Submit Bug**.

CHAPTER 1. INTRODUCTION TO OPENSHIFT DATA FOUNDATION

Red Hat OpenShift Data Foundation is a highly integrated collection of cloud storage and data services for Red Hat OpenShift Container Platform. It is available as part of the Red Hat OpenShift Container Platform Service Catalog, packaged as an operator to facilitate simple deployment and management.

Red Hat OpenShift Data Foundation services are primarily made available to applications by way of storage classes that represent the following components:

- Block storage devices, catering primarily to database workloads. Prime examples include Red Hat OpenShift Container Platform logging and monitoring, and PostgreSQL.
- Shared and distributed file system, catering primarily to software development, messaging, and data aggregation workloads. Examples include Jenkins build sources and artifacts, Wordpress uploaded content, Red Hat OpenShift Container Platform registry, and messaging using JBoss AMQ.
- Multicloud object storage, featuring a lightweight S3 API endpoint that can abstract the storage and retrieval of data from multiple cloud object stores.
- On premises object storage, featuring a robust S3 API endpoint that scales to tens of petabytes and billions of objects, primarily targeting data intensive applications. Examples include the storage and access of row, columnar, and semi-structured data with applications like Spark, Presto, Red Hat AMQ Streams (Kafka), and even machine learning frameworks like TensorFlow and Pytorch.



NOTE

Running PostgresSQL workload on CephFS persistent volume is not supported and it is recommended to use RADOS Block Device (RBD) volume.

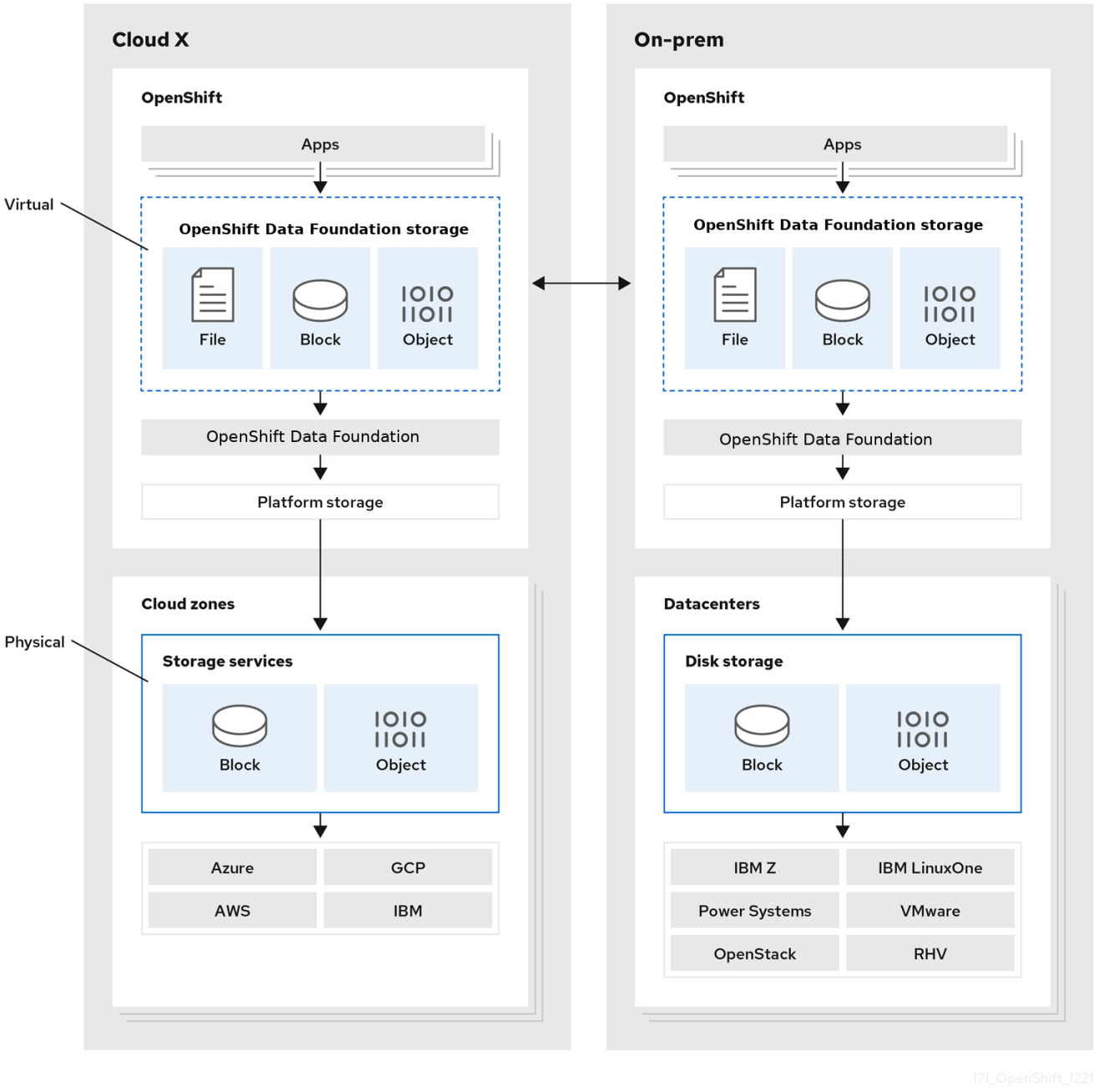
Red Hat OpenShift Data Foundation version 4.x integrates a collection of software projects, including:

- Ceph, providing block storage, a shared and distributed file system, and on-premises object storage
- Ceph CSI, to manage provisioning and lifecycle of persistent volumes and claims
- NooBaa, providing a Multicloud Object Gateway
- OpenShift Data Foundation, Rook-Ceph, and NooBaa operators to initialize and manage OpenShift Data Foundation services.

CHAPTER 2. AN OVERVIEW OF OPENSHIFT DATA FOUNDATION ARCHITECTURE

Red Hat OpenShift Data Foundation provides services for, and can run internally from Red Hat OpenShift Container Platform.

Figure 2.1. Red Hat OpenShift Data Foundation architecture



Red Hat OpenShift Data Foundation supports deployment into Red Hat OpenShift Container Platform clusters deployed on Installer Provisioned Infrastructure or User Provisioned Infrastructure. For details about these two approaches, see OpenShift Container Platform - Installation process. To know more about interoperability of components for the Red Hat OpenShift Data Foundation and Red Hat OpenShift Container Platform, see the interoperability matrix.

