



OpenShift Container Platform 4.20

Registry

Configuring registries for OpenShift Container Platform

OpenShift Container Platform 4.20 Registry

Configuring registries for OpenShift Container Platform

Legal Notice

Copyright © Red Hat.

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, 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 Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

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

All other trademarks are the property of their respective owners.

Abstract

This document provides instructions for configuring and managing the internal registry for OpenShift Container Platform. It also provides a general overview of registries associated with OpenShift Container Platform.

Table of Contents

CHAPTER 1. OPENSIFT IMAGE REGISTRY OVERVIEW	4
1.1. GLOSSARY OF COMMON TERMS FOR OPENSIFT IMAGE REGISTRY	4
1.2. INTEGRATED OPENSIFT IMAGE REGISTRY	5
1.3. AUTOMATICALLY PRUNING IMAGES	5
1.4. CREATING CONTAINERS BY USING IMAGES FROM THIRD-PARTY REGISTRIES	7
1.5. RED HAT QUAY REGISTRIES	8
1.6. AUTHENTICATION ENABLED RED HAT REGISTRY	8
CHAPTER 2. IMAGE REGISTRY OPERATOR IN OPENSIFT CONTAINER PLATFORM	10
2.1. IMAGE REGISTRY ON CLOUD PLATFORMS AND OPENSTACK	10
2.2. IMAGE REGISTRY ON BARE METAL, NUTANIX, AND VSPHERE	10
2.2.1. Image registry removed during installation	10
2.3. IMAGE REGISTRY OPERATOR DISTRIBUTION ACROSS AVAILABILITY ZONES	11
2.4. ADDITIONAL RESOURCES	11
2.5. IMAGE REGISTRY OPERATOR CONFIGURATION PARAMETERS	12
2.6. ENABLING THE IMAGE REGISTRY DEFAULT ROUTE BY USING A CRD	14
2.7. CONFIGURING ADDITIONAL TRUST STORES FOR IMAGE REGISTRY ACCESS	14
2.8. CONFIGURING STORAGE CREDENTIALS FOR THE IMAGE REGISTRY OPERATOR	15
2.9. ADDITIONAL RESOURCES	16
CHAPTER 3. SETTING UP AND CONFIGURING THE REGISTRY	17
3.1. CONFIGURING THE REGISTRY FOR AWS USER-PROVISIONED INFRASTRUCTURE	17
3.1.1. Configuring a secret for the Image Registry Operator	17
3.1.2. Configuring registry storage for AWS with user-provisioned infrastructure	17
3.1.3. Image Registry Operator configuration parameters for AWS S3	18
3.2. CONFIGURING THE REGISTRY FOR GOOGLE CLOUD USER-PROVISIONED INFRASTRUCTURE	19
3.2.1. Configuring a secret for the Image Registry Operator	19
3.2.2. Configuring the registry storage for Google Cloud with user-provisioned infrastructure	20
3.2.3. Image Registry Operator configuration parameters for Google Cloud GCS	21
3.3. CONFIGURING THE REGISTRY FOR OPENSTACK USER-PROVISIONED INFRASTRUCTURE	21
3.3.1. Configuring Image Registry Operator redirects	21
3.3.2. Configuring a secret for the Image Registry Operator	22
3.3.3. Registry storage for RHOSP with user-provisioned infrastructure	22
3.3.4. Image Registry Operator configuration parameters for RHOSP Swift	23
3.4. CONFIGURING THE REGISTRY FOR AZURE USER-PROVISIONED INFRASTRUCTURE	23
3.4.1. Configuring a secret for the Image Registry Operator	23
3.4.2. Configuring registry storage for Azure	24
3.4.3. Configuring registry storage for Azure Government	25
3.5. CONFIGURING THE REGISTRY FOR RHOSP	25
3.5.1. Configuring an image registry with custom storage on clusters that run on RHOSP	25
3.6. CONFIGURING THE REGISTRY FOR BARE METAL	28
3.6.1. Image registry removed during installation	28
3.6.2. Changing the image registry's management state	28
3.6.3. Image registry storage configuration	28
3.6.3.1. Configuring registry storage for bare metal and other manual installations	28
3.6.3.2. Configuring storage for the image registry in non-production clusters	30
3.6.3.3. Configuring block registry storage for bare metal	30
3.6.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation	32
3.6.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation	33
3.6.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data	

Foundation	35
3.6.5. Additional resources	36
3.7. CONFIGURING THE REGISTRY FOR VSPHERE	36
3.7.1. Image registry removed during installation	36
3.7.2. Changing the image registry's management state	36
3.7.3. Image registry storage configuration	37
3.7.3.1. Configuring registry storage for VMware vSphere	37
3.7.3.2. Configuring storage for the image registry in non-production clusters	39
3.7.3.3. Configuring block registry storage for VMware vSphere	39
3.7.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation	41
3.7.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation	42
3.7.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation	44
3.7.5. Additional resources	45
3.8. CONFIGURING THE REGISTRY FOR RED HAT OPENSIFT DATA FOUNDATION	45
3.8.1. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation	45
3.8.2. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation	47
3.8.3. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation	48
3.8.4. Additional resources	49
3.9. CONFIGURING THE REGISTRY FOR NUTANIX	49
3.9.1. Image registry removed during installation	49
3.9.2. Changing the image registry's management state	49
3.9.3. Image registry storage configuration	50
3.9.3.1. Configuring registry storage for Nutanix	50
3.9.3.2. Configuring storage for the image registry in non-production clusters	51
3.9.3.3. Configuring block registry storage for Nutanix volumes	52
3.9.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation	53
3.9.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation	55
3.9.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation	56
3.9.5. Additional resources	57
CHAPTER 4. ACCESSING THE REGISTRY	59
4.1. PREREQUISITES	59
4.2. ACCESSING THE REGISTRY DIRECTLY FROM THE CLUSTER	59
4.3. CHECKING THE STATUS OF THE REGISTRY PODS	61
4.4. VIEWING REGISTRY LOGS	61
4.5. ACCESSING REGISTRY METRICS	61
4.6. ADDITIONAL RESOURCES	63
CHAPTER 5. EXPOSING THE REGISTRY	64
5.1. EXPOSING A DEFAULT REGISTRY MANUALLY	64
5.2. EXPOSING A SECURE REGISTRY MANUALLY	65

CHAPTER 1. OPENSIFT IMAGE REGISTRY OVERVIEW

OpenShift Container Platform can build images from your source code, deploy them, and manage their lifecycle. OpenShift Container Platform provides an internal, integrated container image registry that can be deployed in your OpenShift Container Platform environment to locally manage images. The overview section includes OpenShift image registry reference information and links for registries commonly used with OpenShift Container Platform.

1.1. GLOSSARY OF COMMON TERMS FOR OPENSIFT IMAGE REGISTRY

This glossary defines the common terms that are used in the registry content.

container

Lightweight and executable images that consist of software and all its dependencies. Because containers virtualize the operating system, you can run containers in a data center, a public or private cloud, or your local host.

image repository

An image repository is a collection of related container images and tags identifying images.

mirror registry

The mirror registry is a registry that holds the mirror of OpenShift Container Platform images.

namespace

A namespace isolates groups of resources within a single cluster.

pod

The pod is the smallest logical unit in Kubernetes. A pod consists of one or more containers to run in a compute node.

private registry

A registry is a server that implements the container image registry API. A private registry is a registry that requires authentication so that users can access the content of the registry.

public registry

A registry is a server that implements the container image registry API. A public registry is a registry that serves its content publicly.

Quay.io

A public Red Hat Quay Container Registry instance provided and maintained by Red Hat, which serves most of the container images and Operators to OpenShift Container Platform clusters.

OpenShift image registry

OpenShift image registry is the registry provided by OpenShift Container Platform to manage images.

registry authentication

To push and pull images to and from private image repositories, the registry needs to authenticate its users with credentials.

route

Exposes a service to allow for network access to pods from users and applications outside the OpenShift Container Platform instance.

scale down

To decrease the number of replicas.

scale up

To increase the number of replicas.

service

A service exposes a running application on a set of pods.

1.2. INTEGRATED OPENSIFT IMAGE REGISTRY

OpenShift Container Platform provides a built-in container image registry that runs as a standard workload on the cluster. The registry is configured and managed by an infrastructure Operator. The registry provides an out-of-the-box solution that runs on top of existing cluster infrastructure, so that you can manage the images that run workloads.

You can scale the registry up or down like any other cluster workload and the registry does not require specific infrastructure provisioning. The registry also integrates into the cluster user authentication and authorization system, which means that defining user permissions on image resources controls access to create and retrieve images.

The registry is typically used as a publication target for images built on the cluster and as a source of images for workloads running on the cluster. When you push a new image to the registry, a notification gets sent to the cluster of the new image so that other components can react to and consume the updated image.

Image data is stored in two locations. The actual image data is stored in a configurable storage location, such as cloud storage or a filesystem volume. The image metadata, which is exposed by the standard cluster APIs and is used to perform access control, is stored as standard API resources, specifically images and image streams.

Additional resources

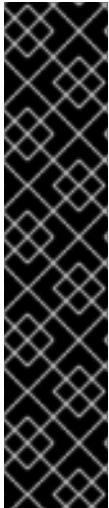
- [Image Registry Operator in OpenShift Container Platform](#)

1.3. AUTOMATICALLY PRUNING IMAGES

Images from the OpenShift image registry that are no longer required by the system because of age, status, or exceed limits being automatically pruned. As a cluster administrator, can configure or suspend the pruning Custom Resource (CR).