For information about the architecture and lifecycle of OpenShift Container Platform, see OpenShift Container Platform architecture.

CHAPTER 3. OPENSHIFT DATA FOUNDATION OPERATORS

Red Hat OpenShift Data Foundation is comprised of the following three Operator Lifecycle Manager (OLM) operator bundles, deploying four operators which codify administrative tasks and custom resources so that task and resource characteristics can be easily automated:

- OpenShift Data Foundation
  - **odf-operator**
- OpenShift Container Storage
  - **ocs-operator**
  - **rook-ceph-operator**
- Multicloud Object Gateway
  - **mcg-operator**

Administrators define the desired end state of the cluster, and the OpenShift Data Foundation operators ensure the cluster is either in that state or approaching that state, with minimal administrator intervention.

3.1. OPENSHIFT DATA FOUNDATION OPERATOR

The **odf-operator** can be described as a "meta" operator for OpenShift Data Foundation, that is, an operator meant to influence other operators.

The **odf-operator** has the following primary functions:

- Enforces the configuration and versioning of the other operators that comprise OpenShift Data Foundation. It does this by using two primary mechanisms: operator dependencies and Subscription management.
  - The **odf-operator** bundle specifies dependencies on other OLM operators to make sure they are always installed at specific versions.
  - The operator itself manages the Subscriptions for all other operators to make sure the desired versions of those operators are available for installation by the OLM.
- Provides the OpenShift Data Foundation external plugin for the OpenShift Console.
- Provides an API to integrate storage solutions with the OpenShift Console.

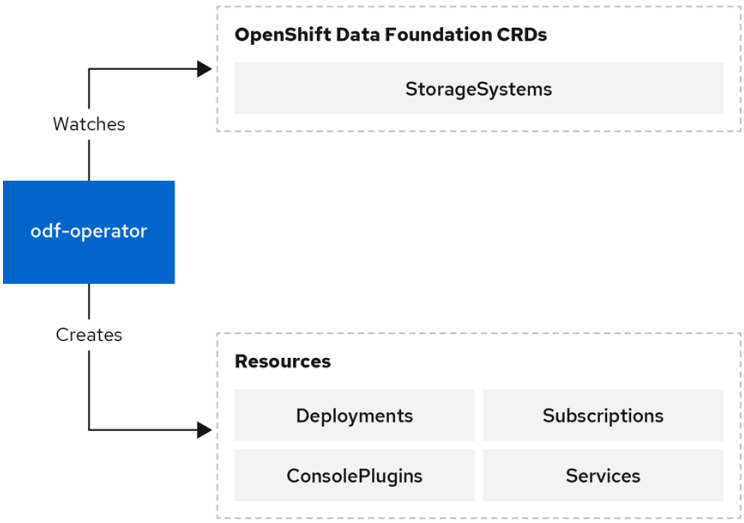
3.1.1. Components

The **odf-operator** has a dependency on the **ocs-operator** package. It also manages the Subscription of the **mcg-operator**. In addition, the **odf-operator** bundle defines a second Deployment for the OpenShift Data Foundation external plugin for the OpenShift Console. This defines an **nginx**-based Pod that serves the necessary files to register and integrate OpenShift Data Foundation dashboards directly into the OpenShift Container Platform Console.

3.1.2. Design diagram

This diagram illustrates how **odf-operator** is integrated with the OpenShift Container Platform.

Figure 3.1. OpenShift Data Foundation Operator



3.1.3. Responsibilites

The odf-operator defines the following CRD:

- **StorageSystem**

The **StorageSystem** CRD represents an underlying storage system that provides data storage and services for OpenShift Container Platform. It triggers the operator to ensure the existence of a **Subscription** for a given **Kind** of storage system.

3.1.4. Resources

The **ocs-operator** creates the following CRs in response to the spec of a given StorageSystem.

Operator Lifecycle Manager Resources

Creates a **Subscription** for the operator which defines and reconciles the given **StorageSystem's** Kind.

3.1.5. Limitation

The **odf-operator** does not provide any data storage or services itself. It exists as an integration and management layer for other storage systems.

3.1.6. High availability

High availability is not a primary requirement for the **odf-operator** Pod similar to most of the other operators. In general, there are no operations that require or benefit from process distribution. OpenShift Container Platform quickly spins up a replacement Pod whenever the current Pod becomes unavailable or is deleted.

3.1.7. Relevant config files

The **odf-operator** comes with a **ConfigMap** of variables that can be used to modify the behavior of the operator.

3.1.8. Relevant log files

To get an understanding of the OpenShift Data Foundation and troubleshoot issues, you can look at the following:

- Operator Pod logs
- **StorageSystem** status
- Underlying storage system CRD statuses

Operator Pod logs

Each operator provides standard Pod logs that include information about reconciliation and errors encountered. These logs often have information about successful reconciliation which can be filtered out and ignored.

StorageSystem status and events

The **StorageSystem** CR stores the reconciliation details in the status of the CR and has associated events. The spec of the **StorageSystem** contains the name, namespace, and Kind of the actual storage system's CRD, which the administrator can use to find further information on the status of the storage system.

3.1.9. Lifecycle

The **odf-operator** is required to be present as long as the OpenShift Data Foundation bundle remains installed. This is managed as part of OLM's reconciliation of the OpenShift Data Foundation CSV. At least one instance of the pod should be in **Ready** state.

The operator operands such as CRDs should not affect the lifecycle of the operator. The creation and deletion of **StorageSystems** is an operation outside the operator's control and must be initiated by the administrator or automated with the appropriate application programming interface (API) calls.

3.2. OPENSHIFT CONTAINER STORAGE OPERATOR

The **ocs-operator** can be described as a "meta" operator for OpenShift Data Foundation, that is, an operator meant to influence other operators and serves as a configuration gateway for the features provided by the other operators. It does not directly manage the other operators.

The **ocs-operator** has the following primary functions:

- Creates Custom Resources (CRs) that trigger the other operators to reconcile against them.
- Abstracts the Ceph and Multicloud Object Gateway configurations and limits them to known best practices that are validated and supported by Red Hat.
- Creates and reconciles the resources required to deploy containerized Ceph and NooBaa according to the support policies.