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with cluster administrator permissions.
- Install the **oc** CLI.



IMPORTANT

The behavior of the Image Registry Operator for managing the pruner is independent to the **managementState** specified on the **ClusterOperator** object of the Image Registry Operator. If the Image Registry Operator is not in the **Managed** state, the image pruner can still be configured and managed by the Pruning Custom Resource.

However, the **managementState** of the Image Registry Operator alters the behavior of the deployed image pruner job:

- **Managed:** the **--prune-registry** flag for the image pruner is set to **true**.
- **Removed:** the **--prune-registry** flag for the image pruner is set to **false**, meaning it only prunes image metadata in etcd.

Procedure

- Verify that the object named **imagepruners.imageregistry.operator.openshift.io/cluster** contains the following **spec** and **status** fields:

```
spec:
  schedule: 0 0 * * *
  suspend: false
  keepTagRevisions: 3
  keepYoungerThanDuration: 60m
  keepYoungerThan: 3600000000000
  resources: {}
  affinity: {}
  nodeSelector: {}
  tolerations: []
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 3
status:
  observedGeneration: 2
  conditions:
    - type: Available
      status: "True"
      lastTransitionTime: 2019-10-09T03:13:45
      reason: Ready
      message: "Periodic image pruner has been created."
    - type: Scheduled
      status: "True"
      lastTransitionTime: 2019-10-09T03:13:45
      reason: Scheduled
      message: "Image pruner job has been scheduled."
    - type: Failed
      status: "False"
      lastTransitionTime: 2019-10-09T03:13:45
      reason: Succeeded
      message: "Most recent image pruning job succeeded."
```

- **schedule:** **CronJob** formatted schedule. This is an optional field, default is daily at midnight.
- **suspend:** If set to **true**, the **CronJob** running pruning is suspended. This is an optional field, default is **false**. The initial value on new clusters is **false**.

- **keepTagRevisions**: The number of revisions per tag to keep. This is an optional field, default is **3**. The initial value is **3**.
- **keepYoungerThanDuration**: Retain images younger than this duration. This is an optional field. If a value is not specified, either **keepYoungerThan** or the default value **60m** (60 minutes) is used.
- **keepYoungerThan**: Deprecated. The same as **keepYoungerThanDuration**, but the duration is specified as an integer in nanoseconds. This is an optional field. When **keepYoungerThanDuration** is set, this field is ignored.
- **resources**: Standard pod resource requests and limits. This is an optional field.
- **affinity**: Standard pod affinity. This is an optional field.
- **nodeSelector**: Standard pod node selector. This is an optional field.
- **tolerations**: Standard pod tolerations. This is an optional field.
- **successfulJobsHistoryLimit**: The maximum number of successful jobs to retain. Must be greater than or equal to **1** to ensure metrics are reported. This is an optional field, default is **3**. The initial value is **3**.
- **failedJobsHistoryLimit**: The maximum number of failed jobs to retain. Must be greater than or equal **1** to ensure metrics are reported. This is an optional field, default is **3**. The initial value is **3**.
- **observedGeneration**: The generation observed by the Operator.
- **conditions**: The standard condition objects with the following types:
 - **Available**: Indicates if the pruning job has been created. Reasons can be **Ready** or **Error**.
 - **Scheduled**: Indicates if the next pruning job has been scheduled. Reasons can be **Scheduled**, **Suspended**, or **Error**.
 - **Failed**: Indicates if the most recent pruning job failed.

1.4. CREATING CONTAINERS BY USING IMAGES FROM THIRD-PARTY REGISTRIES

Some container image registries require access authorization. Podman is an open source tool for managing containers and container images and interacting with image registries. You can use Podman to authenticate your credentials, pull the registry image, and store local images in a local file system. The procedure provides a generic example of authenticating the registry with Podman.

OpenShift Container Platform can communicate with registries to access private image repositories by using credentials supplied by the user. This allows OpenShift Container Platform to push and pull images to and from private repositories.

OpenShift Container Platform can create containers by using images from third-party registries, but these registries unlikely offer the same image notification support as the integrated OpenShift image registry. In this situation, OpenShift Container Platform fetches tags from the remote registry upon image stream creation. To refresh the fetched tags, run **oc import-image <stream>**. When new images are detected, the previously described build and deployment reactions occur.

Procedure

1. Use the [Red Hat Ecosystem Catalog](#) to search for specific container images from the Red Hat Repository and select the required image.
2. Click **Get this image** to find the command for your container image.
3. Log in by running the following command and entering your username and password to authenticate. Example output is shown for demonstrative purposes.

```
$ podman login registry.redhat.io
```

```
Username:<your_registry_account_username>  
Password:<your_registry_account_password>
```

4. Download the image and save it locally by running the following command:

```
$ podman pull registry.redhat.io/<repository_name>
```

1.5. RED HAT QUAY REGISTRIES

If you need an enterprise-quality container image registry, Red Hat Quay is available both as a hosted service and as software that you can install in your own data center or cloud environment. Advanced features in Red Hat Quay include geo-replication, image scanning, and the ability to roll back images.

Visit the [Quay.io](#) site to set up your own hosted Quay registry account. After that, follow the Quay Tutorial to log in to the Quay registry and start managing your images.

You can access your Red Hat Quay registry from OpenShift Container Platform, similar to any remote container image registry.

Additional resources

- [Red Hat Quay product documentation](#)

1.6. AUTHENTICATION ENABLED RED HAT REGISTRY

All container images available through the Container images section of the Red Hat Ecosystem Catalog are hosted on an image registry, **registry.redhat.io**. The registry, **registry.redhat.io**, requires authentication for access to images and hosted content on OpenShift Container Platform.



NOTE

OpenShift Container Platform pulls images from **registry.redhat.io**, so you must configure your cluster to use it.

The new registry uses standard OAuth mechanisms for authentication, with the following methods:

- **Authentication token.** Tokens, which are generated by administrators, are service accounts that give systems the ability to authenticate against the container image registry. Service accounts are not affected by changes in user accounts, so the token authentication method is reliable and resilient. This is the only supported authentication option for production clusters.

- **Web username and password.** This is the standard set of credentials you use to log in to resources such as **access.redhat.com**. While you could use this authentication method with OpenShift Container Platform, Red Hat does not support the method for production deployments. Restrict this authentication method to standalone projects outside OpenShift Container Platform.

You can use **podman login** with your credentials, either username and password or authentication token, to access content on the new registry.

All image streams point to the new registry, which uses the installation pull secret to authenticate.

You must place your credentials in either of the following places:

- **openshift namespace.** Your credentials must exist in the **openshift** namespace so that the image streams in the **openshift** namespace can import.
- **Your host.** Your credentials must exist on your host because Kubernetes uses the credentials from your host when it goes to pull images.

Additional resources

- [Registry service accounts](#)

CHAPTER 2. IMAGE REGISTRY OPERATOR IN OPENSIFT CONTAINER PLATFORM

2.1. IMAGE REGISTRY ON CLOUD PLATFORMS AND OPENSTACK

The Image Registry Operator installs a single instance of the OpenShift image registry and manages all registry configuration, including setting up registry storage.



NOTE

Storage is only automatically configured when you install an installer-provisioned infrastructure cluster on AWS, Azure, Google Cloud, IBM®, or RHOSP.

When you install or upgrade an installer-provisioned infrastructure cluster on AWS, Azure, Google Cloud, IBM®, or RHOSP, the Image Registry Operator sets the **spec.storage.managementState** parameter to **Managed**. If the **spec.storage.managementState** parameter is set to **Unmanaged**, the Image Registry Operator takes no action related to storage.

After the control plane deploys in the management cluster, the Operator creates a default **configs.imageregistry.operator.openshift.io** custom resource (CR) instance based on configuration detected in the cluster.

If insufficient information is available to define a complete **configs.imageregistry.operator.openshift.io** CR, the incomplete resource is defined and the Operator updates the resource status with information about what is missing.



IMPORTANT

The Image Registry Operator's behavior for managing the pruner is orthogonal to the **managementState** specified on the **ClusterOperator** object for the Image Registry Operator. If the Image Registry Operator is not in the **Managed** state, the image pruner can still be configured and managed by the **Pruning** custom resource.

However, the **managementState** of the Image Registry Operator alters the behavior of the deployed image pruner job:

- **Managed:** the **--prune-registry** flag for the image pruner is set to **true**.
- **Removed:** the **--prune-registry** flag for the image pruner is set to **false**, meaning the image pruner job only prunes image metadata in etcd.

2.2. IMAGE REGISTRY ON BARE METAL, NUTANIX, AND VSPHERE

2.2.1. Image registry removed during installation

On platforms that do not provide shareable object storage, the OpenShift Image Registry Operator bootstraps itself as **Removed**. This allows **openshift-installer** to complete installations on these platform types.

After installation, you must edit the Image Registry Operator configuration to switch the **managementState** from **Removed** to **Managed**. When this has completed, you must configure storage.

2.3. IMAGE REGISTRY OPERATOR DISTRIBUTION ACROSS AVAILABILITY ZONES

The default configuration of the Image Registry Operator spreads image registry pods across topology zones to prevent delayed recovery times in case of a complete zone failure where all pods are impacted. Reference the following YAML to understand the default parameter values that the Image Registry Operator uses when the Operator deploys with a zone-related topology constraint:

```
topologySpreadConstraints:
- labelSelector:
  matchLabels:
    docker-registry: default
  maxSkew: 1
  topologyKey: kubernetes.io/hostname
  whenUnsatisfiable: DoNotSchedule
- labelSelector:
  matchLabels:
    docker-registry: default
  maxSkew: 1
  topologyKey: node-role.kubernetes.io/worker
  whenUnsatisfiable: DoNotSchedule
- labelSelector:
  matchLabels:
    docker-registry: default
  maxSkew: 1
  topologyKey: topology.kubernetes.io/zone
  whenUnsatisfiable: DoNotSchedule
```

Reference the following YAML to understand the default parameter value that the Image Registry Operator uses when the Operator deploys with a zone-related topology constraint, which applies to bare metal and vSphere instances:

```
topologySpreadConstraints:
- labelSelector:
  matchLabels:
    docker-registry: default
  maxSkew: 1
  topologyKey: kubernetes.io/hostname
  whenUnsatisfiable: DoNotSchedule
- labelSelector:
  matchLabels:
    docker-registry: default
  maxSkew: 1
  topologyKey: node-role.kubernetes.io/worker
  whenUnsatisfiable: DoNotSchedule
```

As a cluster administrator, you can override the default **topologySpreadConstraints** section values by configuring the **configs.imageregistry.operator.openshift.io/cluster** spec file.

2.4. ADDITIONAL RESOURCES

- [Configuring pod topology spread constraints](#)

2.5. IMAGE REGISTRY OPERATOR CONFIGURATION PARAMETERS

You can configure the Image Registry Operator using the **configs.imageregistry.operator.openshift.io** resource. The resource provides parameters for managing registry state, storage, logging, routing, and deployment settings.

Parameter	Description
managementState	Managed: The Operator updates the registry as configuration resources are updated. Unmanaged: The Operator ignores changes to the configuration resources. Removed: The Operator removes the registry instance and tear down any storage that the Operator provisioned.
logLevel	Sets logLevel of the registry instance. Defaults to Normal. The following values for logLevel are supported: • Normal • Debug • Trace • TraceAll
httpSecret	Value needed by the registry to secure uploads, generated by default.
operatorLogLevel	The operatorLogLevel configuration parameter provides intent-based logging for the Operator itself and a simple way to manage coarse-grained logging choices that Operators must interpret for themselves. This configuration parameter defaults to Normal. It does not provide fine-grained control. The following values for operatorLogLevel are supported: • Normal • Debug • Trace • TraceAll
proxy	Defines the Proxy to be used when calling master API and upstream registries.

Parameter	Description
affinity	You can use the affinity parameter to configure pod scheduling preferences and constraints for Image Registry Operator pods. Affinity settings can use the podAffinity or podAntiAffinity spec. Both options can use either preferredDuringSchedulingIgnoredDuringExecution rules or requiredDuringSchedulingIgnoredDuringExecution rules.
storage	StorageType : Details for configuring registry storage, for example S3 bucket coordinates. Normally configured by default.
readOnly	Indicates whether the registry instance should reject attempts to push new images or delete existing ones.
requests	API Request Limit details. Controls how many parallel requests a given registry instance will handle before queuing additional requests.
defaultRoute	Determines whether or not an external route is defined using the default hostname. If enabled, the route uses re-encrypt encryption. Defaults to false .
routes	Array of additional routes to create. You provide the hostname and certificate for the route.
rolloutStrategy	Defines rollout strategy for the image registry deployment. Defaults to RollingUpdate .
replicas	Replica count for the registry.
disableRedirect	Controls whether to route all data through the registry, rather than redirecting to the back end. Defaults to false .

Parameter	Description
spec.storage.managementState	The Image Registry Operator sets the spec.storage.managementState parameter to Managed on new installations or upgrades of clusters using installer-provisioned infrastructure on AWS or Azure. ● Managed: Determines that the Image Registry Operator manages underlying storage. If the Image Registry Operator's managementState is set to Removed, then the storage is deleted. ○ If the managementState is set to Managed, the Image Registry Operator attempts to apply some default configuration on the underlying storage unit. For example, if set to Managed, the Operator tries to enable encryption on the S3 bucket before making it available to the registry. If you do not want the default settings to be applied on the storage you are providing, make sure the managementState is set to Unmanaged. ● Unmanaged: Determines that the Image Registry Operator ignores the storage settings. If the Image Registry Operator's managementState is set to Removed, then the storage is not deleted. If you provided an underlying storage unit configuration, such as a bucket or container name, and the spec.storage.managementState is not yet set to any value, then the Image Registry Operator configures it to Unmanaged.

2.6. ENABLING THE IMAGE REGISTRY DEFAULT ROUTE BY USING A CRD

In OpenShift Container Platform, the **Registry** Operator controls the OpenShift image registry feature and you define this Operator in the **configs.imageregistry.operator.openshift.io** Custom Resource Definition (CRD). If you need to automatically enable the Image Registry default route, patch the Image Registry Operator CRD.

Procedure

- Patch the Image Registry Operator CRD:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --type merge -p '{"spec": {"defaultRoute":true}}'
```

2.7. CONFIGURING ADDITIONAL TRUST STORES FOR IMAGE REGISTRY ACCESS

You can add references to a config map that has additional certificate authorities (CAs) to be trusted during image registry access to the **image.config.openshift.io/cluster** custom resource (CR).

Prerequisites

- The certificate authorities (CAs) must be PEM-encoded.

Procedure

1. Create a config map in the **openshift-config** namespace, then and use the config map name in the **AdditionalTrustedCA** parameter of the **image.config.openshift.io** CR. This adds CAs that should be trusted when the cluster contacts external image registries.

Image registry CA config map example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

where:

data:registry.example.com:

An example hostname of a registry for which this CA is to be trusted.

data:registry-with-port.example.com..5000:

An example hostname of a registry with the port for which this CA is to be trusted. If the registry has a port, such as **registry-with-port.example.com:5000**, **:** must be replaced with **...** The PEM certificate content is the value for each additional registry CA to trust.

2. Optional. Configure an additional CA by running the following command:

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n
openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

2.8. CONFIGURING STORAGE CREDENTIALS FOR THE IMAGE REGISTRY OPERATOR

In addition to the **configs.imageregistry.operator.openshift.io** Custom Resource (CR) and ConfigMap resources, storage credential configuration is provided to the Operator by a separate secret resource. This resource is located within the **openshift-image-registry** namespace.

You can create an **image-registry-private-configuration-user** secret that in turn creates custom credentials needed for storage access and management. If default credentials exist, the custom credentials override the default credentials used by the Operator.

Procedure

- Create an OpenShift Container Platform secret that contains the required keys.

```
$ oc create secret generic image-registry-private-configuration-user --from-literal=KEY1=value1 --from-literal=KEY2=value2 --namespace openshift-image-registry
```

2.9. ADDITIONAL RESOURCES

- [Configuring the registry for AWS user-provisioned infrastructure](#)
- [Configuring the registry for Google Cloud user-provisioned infrastructure](#)
- [Configuring the registry for Azure user-provisioned infrastructure](#)
- [Configuring the registry for bare metal](#)
- [Configuring the registry for vSphere](#)
- [Configuring the registry for RHOSP](#)
- [Configuring the registry for Red Hat OpenShift Data Foundation](#)
- [Configuring the registry for Nutanix](#)

CHAPTER 3. SETTING UP AND CONFIGURING THE REGISTRY

3.1. CONFIGURING THE REGISTRY FOR AWS USER-PROVISIONED INFRASTRUCTURE

Save your container images to a durable storage location by configuring the built-in image registry to use dedicated AWS storage. This setup provides persistent scalable storage for your registry, separate from ephemeral cluster storage.

3.1.1. Configuring a secret for the Image Registry Operator

In addition to the **configs.imageregistry.operator.openshift.io** and ConfigMap resources, configuration is provided to the Operator by a separate secret resource located within the **openshift-image-registry** namespace.

The **image-registry-private-configuration-user** secret provides credentials needed for storage access and management. It overrides the default credentials used by the Operator, if default credentials were found.

For S3 on AWS storage, the secret is expected to contain two keys:

- **REGISTRY_STORAGE_S3_ACCESSKEY**
- **REGISTRY_STORAGE_S3_SECRETKEY**

Procedure

- Create an OpenShift Container Platform secret that contains the required keys.

```
$ oc create secret generic image-registry-private-configuration-user --from-  
literal=REGISTRY_STORAGE_S3_ACCESSKEY=myaccesskey --from-  
literal=REGISTRY_STORAGE_S3_SECRETKEY=mysecretkey --namespace openshift-  
image-registry
```

3.1.2. Configuring registry storage for AWS with user-provisioned infrastructure

During installation, your cloud credentials are sufficient to create an Amazon S3 bucket and the Registry Operator will automatically configure storage.

If the Registry Operator cannot create an S3 bucket and automatically configure storage, you can create an S3 bucket and configure storage with the following procedure.



WARNING

To secure your registry images in AWS, [block public access](#) to the S3 bucket.

Prerequisites

- You have a cluster on AWS with user-provisioned infrastructure.
- For Amazon S3 storage, the secret is expected to contain two keys:
 - **REGISTRY_STORAGE_S3_ACCESSKEY**
 - **REGISTRY_STORAGE_S3_SECRETKEY**

Procedure

1. Set up a [Bucket Lifecycle Policy](#) to abort incomplete multipart uploads that are one day old.
2. Fill in the storage configuration in **configs.imageregistry.operator.openshift.io/cluster**:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

Example configuration

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  storage:
    s3:
      bucket: <bucket_name>
      region: <region_name>
```

3.1.3. Image Registry Operator configuration parameters for AWS S3

The following configuration parameters are available for AWS S3 registry storage.