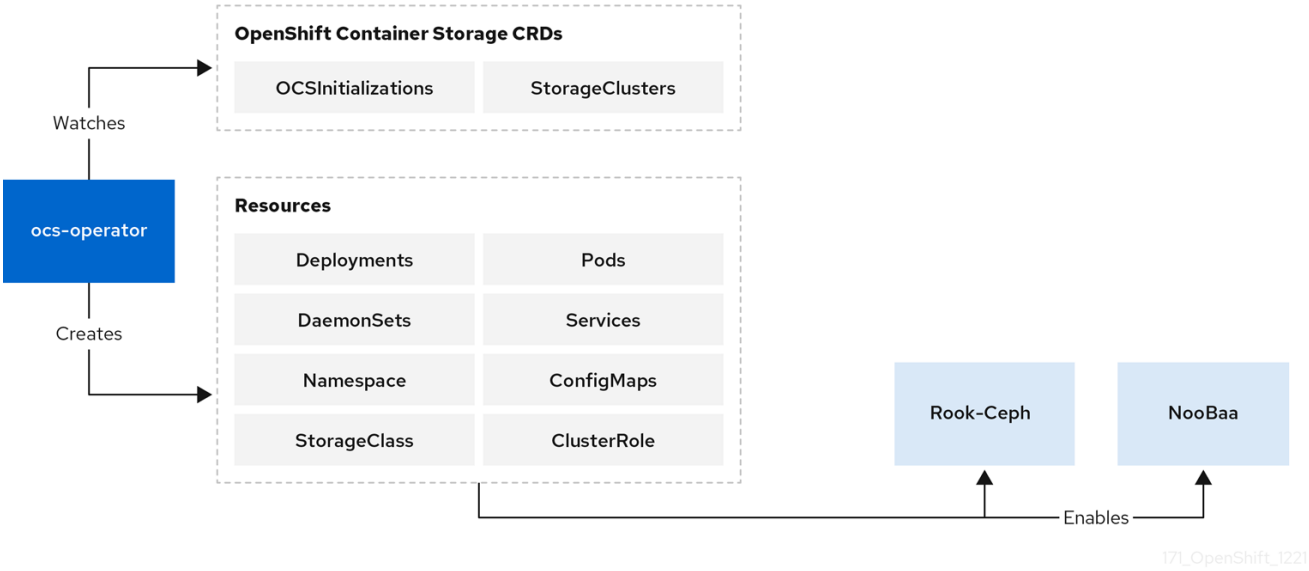
3.2.1. Components

The **ocs-operator** does not have any dependent components. However, the operator has a dependency on the existence of all the custom resource definitions (CRDs) from other operators, which are defined in the **ClusterServiceVersion** (CSV).

3.2.2. Design diagram

This diagram illustrates how OpenShift Container Storage is integrated with the OpenShift Container Platform.

Figure 3.2. OpenShift Container Storage Operator



3.2.3. Responsibilities

The two **ocs-operator** CRDs are:

- **OCSInitialization**
- **StorageCluster**

OCSInitialization is a singleton CRD used for encapsulating operations that apply at the operator level. The operator takes care of ensuring that one instance always exists. The CR triggers the following:

- Performs initialization tasks required for OpenShift Container Storage. If needed, these tasks can be triggered to run again by deleting the **OCSInitialization** CRD.
  - Ensures that the required Security Context Constraints (SCCs) for OpenShift Container Storage are present.
- Manages the deployment of the Ceph toolbox Pod, used for performing advanced troubleshooting and recovery operations.

The **StorageCluster** CRD represents the system that provides the full functionality of OpenShift Container Storage. It triggers the operator to ensure the generation and reconciliation of **Rook-Ceph** and **NooBaa** CRDs. The **ocs-operator** algorithmically generates the **CephCluster** and **NooBaa** CRDs based on the configuration in the **StorageCluster** spec. The operator also creates additional CRs, such as **CephBlockPools**, **Routes**, and so on. These resources are required for enabling different features of OpenShift Container Storage. Currently, only one StorageCluster CR per OpenShift Container Platform cluster is supported.

3.2.4. Resources

The **ocs-operator** creates the following CRs in response to the spec of the CRDs it defines . The configuration of some of these resources can be overridden, allowing for changes to the generated spec or not creating them altogether.

General resources

Events

Creates various events when required in response to reconciliation.

Persistent Volumes (PVs)

PVs are not created directly by the operator. However, the operator keeps track of all the PVs created by the Ceph CSI drivers and ensures that the PVs have appropriate annotations for the supported features.

Quickstarts

Deploys various Quickstart CRs for the OpenShift Container Platform Console.

Rook-Ceph resources

CephBlockPool

Define the default Ceph block pools. **CephFilesysPrometheusRulesoute** for the Ceph object store.

StorageClass

Define the default Storage classes. For example, for **CephBlockPool** and **CephFilesystem**).

VolumeSnapshotClass

Define the default volume snapshot classes for the corresponding storage classes.

Multicloud Object Gateway resources

NooBaa

Define the default Multicloud Object Gateway system.

Monitoring resources

- Metrics Exporter Service
- Metrics Exporter Service Monitor
- PrometheusRules

3.2.5. Limitation

The **ocs-operator** neither deploys nor reconciles the other Pods of OpenShift Data Foundation. The **ocs-operator** CSV defines the top-level components such as operator Deployments and the Operator Lifecycle Manager (OLM) reconciles the specified component.

3.2.6. High availability

High availability is not a primary requirement for the **ocs-operator** Pod similar to most of the other operators. In general, there are no operations that require or benefit from process distribution. OpenShift Container Platform quickly spins up a replacement Pod whenever the current Pod becomes unavailable or is deleted.

3.2.7. Relevant config files

The **ocs-operator** configuration is entirely specified by the CSV and is not modifiable without a custom build of the CSV.

3.2.8. Relevant log files

To get an understanding of the OpenShift Container Storage and troubleshoot issues, you can look at the following:

- Operator Pod logs
- StorageCluster status and events
- OCSInitialization status

Operator Pod logs

Each operator provides standard Pod logs that include information about reconciliation and errors encountered. These logs often have information about successful reconciliation which can be filtered out and ignored.

StorageCluster status and events

The **StorageCluster** CR stores the reconciliation details in the status of the CR and has associated events. Status contains a section of the expected container images. It shows the container images that it expects to be present in the pods from other operators and the images that it currently detects. This helps to determine whether the OpenShift Container Storage upgrade is complete.

OCSInitialization status

This status shows whether the initialization tasks are completed successfully.

3.2.9. Lifecycle

The **ocs-operator** is required to be present as long as the OpenShift Container Storage bundle remains installed. This is managed as part of OLM's reconciliation of the OpenShift Container Storage CSV. At least one instance of the pod should be in **Ready** state.