The image registry **spec.storage.s3** configuration parameter holds the information to configure the registry to use the AWS S3 service for back-end storage. See the [S3 storage driver documentation](#) for more information.

Parameter	Description
bucket	Bucket is the bucket name in which you want to store the registry's data. It is optional and is generated if not provided.
chunkSizeMiB	ChunkSizeMiB is the size of the multipart upload chunks of the S3 API. The default is 10 MiB with a minimum of 5 MiB.
region	Region is the AWS region in which your bucket exists. It is optional and is set based on the installed AWS Region.
regionEndpoint	RegionEndpoint is the endpoint for S3 compatible storage services. It is optional and defaults based on the Region that is provided.

Parameter	Description
virtualHostedStyle	VirtualHostedStyle enables using S3 virtual hosted style bucket paths with a custom RegionEndpoint. It is optional and defaults to false. Set this parameter to deploy OpenShift Container Platform to hidden regions.
encrypt	Encrypt specifies whether or not the registry stores the image in encrypted format. It is optional and defaults to false.
keyID	KeyID is the KMS key ID to use for encryption. It is optional. Encrypt must be true, or this parameter is ignored.
cloudFront	CloudFront configures Amazon Cloudfront as the storage middleware in a registry. It is optional.
trustedCA	The namespace for the config map referenced by trustedCA is openshift-config . The key for the bundle in the config map is ca-bundle.crt . It is optional.

**NOTE**

When the value of the **regionEndpoint** parameter is configured to a URL of a Rados Gateway, an explicit port must not be specified. For example:

```
regionEndpoint: http://rook-ceph-rgw-ocs-storagecluster-cephobjectstore.openshift-storage.svc.cluster.local
```

3.2. CONFIGURING THE REGISTRY FOR GOOGLE CLOUD USER-PROVISIONED INFRASTRUCTURE

Save your container images to a durable storage location by configuring the built-in image registry to use dedicated Google Cloud storage. This setup provides persistent scalable storage for your registry, separate from ephemeral cluster storage.

3.2.1. Configuring a secret for the Image Registry Operator

In addition to the **configs.imageregistry.operator.openshift.io** and ConfigMap resources, configuration is provided to the Operator by a separate secret resource located within the **openshift-image-registry** namespace.

The **image-registry-private-configuration-user** secret provides credentials needed for storage access and management. It overrides the default credentials used by the Operator, if default credentials were found.

For GCS on Google Cloud storage, the secret is expected to contain one key whose value is the contents of a credentials file provided by Google Cloud:

- **REGISTRY_STORAGE_GCS_KEYFILE**

Procedure

- Create an OpenShift Container Platform secret that contains the required keys.

```
$ oc create secret generic image-registry-private-configuration-user --from-
file=REGISTRY_STORAGE_GCS_KEYFILE=<path_to_keyfile> --namespace openshift-
image-registry
```

3.2.2. Configuring the registry storage for Google Cloud with user-provisioned infrastructure

If the Registry Operator cannot create a Google Cloud bucket, you must set up the storage medium manually and configure the settings in the registry custom resource (CR).

Prerequisites

- A cluster on Google Cloud with user-provisioned infrastructure.
- To configure registry storage for Google Cloud, you need to provide Registry Operator cloud credentials.
- For GCS on Google Cloud storage, the secret is expected to contain one key whose value is the contents of a credentials file provided by Google Cloud:
 - **REGISTRY_STORAGE_GCS_KEYFILE**



WARNING

You can secure your registry images that use a Google Cloud Storage bucket by setting [public access prevention](#).

Procedure

1. Set up an [Object Lifecycle Management policy](#) to abort incomplete multipart uploads that are one day old.
2. Fill in the storage configuration in **configs.imageregistry.operator.openshift.io/cluster**:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

Example configuration

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  storage:
    gcs:
```

bucket: <bucket_name>
 projectID: <project_id>
 region: <region_name>

3.2.3. Image Registry Operator configuration parameters for Google Cloud GCS

The following parameters configure are available to configure your Google Cloud GCS registry storage.

Parameter	Description
bucket	Bucket is the bucket name in which you want to store the registry's data. It is optional and is generated if not provided.
region	Region is the GCS location in which your bucket exists. It is optional and is set based on the installed GCS Region.
projectID	ProjectID is the Project ID of the Google Cloud project that this bucket should be associated with. It is optional.
keyID	KeyID is the KMS key ID to use for encryption. It is optional because buckets are encrypted by default on Google Cloud. This allows for the use of a custom encryption key.

3.3. CONFIGURING THE REGISTRY FOR OPENSTACK USER-PROVISIONED INFRASTRUCTURE

You can configure the registry of a cluster that runs on your own Red Hat OpenStack Platform (RHOSP) infrastructure.

3.3.1. Configuring Image Registry Operator redirects

By disabling redirects, you can configure the Image Registry Operator to control whether clients such as OpenShift Container Platform cluster builds or external systems like developer machines are redirected to pull images directly from Red Hat OpenStack Platform (RHOSP) Swift storage. This configuration is optional and depends on whether the clients trust the storage's SSL/TLS certificates.



NOTE

In situations where clients do not trust the storage certificate, setting the **disableRedirect** option can be set to **true** proxies traffic through the image registry. Consequently, however, the image registry might require more resources, especially network bandwidth, to handle the increased load.

Alternatively, if clients trust the storage certificate, the registry can allow redirects. This reduces resource demand on the registry itself.

Some users might prefer to configure their clients to trust their self-signed certificate authorities (CAs) instead of disabling redirects. If you are using a self-signed CA, you must decide between trusting the custom CAs or disabling redirects.

Procedure

- To ensure that the image registry proxies traffic instead of relying on Swift storage, change the value of the **spec.disableRedirect** field in the **config.imageregistry** object to **true** by running the following command:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"disableRedirect":true}}'
```

3.3.2. Configuring a secret for the Image Registry Operator

In addition to the **configs.imageregistry.operator.openshift.io** and ConfigMap resources, configuration is provided to the Operator by a separate secret resource located within the **openshift-image-registry** namespace.

The **image-registry-private-configuration-user** secret provides credentials needed for storage access and management. It overrides the default credentials used by the Operator, if default credentials were found.

For Swift on Red Hat OpenStack Platform (RHOSP) storage, the secret is expected to contain the following two keys:

- **REGISTRY_STORAGE_SWIFT_USERNAME**
- **REGISTRY_STORAGE_SWIFT_PASSWORD**

Procedure

- Create an OpenShift Container Platform secret that contains the required keys.

```
$ oc create secret generic image-registry-private-configuration-user --from-literal=REGISTRY_STORAGE_SWIFT_USERNAME=<username> --from-literal=REGISTRY_STORAGE_SWIFT_PASSWORD=<password> -n openshift-image-registry
```

3.3.3. Registry storage for RHOSP with user-provisioned infrastructure

If the Registry Operator cannot create a Swift bucket, you must set up the storage medium manually and configure the settings in the registry custom resource (CR).

Prerequisites

- A cluster on Red Hat OpenStack Platform (RHOSP) with user-provisioned infrastructure.
- To configure registry storage for RHOSP, you need to provide Registry Operator cloud credentials.
- For Swift on RHOSP storage, the secret is expected to contain the following two keys:
 - **REGISTRY_STORAGE_SWIFT_USERNAME**
 - **REGISTRY_STORAGE_SWIFT_PASSWORD**

Procedure

- Fill in the storage configuration in **configs.imageregistry.operator.openshift.io/cluster**:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

Example configuration

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  storage:
    swift:
      container: <container_id>
```

3.3.4. Image Registry Operator configuration parameters for RHOSP Swift

The following parameters are available for you to configure your Red Hat OpenStack Platform (RHOSP) Swift registry storage.

Parameter	Description
authURL	Defines the URL for obtaining the authentication token. This value is optional.
authVersion	Specifies the Auth version of RHOSP, for example, authVersion: "3" . This value is optional.
container	Defines the name of a Swift container for storing registry data. This value is optional.
domain	Specifies the RHOSP domain name for the Identity v3 API. This value is optional.
domainID	Specifies the RHOSP domain ID for the Identity v3 API. This value is optional.
tenant	Defines the RHOSP tenant name to be used by the registry. This value is optional.
tenantID	Defines the RHOSP tenant ID to be used by the registry. This value is optional.
regionName	Defines the RHOSP region in which the container exists. This value is optional.

3.4. CONFIGURING THE REGISTRY FOR AZURE USER-PROVISIONED INFRASTRUCTURE

Save your container images to a durable storage location by configuring the built-in image registry to use dedicated Azure storage. This setup provides persistent scalable storage for your registry, separate from ephemeral cluster storage.

3.4.1. Configuring a secret for the Image Registry Operator

In addition to the **configs.imageregistry.operator.openshift.io** and ConfigMap resources, configuration is provided to the Operator by a separate secret resource located within the **openshift-image-registry** namespace.

The **image-registry-private-configuration-user** secret provides credentials needed for storage access and management. It overrides the default credentials used by the Operator, if default credentials were found.

For Azure registry storage, the secret is expected to contain one key whose value is the contents of a credentials file provided by Azure:

- **REGISTRY_STORAGE_AZURE_ACCOUNTKEY**

Procedure

- Create an OpenShift Container Platform secret that contains the required key.