The operator operands such as CRDs should not affect the lifecycle of the operator. An **OCSInitialization** CR should always exist. The operator creates one if it does not exist. The creation and deletion of StorageClusters is an operation outside the operator's control and must be initiated by the administrator or automated with the appropriate API calls.

3.3. ROOK-CEPH OPERATOR

Rook-Ceph operator is the Rook operator for Ceph in the OpenShift Data Foundation. Rook enables Ceph storage systems to run on the OpenShift Container Platform.

The Rook-Ceph operator is a simple container that automatically bootstraps the storage clusters and monitors the storage daemons to ensure the storage clusters are healthy.

3.3.1. Components

The Rook-Ceph operator manages a number of components as part of the OpenShift Data Foundation deployment.

Ceph-CSI Driver

The operator creates and updates the CSI driver, including a provisioner for each of the two drivers, RADOS block device (RBD) and Ceph filesystem (CephFS) and a volume plugin **daemonset** for each of the two drivers.

Ceph daemons

Mons

The monitors (mons) provide the core metadata store for Ceph.

OSDs

The object storage daemons (OSDs) store the data on underlying devices.

Mgr

The manager (mgr) collects metrics and provides other internal functions for Ceph.

RGW

The RADOS Gateway (RGW) provides the S3 endpoint to the object store.

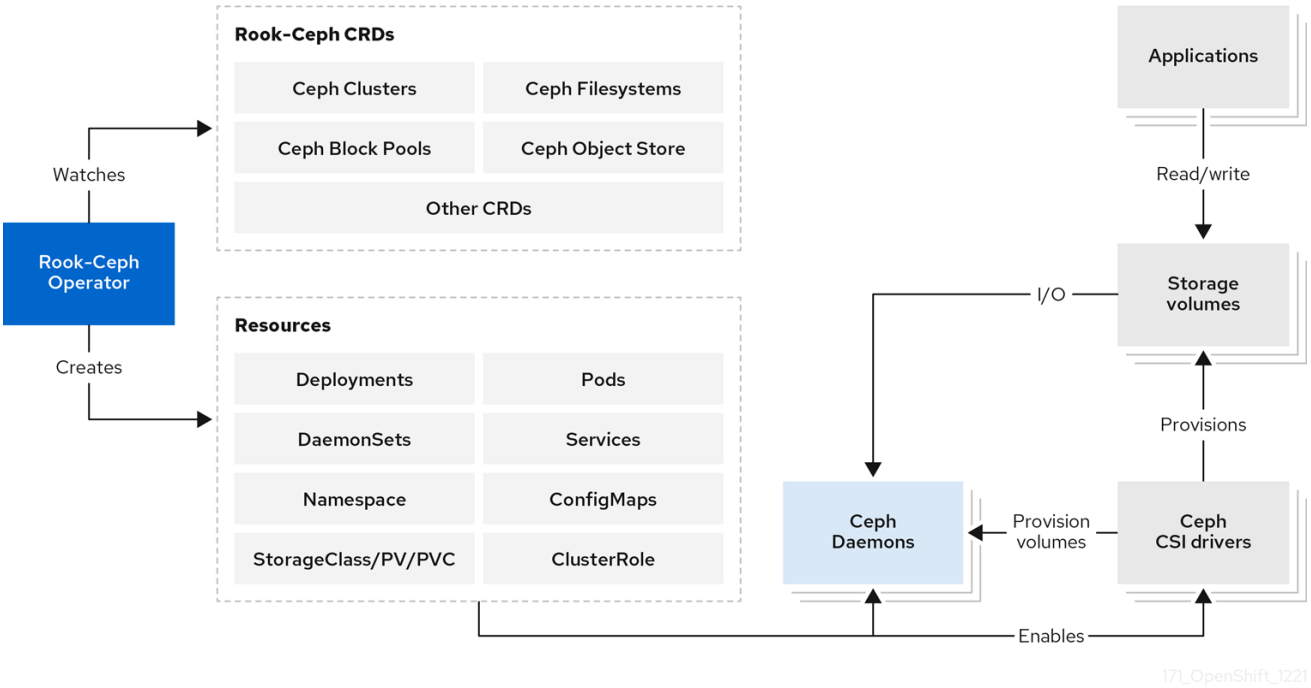
MDS

The metadata server (MDS) provides CephFS shared volumes.

3.3.2. Design diagram

The following image illustrates how Ceph Rook integrates with OpenShift Container Platform.

Figure 3.3. Rook-Ceph Operator



With Ceph running in the OpenShift Container Platform cluster, OpenShift Container Platform applications can mount block devices and filesystems managed by Rook-Ceph, or can use the S3/Swift API for object storage.

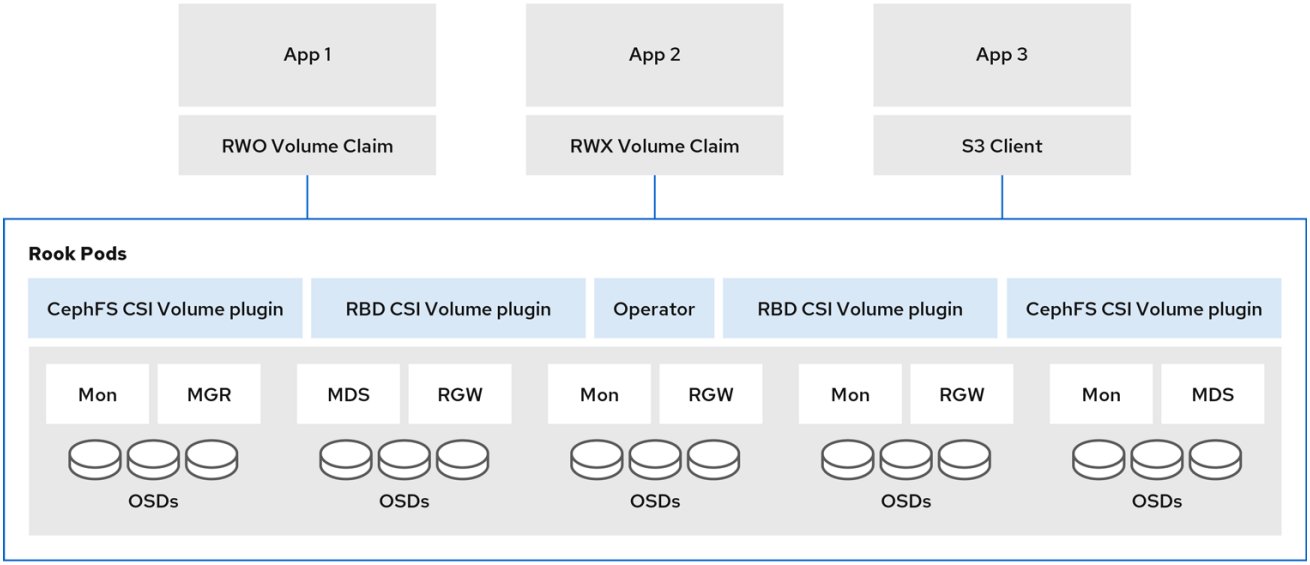
3.3.3. Responsibilities

The Rook-Ceph operator is a container that bootstraps and monitors the storage cluster. It performs the following functions:

- Automates the configuration of storage components

- Starts, monitors, and manages the Ceph monitor pods and Ceph OSD daemons to provide the RADOS storage cluster
- Initializes the pods and other artifacts to run the services to manage:
  - CRDs for pools
  - Object stores (S3/Swift)
  - Filesystems
- Monitors the Ceph mons and OSDs to ensure that the storage remains available and healthy
- Deploys and manages Ceph mons placement while adjusting the mon configuration based on cluster size
- Watches the desired state changes requested by the API service and applies the changes
- Initializes the Ceph-CSI drivers that are needed for consuming the storage
- Automatically configures the Ceph-CSI driver to mount the storage to pods

Rook-Ceph Operator architecture



171_OpenShift_1221

The Rook-Ceph operator image includes all required tools to manage the cluster. There is no change to the data path. However, the operator does not expose all Ceph configurations. Many of the Ceph features like placement groups and crush maps are hidden from the users and are provided with a better user experience in terms of physical resources, pools, volumes, filesystems, and buckets.

3.3.4. Resources

Rook-Ceph operator adds owner references to all the resources it creates in the **openshift-storage** namespace. When the cluster is uninstalled, the owner references ensure that the resources are all cleaned up. This includes OpenShift Container Platform resources such as **configmaps**, **secrets**, **services**, **deployments**, **daemonsets**, and so on.

The Rook-Ceph operator watches CRs to configure the settings determined by OpenShift Data Foundation, which includes **CephCluster**, **CephObjectStore**, **CephFilesystem**, and **CephBlockPool**.

3.3.5. Lifecycle

Rook-Ceph operator manages the lifecycle of the following pods in the Ceph cluster:

Rook operator

A single pod that owns the reconcile of the cluster.

RBD CSI Driver

- Two provisioner pods, managed by a single deployment.
- One plugin pod per node, managed by a **daemonset**.

CephFS CSI Driver

- Two provisioner pods, managed by a single deployment.
- One plugin pod per node, managed by a **daemonset**.

Monitors (mons)

Three mon pods, each with its own deployment.

Stretch clusters

Contain five mon pods, one in the arbiter zone and two in each of the other two data zones.

Manager (mgr)

There is a single mgr pod for the cluster.

Stretch clusters

There are two mgr pods (starting with OpenShift Data Foundation 4.8), one in each of the two non-arbiter zones.

Object storage daemons (OSDs)

At least three OSDs are created initially in the cluster. More OSDs are added when the cluster is expanded.

Metadata server (MDS)

The CephFS metadata server has a single pod.

RADOS gateway (RGW)

The Ceph RGW daemon has a single pod.

3.4. MCG OPERATOR

The Multicloud Object Gateway (MCG) operator is an operator for OpenShift Data Foundation along with the OpenShift Data Foundation operator and the Rook-Ceph operator. The MCG operator is available upstream as a standalone operator.

The MCG operator performs the following primary functions:

- Controls and reconciles the Multicloud Object Gateway (MCG) component within OpenShift Data Foundation.
- Manages new user resources such as object bucket claims, bucket classes, and backing stores.
- Creates the default out-of-the-box resources.

A few configurations and information are passed to the MCG operator through the OpenShift Data Foundation operator.

3.4.1. Components

The MCG operator does not have sub-components. However, it consists of a reconcile loop for the different resources that are controlled by it.

The MCG operator has a command-line interface (CLI) and is available as a part of OpenShift Data Foundation. It enables the creation, deletion, and querying of various resources. This CLI adds a layer of input sanitation and status validation before the configurations are applied unlike applying a YAML file directly.

3.4.2. Responsibilities and resources

The MCG operator reconciles and is responsible for the custom resource definitions (CRDs) and OpenShift Container Platform entities.

- Backing store
- Namespace store
- Bucket class
- Object bucket claims (OBCs)
- NooBaa, pod stateful sets CRD
- Prometheus Rules and Service Monitoring
- Horizontal pod autoscaler (HPA)

Backing store

A resource that the customer has connected to the MCG component. This resource provides MCG the ability to save the data of the provisioned buckets on top of it.

A default backing store is created as part of the deployment depending on the platform that the OpenShift Container Platform is running on. For example, when OpenShift Container Platform or OpenShift Data Foundation is deployed on Amazon Web Services (AWS), it results in a default backing store which is an AWS::S3 bucket. Similarly, for Microsoft Azure, the default backing store is a blob container and so on.

The default backing stores are created using CRDs for the cloud credential operator, which comes with OpenShift Container Platform. There is no limit on the amount of the backing stores that can be added to MCG. The backing stores are used in the bucket class CRD to define the different policies of the bucket. Refer the documentation of the specific OpenShift Data Foundation version to identify the types of services or resources supported as backing stores.

Namespace store

Resources that are used in namespace buckets. No default is created during deployment.

Bucketclass

A default or initial policy for a newly provisioned bucket. The following policies are set in a bucketclass:

Placement policy

Indicates the backing stores to be attached to the bucket and used to write the data of the bucket. This policy is used for data buckets and for cache policies to indicate the local cache placement. There are two modes of placement policy:

- Spread. Strips the data across the defined backing stores
- Mirror. Creates a full replica on each backing store

Namespace policy

A policy for the namespace buckets that defines the resources that are being used for aggregation and the resource used for the write target.

Cache Policy

This is a policy for the bucket and sets the hub (the source of truth) and the time to live (TTL) for the cache items.

A default bucket class is created during deployment and it is set with a placement policy that uses the default backing store. There is no limit to the number of bucket class that can be added.

Refer to the documentation of the specific OpenShift Data Foundation version to identify the types of policies that are supported.