```
$ oc create secret generic image-registry-private-configuration-user --from-literal=REGISTRY_STORAGE_AZURE_ACCOUNTKEY=<accountkey> --namespace openshift-image-registry
```

3.4.2. Configuring registry storage for Azure

During installation, your cloud credentials are sufficient to create Azure Blob Storage, and the Registry Operator automatically configures storage.

Prerequisites

- A cluster on Azure with user-provisioned infrastructure.
- To configure registry storage for Azure, provide Registry Operator cloud credentials.
- For Azure storage the secret is expected to contain one key:
 - **REGISTRY_STORAGE_AZURE_ACCOUNTKEY**

Procedure

1. Create an [Azure storage container](#).
2. Fill in the storage configuration in **configs.imageregistry.operator.openshift.io/cluster**:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

Example configuration

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  storage:
```

```

azure:
  accountName: <storage_account_name>
  container: <container_name>

```

3.4.3. Configuring registry storage for Azure Government

During installation, your cloud credentials are sufficient to create Azure Blob Storage, and the Registry Operator automatically configures storage.

Prerequisites

- A cluster on Azure with user-provisioned infrastructure in a government region.
- To configure registry storage for Azure, provide Registry Operator cloud credentials.
- For Azure storage, the secret is expected to contain one key:
 - **REGISTRY_STORAGE_AZURE_ACCOUNTKEY**

Procedure

1. Create an [Azure storage container](#).
2. Fill in the storage configuration in **configs.imageregistry.operator.openshift.io/cluster**:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

Example configuration

```

storage:
  azure:
    accountName: <storage-account-name>
    container: <container-name>
    cloudName: AzureUSGovernmentCloud

```

cloudName is the name of the Azure cloud environment, which can be used to configure the Azure SDK with the appropriate Azure API endpoints. Defaults to **AzurePublicCloud**. You can also set **cloudName** to **AzureUSGovernmentCloud**, **AzureChinaCloud**, or **AzureGermanCloud** with sufficient credentials.

3.5. CONFIGURING THE REGISTRY FOR RHOSP

You configure the image registry to use custom storage on clusters that run on RHOSP. You must configure custom storage to use a Cinder volume in a specific availability zone.

3.5.1. Configuring an image registry with custom storage on clusters that run on RHOSP

After you install a cluster on Red Hat OpenStack Platform (RHOSP), you can use a Cinder volume that is in a specific availability zone for registry storage.

Procedure

1. Create a YAML file that specifies the storage class and availability zone to use. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: custom-csi-storageclass
provisioner: cinder.csi.openstack.org
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
parameters:
  availability: <availability_zone_name>
```



NOTE

OpenShift Container Platform does not verify the existence of the availability zone you choose. Verify the name of the availability zone before you apply the configuration.

2. From a command line, apply the configuration:

```
$ oc apply -f <storage_class_file_name>
```

Example output

```
storageclass.storage.k8s.io/custom-csi-storageclass created
```

3. Create a YAML file that specifies a persistent volume claim (PVC) that uses your storage class and the **openshift-image-registry** namespace. For example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-imageregistry
  namespace: openshift-image-registry
  annotations:
    imageregistry.openshift.io: "true"
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 100Gi
  storageClassName: <your_custom_storage_class>
```

where:

openshift-image-registry

Specifying this namespace allows the Cluster Image Registry Operator to consume the PVC.

storage

This optional field adjusts the volume size.

storageClassName

Specifies the name of the storage class that you created.

- From a command line, apply the configuration:

```
$ oc apply -f <pvc_file_name>
```

Example output

```
persistentvolumeclaim/csi-pvc-imageregistry created
```

- Replace the original persistent volume claim in the image registry configuration with the new claim:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --type 'json' -p='[{"op":
"replace", "path": "/spec/storage/pvc/claim", "value": "csi-pvc-imageregistry"}]'
```

Example output

```
config.imageregistry.operator.openshift.io/cluster patched
```

Over the next several minutes, the configuration is updated.

Verification

To confirm that the registry is using the resources that you defined:

- Verify that the PVC claim value is identical to the name that you provided in your PVC definition:

```
$ oc get configs.imageregistry.operator.openshift.io/cluster -o yaml
```

Example output

```
...
status:
...
  managementState: Managed
  pvc:
    claim: csi-pvc-imageregistry
...
```

- Verify that the status of the PVC is **Bound**:

```
$ oc get pvc -n openshift-image-registry csi-pvc-imageregistry
```

Example output

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
STORAGECLASS	AGE			
csi-pvc-imageregistry	Bound	pvc-72a8f9c9-f462-11e8-b6b6-fa163e18b7b5	100Gi	
RWO		custom-csi-storageclass	11m	

3.6. CONFIGURING THE REGISTRY FOR BARE METAL

Configure image registry storage for bare-metal clusters after installation. Because bare-metal installations do not automatically provision storage, you must change the registry management state from **Removed** to **Managed** and configure persistent storage or use Red Hat OpenShift Data Foundation before the registry can store container images.

3.6.1. Image registry removed during installation

On platforms that do not provide shareable object storage, the OpenShift Image Registry Operator bootstraps itself as **Removed**. This allows **openshift-installer** to complete installations on these platform types.

After installation, you must edit the Image Registry Operator configuration to switch the **managementState** from **Removed** to **Managed**. When this has completed, you must configure storage.

3.6.2. Changing the image registry's management state

To start the image registry, you must change the Image Registry Operator configuration's **managementState** from **Removed** to **Managed**.

Procedure

- Change **managementState** Image Registry Operator configuration from **Removed** to **Managed**. For example:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"managementState": "Managed"}}'
```

3.6.3. Image registry storage configuration

The Image Registry Operator is not initially available for platforms that do not provide default storage. After installation, you must configure your registry to use storage so that the Registry Operator is made available.

Instructions are shown for configuring a persistent volume, which is required for production clusters. Where applicable, instructions are shown for configuring an empty directory as the storage location, which is available for only non-production clusters.

Additional instructions are provided for allowing the image registry to use block storage types by using the **Recreate** rollout strategy during upgrades.

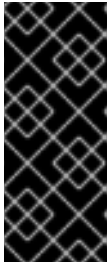
3.6.3.1. Configuring registry storage for bare metal and other manual installations

As a cluster administrator, following installation you must configure your registry to use storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a cluster that uses manually-provisioned Red Hat Enterprise Linux CoreOS (RHCOS) nodes, such as bare metal.

- You have provisioned persistent storage for your cluster, such as Red Hat OpenShift Data Foundation.



IMPORTANT

OpenShift Container Platform supports **ReadWriteOnce** access for image registry storage when you have only one replica. **ReadWriteOnce** access also requires that the registry uses the **Recreate** rollout strategy. To deploy an image registry that supports high availability with two or more replicas, **ReadWriteMany** access is required.

- Must have 100Gi capacity.

Procedure

1. To configure your registry to use storage, change the **spec.storage.pvc** in the **configs.imageregistry/cluster** resource.



NOTE

When you use shared storage, review your security settings to prevent outside access.

2. Verify that you do not have a registry pod:

```
$ oc get pod -n openshift-image-registry -l docker-registry=default
```

Example output

```
No resources found in openshift-image-registry namespace
```



NOTE

If you do have a registry pod in your output, you do not need to continue with this procedure.

3. Check the registry configuration:

```
$ oc edit configs.imageregistry.operator.openshift.io
```

Example output

```
storage:
  pvc:
    claim:
```

Leave the **claim** field blank to allow the automatic creation of an **image-registry-storage** PVC.

4. Check the **clusteroperator** status:

```
$ oc get clusteroperator image-registry
```

-

Example output

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
image-registry	4.20	True	False	False	6h50m

5. Ensure that your registry is set to managed to enable building and pushing of images.

- Run:

```
$ oc edit configs.imageregistry/cluster
```

Then, change the line

```
managementState: Removed
```

to

```
managementState: Managed
```

3.6.3.2. Configuring storage for the image registry in non-production clusters

You must configure storage for the Image Registry Operator. For non-production clusters, you can set the image registry to an empty directory. If you do so, all images are lost if you restart the registry.

Procedure

- To set the image registry storage to an empty directory:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"storage":{"emptyDir":{}}}}'
```

**WARNING**

Configure this option for only non-production clusters.

If you run this command before the Image Registry Operator initializes its components, the **oc patch** command fails with the following error:

```
Error from server (NotFound): configs.imageregistry.operator.openshift.io "cluster" not found
```

Wait a few minutes and run the command again.

3.6.3.3. Configuring block registry storage for bare metal

To allow the image registry to use block storage types during upgrades as a cluster administrator, you can use the **Recreate** rollout strategy.



IMPORTANT

Block storage volumes, or block persistent volumes, are supported but not recommended for use with the image registry on production clusters. An installation where the registry is configured on block storage is not highly available because the registry cannot have more than one replica.

If you choose to use a block storage volume with the image registry, you must use a filesystem persistent volume claim (PVC).

Procedure

1. Enter the following command to set the image registry storage as a block storage type, patch the registry so that it uses the **Recreate** rollout strategy, and runs with only one (**1**) replica:

```
$ oc patch config.imageregistry.operator.openshift.io/cluster --type=merge -p '{"spec": {"rolloutStrategy": "Recreate", "replicas": 1}}'
```

2. Provision the PV for the block storage device, and create a PVC for that volume. The requested block volume uses the ReadWriteOnce (RWO) access mode.
 - a. Create a **pvc.yaml** file with the following contents to define a VMware vSphere **PersistentVolumeClaim** object:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: image-registry-storage
  namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
```

where:

name

Specifies a unique name that represents the **PersistentVolumeClaim** object.

namespace

Specifies the **namespace** for the **PersistentVolumeClaim** object, which is **openshift-image-registry**.

accessModes

Specifies the access mode of the persistent volume claim. With **ReadWriteOnce**, the volume can be mounted with read and write permissions by a single node.

storage

The size of the persistent volume claim.

b. Enter the following command to create the **PersistentVolumeClaim** object from the file:

```
$ oc create -f pvc.yaml -n openshift-image-registry
```

3. Enter the following command to edit the registry configuration so that it references the correct PVC:

```
$ oc edit config.imageregistry.operator.openshift.io -o yaml
```

Example output

```
storage:
  pvc:
    claim:
```

By creating a custom PVC, you can leave the **claim** field blank for the default automatic creation of an **image-registry-storage** PVC.