Object bucket claims (OBCs)

CRDs that enable provisioning of S3 buckets. With MCG, OBCs receive an optional bucket class to note the initial configuration of the bucket. If a bucket class is not provided, the default bucket class is used.

NooBaa, pod stateful sets CRD

An internal CRD that controls the different pods of the NooBaa deployment such as the DB pod, the core pod, and the endpoints. This CRD must not be changed as it is internal. This operator reconciles the following entities:

- DB pod SCC
- Role Binding and Service Account to allow SSO single sign-on between OpenShift Container Platform and NooBaa user interfaces
- Route for S3 access
- Certificates that are taken and signed by the OpenShift Container Platform and are set on the S3 route

Prometheus rules and service monitoring

These CRDs set up scraping points for Prometheus and alert rules that are supported by MCG.

Horizontal pod autoscaler (HPA)

It is Integrated with the MCG endpoints. The endpoint pods scale up and down according to CPU pressure (amount of S3 traffic).

3.4.3. High availability

As an operator, the only high availability provided is that the OpenShift Container Platform reschedules a failed pod.

3.4.4. Relevant log files

To troubleshoot issues with the NooBaa operator, you can look at the following:

- Operator pod logs, which are also available through the must-gather.
- Different CRDs or entities and their statuses that are available through the must-gather.

3.4.5. Lifecycle

The MCG operator runs and reconciles after OpenShift Data Foundation is deployed and until it is uninstalled.

CHAPTER 4. OPENSHIFT DATA FOUNDATION INSTALLATION OVERVIEW

OpenShift Data Foundation consists of multiple components managed by multiple operators.

4.1. INSTALLED OPERATORS

When you install OpenShift Data Foundation from the Operator Hub, the following four separate Deployments are created:

- **odf-operator**: Defines the **odf-operator** Pod
- **ocs-operator**: Defines the **ocs-operator** Pod which runs processes for **ocs-operator** and its **metrics-exporter** in the same container.
- **rook-ceph-operator**: Defines the **rook-ceph-operator** Pod.
- **mcg-operator**: Defines the **mcg-operator** Pod.

These operators run independently and interact with each other by creating customer resources (CRs) watched by the other operators. The **ocs-operator** is primarily responsible for creating the CRs to configure Ceph storage and Multicloud Object Gateway. The **mcg-operator** sometimes creates Ceph volumes for use by its components.

4.2. OPENSHIFT CONTAINER STORAGE INITIALIZATION

The OpenShift Data Foundation bundle also defines an external plugin to the OpenShift Container Platform Console, adding new screens and functionality not otherwise available in the Console. This plugin runs as a web server in the **odf-console-plugin** Pod, which is managed by a Deployment created by the OLM at the time of installation.

The **ocs-operator** automatically creates an **OCSInitialization** CR after it gets created. Only one **OCSInitialization** CR exists at any point in time. It controls the **ocs-operator** behaviors that are not restricted to the scope of a single **StorageCluster**, but only performs them once. When you delete the **OCSInitialization** CR, the **ocs-operator** creates it again and this allows you to re-trigger its initialization operations.

The **OCSInitialization** CR controls the following behaviors:

SecurityContextConstraints (SCCs)

After the **OCSInitialization** CR is created, the **ocs-operator** creates various SCCs for use by the component Pods.

Ceph Toolbox Deployment

You can use the **OCSInitialization** to deploy the Ceph Toolbox Pod for the advanced Ceph operations.

Rook-Ceph Operator Configuration

This configuration creates the **rook-ceph-operator-config ConfigMap** that governs the overall configuration for **rook-ceph-operator** behavior.

4.3. STORAGE CLUSTER CREATION

The OpenShift Data Foundation operators themselves provide no storage functionality, and the desired storage configuration must be defined.

After you install the operators, create a new **StorageCluster**, using either the OpenShift Container Platform console wizard or the CLI and the **ocs-operator** reconciles this **StorageCluster**. OpenShift Data Foundation supports a single **StorageCluster** per installation. Any **StorageCluster** CRs created after the first one is ignored by **ocs-operator** reconciliation.

OpenShift Data Foundation allows the following StorageCluster configurations:

Internal

In the Internal mode, all the components run containerized within the OpenShift Container Platform cluster and uses dynamically provisioned persistent volumes (PVs) created against the **StorageClass** specified by the administrator in the installation wizard.

Internal-attached

This mode is similar to the Internal mode but the administrator is required to define the local storage devices directly attached to the cluster nodes that the Ceph uses for its backing storage. Also, the administrator need to create the CRs that the local storage operator reconciles to provide the **StorageClass**. The **ocs-operator** uses this **StorageClass** as the backing storage for Ceph.

External

In this mode, Ceph components do not run inside the OpenShift Container Platform cluster instead connectivity is provided to an external OpenShift Container Storage installation for which the applications can create PVs. The other components run within the cluster as required.

MCG Standalone

This mode facilitates the installation of a Multicloud Object Gateway system without an accompanying CephCluster.

After a **StorageCluster** CR is found, **ocs-operator** validates it and begins to create subsequent resources to define the storage components.

4.3.1. Internal mode storage cluster

Both internal and internal-attached storage clusters have the same setup process as follows:

StorageClasses	Create the storage classes that cluster applications use to create Ceph volumes.
SnapshotClasses	Create the volume snapshot classes that the cluster applications use to create snapshots of Ceph volumes.
Ceph RGW configuration	Create various Ceph object CRs to enable and provide access to the Ceph RGW object storage endpoint.
Ceph RBD Configuration	Create the **CephBlockPool** CR to enable RBD storage.
CephFS Configuration	Create the **CephFilesystem** CR to enable CephFS storage.

Rook-Ceph Configuration	Create the **rook-config-override ConfigMap** that governs the overall behavior of the underlying Ceph cluster.
CephCluster	Create the **CephCluster** CR to trigger Ceph reconciliation from **rook-ceph-operator**. For more information, see Rook-Ceph operator.
NoobaaSystem	Create the **NooBaa** CR to trigger reconciliation from **mcg-operator**. For more information, see MCG operator.
Job templates	Create **OpenShift Template** CRs that define Jobs to run administrative operations for OpenShift Container Storage.
Quickstarts	Create the **QuickStart** CRs that display the quickstart guides in the Web Console.

4.3.1.1. Cluster Creation

After the **ocs-operator** creates the **CephCluster** CR, the **rook-operator** creates the Ceph cluster according to the desired configuration. The **rook-operator** configures the following components:

Ceph **mon** daemons	Three Ceph **mon** daemons are started on different nodes in the cluster. They manage the core metadata for the Ceph cluster and they must form a majority quorum. The metadata for each **mon** is backed either by a PV if it is in a cloud environment or a path on the local host if it is in a local storage device environment.
Ceph **mgr** daemon	This daemon is started and it gathers metrics for the cluster and report them to Prometheus.
Ceph OSDs	These OSDs are created according to the configuration of the **storageClassDeviceSets**. Each OSD consumes a PV that stores the user data. By default, Ceph maintains three replicas of the application data across different OSDs for high durability and availability using the **CRUSH** algorithm.
CSI provisioners	These provisioners are started for RBD and **CephFS**. When volumes are requested for the storage classes of OpenShift Container Storage, the requests are directed to the **Ceph-CSI** driver to provision the volumes in Ceph.
CSI volume plugins and **CephFS**	The CSI volume plugins for RBD and **CephFS** are started on each node in the cluster. The volume plugin needs to be running wherever the Ceph volumes are required to be mounted by the applications.

After the **CephCluster** CR is configured, Rook reconciles the remaining Ceph CRs to complete the setup:

CephBlockPool	The **CephBlockPool** CR provides the configuration for Rook operator to create Ceph pools for RWO volumes.

CephFilesystem	The **CephFilesystem** CR instructs the Rook operator to configure a shared file system with CephFS, typically for RWX volumes. The CephFS metadata server (MDS) is started to manage the shared volumes.
CephObjectStore	The **CephObjectStore** CR instructs the Rook operator to configure an object store with the RGW service
CephObjectStoreUser CR	The **CephObjectStoreUser** CR instructs the Rook operator to configure an object store user for NooBaa to consume, publishing **access**/**private** key as well as the **CephObjectStore** endpoint.

The operator monitors the Ceph health to ensure that storage platform remains healthy. If a **mon** daemon goes down for too long a period (10 minutes), Rook starts a new **mon** in its place so that the full quorum can be fully restored.

When the **ocs-operator** updates the **CephCluster** CR, Rook immediately responds to the requested changes to update the cluster configuration.

4.3.1.2. NooBaa System creation

When a NooBaa system is created, the **mcg-operator** reconciles the following:

Default BackingStore

Depending on the platform that OpenShift Container Platform and OpenShift Data Foundation are deployed on, a default backing store resource is created so that buckets can use it for their placement policy. The different options are as follows:

Amazon Web Services (AWS) deployment	The **mcg-operator** uses the **CloudCredentialsOperator** (CCO) to mint credentials in order to create a new AWS::S3 bucket and creates a **BackingStore** on top of that bucket.
Microsoft Azure deployment	The **mcg-operator** uses the CCO to mint credentials in order to create a new Azure Blob and creates a **BackingStore** on top of that bucket.
Google Cloud Platform (GCP) deployment	The **mcg-operator** uses the CCO to mint credentials in order to create a new GCP bucket and will create a BackingStore on top of that bucket.
On-prem deployment	If RGW exists, the **mcg-operator** creates a new**CephUser** and a new bucket on top of RGW and create a **BackingStore** on top of that bucket.

None of the previously mentioned deployments are applicable	The **mcg-operator** creates a **pv-pool** based on the default storage class and creates a **BackingStore** on top of that bucket.

Default BucketClass

A **BucketClass** with a placement policy to the default **BackingStore** is created.

NooBaa pods

The following NooBaa pods are created and started:

Database (DB)	This is a Postgres DB holding metadata, statistics, events, and so on. However, it does not hold the actual data being stored.
Core	This is the pod that handles configuration, background processes, metadata management, statistics, and so on.
Endpoints	These pods perform the actual I/O-related work such as deduplication and compression, communicating with different services to write and read data, and so on. The endpoints are integrated with the **HorizonalPodAutoscaler** and their number increases and decreases according to the CPU usage observed on the existing endpoint pods.

Route

A Route for the NooBaa S3 interface is created for applications that uses S3.

Service

A Service for the NooBaa S3 interface is created for applications that uses S3.

4.3.2. External mode storage cluster

For external storage clusters, **ocs-operator** follows a slightly different setup process. The **ocs-operator** looks for the existence of the **rook-ceph-external-cluster-details ConfigMap**, which must be created by someone else, either the administrator or the Console. For information about how to create the **ConfigMap**, see Creating an OpenShift Data Foundation Cluster for external mode . The **ocs-operator** then creates some or all of the following resources, as specified in the **ConfigMap**:

External Ceph Configuration	A ConfigMap that specifies the endpoints of the external **mons**.
External Ceph Credentials Secret	A Secret that contains the credentials to connect to the external Ceph instance.

External Ceph StorageClasses	One or more StorageClasses to enable the creation of volumes for RBD, CephFS, and/or RGW.
Enable CephFS CSI Driver	If a **CephFS** StorageClass is specified, configure **rook-ceph-operator** to deploy the **CephFS** CSI Pods.
Ceph RGW Configuration	If an RGW StorageClass is specified, create various Ceph Object CRs to enable and provide access to the Ceph RGW object storage endpoint.

After creating the resources specified in the **ConfigMap**, the StorageCluster creation process proceeds as follows:

CephCluster	Create the **CephCluster** CR to trigger Ceph reconciliation from**rook-ceph-operator** (see subsequent sections).
SnapshotClasses	Create the **SnapshotClasses** that applications use to create snapshots of Ceph volumes.
NoobaaSystem	Create the **NooBaa** CR to trigger reconciliation from noobaa-operator (see subsequent sections).
QuickStarts	Create the **Quickstart** CRs that display the quickstart guides in the Console.

4.3.2.1. Cluster Creation

The Rook operator performs the following operations when the **CephCluster** CR is created in external mode:

- The operator validates that a connection is available to the remote Ceph cluster. The connection requires **mon** endpoints and secrets to be imported into the local cluster.
- The CSI driver is configured with the remote connection to Ceph. The RBD and **CephFS** provisioners and volume plugins are started similarly to the CSI driver when configured in internal mode, the connection to Ceph happens to be external to the OpenShift cluster.
- Periodically watch for monitor address changes and update the Ceph-CSI configuration accordingly.