3.6.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following procedure to configure the image registry to use Ceph RGW storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Ceph RGW object storage.

Procedure

1. Create the object bucket claim using the **ocs-storagecluster-ceph-rgw** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: rgwbucket
  namespace: openshift-storage
spec:
```

```
storageClassName: ocs-storagecluster-ceph-rgw
generateBucketName: rgwbucket
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage rgwbucket -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_ACCESS_KEY_ID}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_SECRET_ACCESS_KEY}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry
```

5. Get the **route** host by entering the following command:

```
$ route_host=$(oc get route ocs-storagecluster-cephobjectstore -n openshift-storage --
template='{{ .spec.host }}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default" -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config
```

7. Configure the image registry to use the Ceph RGW object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge
```

3.6.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following the procedure to configure the image registry to use Noobaa storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Noobaa object storage.

Procedure

1. Create the object bucket claim using the **openshift-storage.noobaa.io** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: noobaatest
  namespace: openshift-storage
spec:
  storageClassName: openshift-storage.noobaa.io
  generateBucketName: noobaatest
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage noobaatest -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage noobaatest -o yaml | grep -w
"AWS_ACCESS_KEY_ID:" | head -n1 | awk '{print $2}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage noobaatest -o yaml |
grep -w "AWS_SECRET_ACCESS_KEY:" | head -n1 | awk '{print $2}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry
```

5. Get the route host by entering the following command:

```
$ route_host=$(oc get route s3 -n openshift-storage -o=jsonpath='{.spec.host}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default"' -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config
```

7. Configure the image registry to use the Nooba object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge
```

3.6.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following procedure to configure the image registry to use CephFS storage.



NOTE

CephFS uses persistent volume claim (PVC) storage. It is not recommended to use PVCs for image registry storage if there are other options available, such as Ceph RGW or Noobaa.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.

- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and CephFS file storage.

Procedure

1. Create a PVC to use the **cephfs** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-storage-pvc
  namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: ocs-storagecluster-cephfs
EOF
```

2. Configure the image registry to use the CephFS file system storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","pvc":{"claim":"registry-storage-pvc"}}}' --type=merge
```

3.6.5. Additional resources

- [Recommended configurable storage technology](#)
- [Configuring Image Registry to use OpenShift Data Foundation](#)

3.7. CONFIGURING THE REGISTRY FOR VSPHERE

Configure image registry storage for vSphere clusters after installation. Because vSphere installations do not automatically provision storage, you must change the registry management state from **Removed** to **Managed** and configure persistent storage or use Red Hat OpenShift Data Foundation before the registry can store container images.

3.7.1. Image registry removed during installation

On platforms that do not provide shareable object storage, the OpenShift Image Registry Operator bootstraps itself as **Removed**. This allows **openshift-installer** to complete installations on these platform types.

After installation, you must edit the Image Registry Operator configuration to switch the **managementState** from **Removed** to **Managed**. When this has completed, you must configure storage.

3.7.2. Changing the image registry's management state

To start the image registry, you must change the Image Registry Operator configuration's **managementState** from **Removed** to **Managed**.

Procedure

- Change **managementState** Image Registry Operator configuration from **Removed** to **Managed**. For example:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"managementState": "Managed"}}'
```

3.7.3. Image registry storage configuration

The Image Registry Operator is not initially available for platforms that do not provide default storage. After installation, you must configure your registry to use storage so that the Registry Operator is made available.

Instructions are shown for configuring a persistent volume, which is required for production clusters. Where applicable, instructions are shown for configuring an empty directory as the storage location, which is available for only non-production clusters.

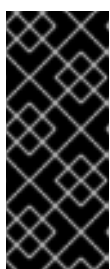
Additional instructions are provided for allowing the image registry to use block storage types by using the **Recreate** rollout strategy during upgrades.

3.7.3.1. Configuring registry storage for VMware vSphere

As a cluster administrator, following installation you must configure your registry to use storage.

Prerequisites

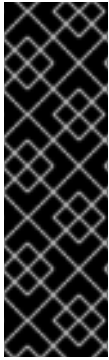
- Cluster administrator permissions.
- A cluster on VMware vSphere.
- Persistent storage provisioned for your cluster, such as Red Hat OpenShift Data Foundation.



IMPORTANT

OpenShift Container Platform supports **ReadWriteOnce** access for image registry storage when you have only one replica. **ReadWriteOnce** access also requires that the registry uses the **Recreate** rollout strategy. To deploy an image registry that supports high availability with two or more replicas, **ReadWriteMany** access is required.

- Must have "100Gi" capacity.



IMPORTANT

Testing shows issues with using the NFS server on RHEL as storage backend for core services. This includes the OpenShift Container Registry and Quay, Prometheus for monitoring storage, and Elasticsearch for logging storage. Therefore, using RHEL NFS to back PVs used by core services is not recommended.

Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift Container Platform core components.

Procedure

1. Change the **spec.storage.pvc** field in the **configs.imageregistry/cluster** resource.



NOTE

When you use shared storage, review your security settings to prevent outside access.

2. Verify that you do not have a registry pod by running the following command:

```
$ oc get pod -n openshift-image-registry -l docker-registry=default
```

Example output

```
No resources found in openshift-image-registry namespace
```



NOTE

If you do have a registry pod in your output, you do not need to continue with this procedure.

3. Check the registry configuration by running the following command:

```
$ oc edit configs.imageregistry.operator.openshift.io
```

Example output

```
storage:
  pvc:
    claim:
```

Leave the **claim** field blank to allow the automatic creation of an **image-registry-storage** persistent volume claim (PVC). The PVC is generated based on the default storage class. However, be aware that the default storage class might provide ReadWriteOnce (RWO) volumes, such as a RADOS Block Device (RBD), which can cause issues when you replicate to more than one replica.

4. Check the **clusteroperator** status by running the following command:

```
$ oc get clusteroperator image-registry
```

Example output

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
MESSAGE					
image-registry	4.7	True	False	False	6h50m

3.7.3.2. Configuring storage for the image registry in non-production clusters

You must configure storage for the Image Registry Operator. For non-production clusters, you can set the image registry to an empty directory. If you do so, all images are lost if you restart the registry.

Procedure

- To set the image registry storage to an empty directory:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"storage":{"emptyDir":{}}}}'
```



WARNING

Configure this option for only non-production clusters.

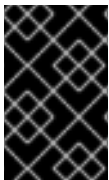
If you run this command before the Image Registry Operator initializes its components, the **oc patch** command fails with the following error:

```
Error from server (NotFound): configs.imageregistry.operator.openshift.io "cluster" not found
```

Wait a few minutes and run the command again.

3.7.3.3. Configuring block registry storage for VMware vSphere

To allow the image registry to use block storage types such as vSphere Virtual Machine Disk (VMDK) during upgrades as a cluster administrator, you can use the **Recreate** rollout strategy.



IMPORTANT

Block storage volumes are supported but not recommended for use with image registry on production clusters. An installation where the registry is configured on block storage is not highly available because the registry cannot have more than one replica.