4.3.2.2. NooBaa System creation

When a NooBaa system is created, the **mcg-operator** reconciles the following:

Default BackingStore

Depending on the platform that OpenShift Container Platform and OpenShift Data Foundation are deployed on, a default backing store resource is created so that buckets can use it for their placement policy. The different options are as follows:

Amazon Web Services (AWS) deployment	The **mcg-operator** uses the **CloudCredentialsOperator** (CCO) to mint credentials in order to create a new AWS::S3 bucket and creates a **BackingStore** on top of that bucket.
Microsoft Azure deployment	The **mcg-operator** uses the CCO to mint credentials in order to create a new Azure Blob and creates a **BackingStore** on top of that bucket.
Google Cloud Platform (GCP) deployment	The **mcg-operator** uses the CCO to mint credentials in order to create a new GCP bucket and will create a BackingStore on top of that bucket.
On-prem deployment	If RGW exists, the **mcg-operator** creates a new **CephUser** and a new bucket on top of RGW and create a **BackingStore** on top of that bucket.
None of the previously mentioned deployments are applicable	The **mcg-operator** creates a **pv-pool** based on the default storage class and creates a **BackingStore** on top of that bucket.

Default BucketClass

A **BucketClass** with a placement policy to the default **BackingStore** is created.

NooBaa pods

The following NooBaa pods are created and started:

Database (DB)	This is a Postgres DB holding metadata, statistics, events, and so on. However, it does not hold the actual data being stored.
Core	This is the pod that handles configuration, background processes, metadata management, statistics, and so on.
Endpoints	These pods perform the actual I/O-related work such as deduplication and compression, communicating with different services to write and read data, and so on. The endpoints are integrated with the **HorizonalPodAutoscaler** and their number increases and decreases according to the CPU usage observed on the existing endpoint pods.

Route

A Route for the NooBaa S3 interface is created for applications that uses S3.

Service

A Service for the NooBaa S3 interface is created for applications that uses S3.

4.3.3. MCG Standalone StorageCluster

In this mode, no CephCluster is created. Instead a NooBaa system CR is created using default values to take advantage of pre-existing StorageClasses in the OpenShift Container Platform. dashboards.

4.3.3.1. NooBaa System creation

When a NooBaa system is created, the **mcg-operator** reconciles the following:

Default BackingStore

Depending on the platform that OpenShift Container Platform and OpenShift Data Foundation are deployed on, a default backing store resource is created so that buckets can use it for their placement policy. The different options are as follows:

Amazon Web Services (AWS) deployment	The **mcg-operator** uses the **CloudCredentialsOperator** (CCO) to mint credentials in order to create a new AWS::S3 bucket and creates a **BackingStore** on top of that bucket.
Microsoft Azure deployment	The **mcg-operator** uses the CCO to mint credentials in order to create a new Azure Blob and creates a **BackingStore** on top of that bucket.
Google Cloud Platform (GCP) deployment	The **mcg-operator** uses the CCO to mint credentials in order to create a new GCP bucket and will create a BackingStore on top of that bucket.
On-prem deployment	If RGW exists, the **mcg-operator** creates a new **CephUser** and a new bucket on top of RGW and create a **BackingStore** on top of that bucket.
None of the previously mentioned deployments are applicable	The **mcg-operator** creates a **pv-pool** based on the default storage class and creates a **BackingStore** on top of that bucket.

Default BucketClass

A **BucketClass** with a placement policy to the default **BackingStore** is created.

NooBaa pods

The following NooBaa pods are created and started:

Database (DB)	This is a Postgres DB holding metadata, statistics, events, and so on. However, it does not hold the actual data being stored.
Core	This is the pod that handles configuration, background processes, metadata management, statistics, and so on.
Endpoints	These pods perform the actual I/O-related work such as deduplication and compression, communicating with different services to write and read data, and so on. The endpoints are integrated with the **HorizonalPodAutoscaler** and their number increases and decreases according to the CPU usage observed on the existing endpoint pods.

Route

A Route for the NooBaa S3 interface is created for applications that uses S3.

Service

A Service for the NooBaa S3 interface is created for applications that uses S3.

4.3.3.2. StorageSystem Creation

As a part of the StorageCluster creation, **odf-operator** automatically creates a corresponding **StorageSystem** CR, which exposes the StorageCluster to the OpenShift Data Foundation.

CHAPTER 5. OPENSHIFT DATA FOUNDATION UPGRADE OVERVIEW

As an operator bundle managed by the Operator Lifecycle Manager (OLM), OpenShift Data Foundation leverages its operators to perform high-level tasks of installing and upgrading the product through **ClusterServiceVersion** (CSV) CRs.

5.1. UPGRADE WORKFLOWS

OpenShift Data Foundation recognizes two types of upgrades: Z-stream release upgrades and Minor Version release upgrades. While the user interface workflows for these two upgrade paths are not quite the same, the resulting behaviors are fairly similar. The distinctions are as follows:

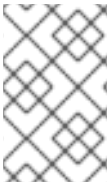
For Z-stream releases, OCS will publish a new bundle in the **redhat-operators CatalogSource**. The OLM will detect this and create an **InstallPlan** for the new CSV to replace the existing CSV. The Subscription approval strategy, whether Automatic or Manual, will determine whether the OLM proceeds with reconciliation or waits for administrator approval.

For Minor Version releases, OpenShift Container Storage will also publish a new bundle in the **redhat-operators CatalogSource**. The difference is that this bundle will be part of a new channel, and channel upgrades are not automatic. The administrator must explicitly select the new release channel. Once this is done, the OLM will detect this and create an **InstallPlan** for the new CSV to replace the existing CSV. Since the channel switch is a manual operation, OLM will automatically start the reconciliation.

From this point onwards, the upgrade processes are identical.

5.2. CLUSTERSERVICEVERSION RECONCILIATION

When the OLM detects an approved **InstallPlan**, it begins the process of reconciling the CSVs. Broadly, it does this by updating the operator resources based on the new spec, verifying the new CSV installs correctly, then deleting the old CSV. The upgrade process will push updates to the operator Deployments, which will trigger the restart of the operator Pods using the images specified in the new CSV.



NOTE

While it is possible to make changes to a given CSV and have those changes propagate to the relevant resource, when upgrading to a new CSV all custom changes will be lost, as the new CSV will be created based on its unaltered spec.

5.3. OPERATOR RECONCILIATION

At this point, the reconciliation of the OpenShift Data Foundation operands proceeds as defined in the OpenShift Data Foundation installation overview . The operators will ensure that all relevant resources exist in their expected configurations as specified in the user-facing resources (for example, **StorageCluster**).