Procedure

- Enter the following command to set the image registry storage as a block storage type, patch the registry so that it uses the **Recreate** rollout strategy, and runs with only **1** replica:

```
$ oc patch config.imageregistry.operator.openshift.io/cluster --type=merge -p '{"spec": {"rolloutStrategy":"Recreate","replicas":1}}'
```

2. Provision the persistent volume (PV) for the block storage device, and create a persistent volume claim (PVC) for that volume. The requested block volume uses the ReadWriteOnce (RWO) access mode.
 - a. Create a **pvc.yaml** file with the following contents to define a VMware vSphere **PersistentVolumeClaim** object:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: image-registry-storage
  namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
```

where:

name

Specifies a unique name that represents the **PersistentVolumeClaim** object.

namespace

Specifies the **namespace** for the **PersistentVolumeClaim** object, which is **openshift-image-registry**.

accessModes

Specifies the access mode of the persistent volume claim. With **ReadWriteOnce**, the volume can be mounted with read and write permissions by a single node.

storage

The size of the persistent volume claim.

- b. Enter the following command to create the **PersistentVolumeClaim** object from the file:

```
$ oc create -f pvc.yaml -n openshift-image-registry
```

3. Enter the following command to edit the registry configuration so that it references the correct PVC:

```
$ oc edit config.imageregistry.operator.openshift.io -o yaml
```

Example output

```
storage:
pvc:
claim:
```

By creating a custom PVC, you can leave the **claim** field blank for the default automatic creation of an **image-registry-storage** PVC.

3.7.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following, procedure to configure the image registry to use Ceph RGW storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Ceph RGW object storage.

Procedure

1. Create the object bucket claim using the **ocs-storagecluster-ceph-rgw** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: rgwbucket
  namespace: openshift-storage
spec:
  storageClassName: ocs-storagecluster-ceph-rgw
  generateBucketName: rgwbucket
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage rgwbucket -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_ACCESS_KEY_ID}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_SECRET_ACCESS_KEY}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry
```

5. Get the **route** host by entering the following command:

```
$ route_host=$(oc get route ocs-storagecluster-cephobjectstore -n openshift-storage --
template='{{ .spec.host }}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default" -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config
```

7. Configure the image registry to use the Ceph RGW object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge
```

3.7.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following the procedure to configure the image registry to use Noobaa storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Noobaa object storage.

Procedure

Procedure

1. Create the object bucket claim using the **openshift-storage.noobaa.io** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: noobaatest
  namespace: openshift-storage
spec:
  storageClassName: openshift-storage.noobaa.io
  generateBucketName: noobaatest
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage noobaatest -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage noobaatest -o yaml | grep -w
"AWS_ACCESS_KEY_ID:" | head -n1 | awk '{print $2}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage noobaatest -o yaml |
grep -w "AWS_SECRET_ACCESS_KEY:" | head -n1 | awk '{print $2}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry
```

5. Get the route host by entering the following command:

```
$ route_host=$(oc get route s3 -n openshift-storage -o=jsonpath='{.spec.host}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default"' -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config
```

7. Configure the image registry to use the Nooba object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge
```

3.7.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following procedure to configure the image registry to use CephFS storage.



NOTE

CephFS uses persistent volume claim (PVC) storage. It is not recommended to use PVCs for image registry storage if there are other options available, such as Ceph RGW or Noobaa.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and CephFS file storage.

Procedure

1. Create a PVC to use the **cephfs** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-storage-pvc
  namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: ocs-storagecluster-cephfs
EOF
```

2. Configure the image registry to use the CephFS file system storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","pvc":{"claim":"registry-storage-pvc"}}}' --type=merge
```

3.7.5. Additional resources

- [Configuring the registry for vSphere](#)
- [Recommended configurable storage technology](#)
- [Configuring Image Registry to use OpenShift Data Foundation](#)

3.8. CONFIGURING THE REGISTRY FOR RED HAT OPENSIFT DATA FOUNDATION

To configure the OpenShift image registry on bare metal and vSphere to use Red Hat OpenShift Data Foundation storage, you must install OpenShift Data Foundation and then configure image registry using Ceph or Noobaa.

3.8.1. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following, procedure to configure the image registry to use Ceph RGW storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Ceph RGW object storage.

Procedure

1. Create the object bucket claim using the **ocs-storagecluster-ceph-rgw** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
```

```

name: rgwbucket
namespace: openshift-storage
spec:
  storageClassName: ocs-storagecluster-ceph-rgw
  generateBucketName: rgwbucket
EOF

```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```

$ bucket_name=$(oc get obc -n openshift-storage rgwbucket -o
jsonpath='{.spec.bucketName}')

```

3. Get the AWS credentials by entering the following commands:

```

$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_ACCESS_KEY_ID}' | base64 --decode)

```

```

$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_SECRET_ACCESS_KEY}' | base64 --decode)

```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```

$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry

```

5. Get the **route** host by entering the following command:

```

$ route_host=$(oc get route ocs-storagecluster-cephobjectstore -n openshift-storage --
template='{.spec.host }')

```

6. Create a config map that uses an ingress certificate by entering the following commands:

```

$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default"' -r) -n openshift-ingress --confirm

```

```

$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config

```

7. Configure the image registry to use the Ceph RGW object storage by entering the following command:

```

$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge

```

3.8.2. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following the procedure to configure the image registry to use Noobaa storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Noobaa object storage.

Procedure

1. Create the object bucket claim using the **openshift-storage.noobaa.io** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: noobaatest
  namespace: openshift-storage
spec:
  storageClassName: openshift-storage.noobaa.io
  generateBucketName: noobaatest
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage noobaatest -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage noobaatest -o yaml | grep -w
"AWS_ACCESS_KEY_ID:" | head -n1 | awk '{print $2}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage noobaatest -o yaml |
grep -w "AWS_SECRET_ACCESS_KEY:" | head -n1 | awk '{print $2}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --namespace openshift-image-registry
```

5. Get the route host by entering the following command:

```
$ route_host=$(oc get route s3 -n openshift-storage -o=jsonpath='{.spec.host}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq '.spec.defaultCertificate.name // "router-certs-default"' -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n openshift-config
```

7. Configure the image registry to use the Nooba object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":{"managementState":"Managed","replicas":2,"storage":{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedCA":{"name":"image-registry-s3-bundle"}}}}}' --type=merge
```

3.8.3. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following procedure to configure the image registry to use CephFS storage.



NOTE

CephFS uses persistent volume claim (PVC) storage. It is not recommended to use PVCs for image registry storage if there are other options available, such as Ceph RGW or Noobaa.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.

- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and CephFS file storage.

Procedure

1. Create a PVC to use the **cephfs** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-storage-pvc
  namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: ocs-storagecluster-cephfs
EOF
```

2. Configure the image registry to use the CephFS file system storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","pvc":{"claim":"registry-storage-pvc"}}}' --type=merge
```

3.8.4. Additional resources

- [Configuring Image Registry to use OpenShift Data Foundation](#)
- [Performance tuning guide for Multicloud Object Gateway \(NooBaa\)](#)

3.9. CONFIGURING THE REGISTRY FOR NUTANIX

Users can optimize container image distribution, security, and access controls, enabling a robust foundation for Nutanix applications on OpenShift Container Platform

3.9.1. Image registry removed during installation

On platforms that do not provide shareable object storage, the OpenShift Image Registry Operator bootstraps itself as **Removed**. This allows **openshift-installer** to complete installations on these platform types.

After installation, you must edit the Image Registry Operator configuration to switch the **managementState** from **Removed** to **Managed**. When this has completed, you must configure storage.

3.9.2. Changing the image registry's management state

To start the image registry, you must change the Image Registry Operator configuration's **managementState** from **Removed** to **Managed**.

Procedure

- Change **managementState** Image Registry Operator configuration from **Removed** to **Managed**. For example:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"managementState": "Managed"}}'
```

3.9.3. Image registry storage configuration

The Image Registry Operator is not initially available for platforms that do not provide default storage. After installation, you must configure your registry to use storage so that the Registry Operator is made available.

Instructions are shown for configuring a persistent volume, which is required for production clusters. Where applicable, instructions are shown for configuring an empty directory as the storage location, which is available for only non-production clusters.

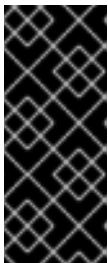
Additional instructions are provided for allowing the image registry to use block storage types by using the **Recreate** rollout strategy during upgrades.

3.9.3.1. Configuring registry storage for Nutanix

As a cluster administrator, following installation you must configure your registry to use storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a cluster on Nutanix.
- You have provisioned persistent storage for your cluster, such as Red Hat OpenShift Data Foundation.



IMPORTANT

OpenShift Container Platform supports **ReadWriteOnce** access for image registry storage when you have only one replica. **ReadWriteOnce** access also requires that the registry uses the **Recreate** rollout strategy. To deploy an image registry that supports high availability with two or more replicas, **ReadWriteMany** access is required.

- You must have 100 Gi capacity.

Procedure

1. To configure your registry to use storage, change the **spec.storage.pvc** in the **configs.imageregistry/cluster** resource.

**NOTE**

When you use shared storage, review your security settings to prevent outside access.

2. Verify that you do not have a registry pod:

```
$ oc get pod -n openshift-image-registry -l docker-registry=default
```

Example output

```
No resources found in openshift-image-registry namespace
```

**NOTE**

If you do have a registry pod in your output, you do not need to continue with this procedure.

3. Check the registry configuration:

```
$ oc edit configs.imageregistry.operator.openshift.io
```

Example output

```
storage:
  pvc:
    claim:
```

Leave the **claim** field blank to allow the automatic creation of an **image-registry-storage** persistent volume claim (PVC). The PVC is generated based on the default storage class. However, be aware that the default storage class might provide ReadWriteOnce (RWO) volumes, such as a RADOS Block Device (RBD), which can cause issues when you replicate to more than one replica.

4. Check the **clusteroperator** status:

```
$ oc get clusteroperator image-registry
```

Example output

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED
image-registry	4.13	True	False	False
				6h50m

3.9.3.2. Configuring storage for the image registry in non-production clusters

You must configure storage for the Image Registry Operator. For non-production clusters, you can set the image registry to an empty directory. If you do so, all images are lost if you restart the registry.

Procedure

- To set the image registry storage to an empty directory:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"storage":{"emptyDir":{}}}}'
```



WARNING

Configure this option for only non-production clusters.

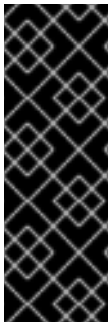
If you run this command before the Image Registry Operator initializes its components, the **oc patch** command fails with the following error:

```
Error from server (NotFound): configs.imageregistry.operator.openshift.io "cluster" not found
```

Wait a few minutes and run the command again.

3.9.3.3. Configuring block registry storage for Nutanix volumes

To allow the image registry to use block storage types such as Nutanix volumes during upgrades as a cluster administrator, you can use the **Recreate** rollout strategy.



IMPORTANT

Block storage volumes, or block persistent volumes, are supported but not recommended for use with the image registry on production clusters. An installation where the registry is configured on block storage is not highly available because the registry cannot have more than one replica.

If you choose to use a block storage volume with the image registry, you must use a filesystem persistent volume claim (PVC).

Procedure

1. Enter the following command to set the image registry storage as a block storage type, patch the registry so that it uses the **Recreate** rollout strategy, and runs with only one (**1**) replica:

```
$ oc patch config.imageregistry.operator.openshift.io/cluster --type=merge -p '{"spec": {"rolloutStrategy":"Recreate","replicas":1}}'
```

2. Provision the PV for the block storage device, and create a PVC for that volume. The requested block volume uses the ReadWriteOnce (RWO) access mode.
 - a. Create a **pvc.yaml** file with the following contents to define a Nutanix **PersistentVolumeClaim** object:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
```

```

name: image-registry-storage
namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi

```

where:

name

Specifies a unique name that represents the **PersistentVolumeClaim** object.

namespace

Specifies the namespace for the **PersistentVolumeClaim** object, which is **openshift-image-registry**.

ReadWriteOnce

Specifies the access mode of the persistent volume claim. With **ReadWriteOnce**, the volume can be mounted with read and write permissions by a single node.

storage

Specifies the size of the persistent volume claim.

- b. Enter the following command to create the **PersistentVolumeClaim** object from the file:

```
$ oc create -f pvc.yaml -n openshift-image-registry
```

3. Enter the following command to edit the registry configuration so that it references the correct PVC:

```
$ oc edit config.imageregistry.operator.openshift.io -o yaml
```

Example output

```

storage:
  pvc:
    claim: 1

```

By creating a custom PVC, you can leave the **claim** field blank for the default automatic creation of an **image-registry-storage** PVC.

3.9.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following procedure to configure the image registry to use Ceph RGW storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Ceph RGW object storage.

Procedure

1. Create the object bucket claim using the **ocs-storagecluster-ceph-rgw** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: rgwbucket
  namespace: openshift-storage
spec:
  storageClassName: ocs-storagecluster-ceph-rgw
  generateBucketName: rgwbucket
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage rgwbucket -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_ACCESS_KEY_ID}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage rgwbucket -o
jsonpath='{.data.AWS_SECRET_ACCESS_KEY}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry** project by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry
```

5. Get the **route** host by entering the following command:

```
$ route_host=$(oc get route ocs-storagecluster-cephobjectstore -n openshift-storage --
template='{{ .spec.host }}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default" -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config
```

7. Configure the image registry to use the Ceph RGW object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge
```

3.9.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following the procedure to configure the image registry to use Noobaa storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and Noobaa object storage.

Procedure

1. Create the object bucket claim using the **openshift-storage.noobaa.io** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: noobaatest
  namespace: openshift-storage
spec:
```

```
storageClassName: openshift-storage.noobaa.io
generateBucketName: noobaatest
EOF
```

Alternatively, you can use the **openshift-image-registry** for the **namespace** value.

2. Get the bucket name by entering the following command:

```
$ bucket_name=$(oc get obc -n openshift-storage noobaatest -o
jsonpath='{.spec.bucketName}')
```

3. Get the AWS credentials by entering the following commands:

```
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage noobaatest -o yaml | grep -w
"AWS_ACCESS_KEY_ID:" | head -n1 | awk '{print $2}' | base64 --decode)
```

```
$ AWS_SECRET_ACCESS_KEY=$(oc get secret -n openshift-storage noobaatest -o yaml |
grep -w "AWS_SECRET_ACCESS_KEY:" | head -n1 | awk '{print $2}' | base64 --decode)
```

4. Create the secret **image-registry-private-configuration-user** with the AWS credentials for the new bucket under **openshift-image-registry project** by entering the following command:

```
$ oc create secret generic image-registry-private-configuration-user --from-
literal=REGISTRY_STORAGE_S3_ACCESSKEY=${AWS_ACCESS_KEY_ID} --from-
literal=REGISTRY_STORAGE_S3_SECRETKEY=${AWS_SECRET_ACCESS_KEY} --
namespace openshift-image-registry
```

5. Get the route host by entering the following command:

```
$ route_host=$(oc get route s3 -n openshift-storage -o=jsonpath='{.spec.host}')
```

6. Create a config map that uses an ingress certificate by entering the following commands:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq
'.spec.defaultCertificate.name // "router-certs-default"' -r) -n openshift-ingress --confirm
```

```
$ oc create configmap image-registry-s3-bundle --from-file=ca-bundle.crt=./tls.crt -n
openshift-config
```

7. Configure the image registry to use the Nooba object storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","s3":{"bucket":"${bucket_name}","region":"us-east-
1","regionEndpoint":"https://${route_host}","virtualHostedStyle":false,"encrypt":false,"trustedC
A":{"name":"image-registry-s3-bundle"}}}}' --type=merge
```

3.9.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the OpenShift image registry:

- Ceph, a shared and distributed file system and on-premise object storage
- NooBaa, providing a Multicloud Object Gateway

Use the following procedure to configure the image registry to use CephFS storage.



NOTE

CephFS uses persistent volume claim (PVC) storage. It is not recommended to use PVCs for image registry storage if there are other options available, such as Ceph RGW or Noobaa.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- You installed the **oc** CLI.
- You installed the [OpenShift Data Foundation Operator](#) to provide object storage and CephFS file storage.

Procedure

1. Create a PVC to use the **cephfs** storage class. For example:

```
cat <<EOF | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-storage-pvc
  namespace: openshift-image-registry
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: ocs-storagecluster-cephfs
EOF
```

2. Configure the image registry to use the CephFS file system storage by entering the following command:

```
$ oc patch config.image/cluster -p '{"spec":
{"managementState":"Managed","replicas":2,"storage":
{"managementState":"Unmanaged","pvc":{"claim":"registry-storage-pvc"}}}' --type=merge
```

3.9.5. Additional resources

- [Recommended configurable storage technology](#)

- [Configuring Image Registry to use OpenShift Data Foundation](#)

CHAPTER 4. ACCESSING THE REGISTRY

You can access a registry to view logs and metrics. You can also secure and expose the registry.

After you logged in to the registry by using the **podman login** command, you can push or pull images from the integrated registry directly by using **podman push** or **podman pull** commands. The commands that you can use depend on your user permissions.

4.1. PREREQUISITES

- You have access to the cluster as a user with the **cluster-admin** role.
- You must have configured an identity provider (IDP).
- For pulling images, for example when using the **podman pull** command, the user must have the **registry-viewer** role. To add this role, run the following command:

```
$ oc policy add-role-to-user registry-viewer <user_name>
```

- For writing or pushing images, such as using **podman push** command, complete the following steps:
 - Your account has the **registry-editor** role. To add this role, run the following command:

```
$ oc policy add-role-to-user registry-editor <user_name>
```

- Your cluster must have an existing project where the images can be pushed to.

4.2. ACCESSING THE REGISTRY DIRECTLY FROM THE CLUSTER

You can access the registry from inside the cluster by using internal routes.

Procedure

1. Access the node by getting its name:

```
$ oc get nodes
```

```
$ oc debug nodes/<node_name>
```

2. To enable access to tools such as **oc** and **podman** on the node, change your root directory to **/host**. Successful output on running the commands states **Login Succeeded!**.

```
sh-4.2# chroot /host
```

3. Log in to the container image registry by using your access token:

```
sh-4.2# oc login -u kubeadmin -p <password_from_install_log> https://api-int.
<cluster_name>.<base_domain>:6443
```

```
sh-4.2# podman login -u kubeadmin -p $(oc whoami -t) image-registry.openshift-image-registry.svc:5000
```



NOTE

You can pass almost any value for the user name. The token contains all necessary information. Passing a user name that contains colons results in a login failure.

The Image Registry Operator creates the route, such as **default-route-openshift-image-registry.<cluster_name>**.

4. Perform **podman pull** and **podman push** operations against your registry. The following example commands demonstrate these operations.

- a. Pull an arbitrary image:

```
sh-4.2# podman pull <name.io>/<image>
```



IMPORTANT

You can pull arbitrary images, but if you have the **system:registry** role added, you can only push images to the registry in your project.

- b. Tag the new image with the form **<registry_ip>:<port>/<project>/<image>**. For example, **172.30.124.220:5000/openshift/image**. The project name must show in the pull specification for OpenShift Container Platform to correctly place and later access the image in the registry.

```
sh-4.2# podman tag <name.io>/<image> image-registry.openshift-image-registry.svc:5000/openshift/<image>
```



NOTE

You must have the **system:image-builder** role for the specified project, which allows the user to write or push an image. Otherwise, the **podman push** in the next step will fail. To test, you can create a new project to push the image.

- c. Push the newly tagged image to your registry:

```
sh-4.2# podman push image-registry.openshift-image-registry.svc:5000/openshift/<image>
```



NOTE

When pushing images to the internal registry, the repository name must use the **<project>/<name>** format. Using multiple project levels in the repository name results in an authentication error.

4.3. CHECKING THE STATUS OF THE REGISTRY PODS

As a cluster administrator, you can list the image registry pods running in the **openshift-image-registry** project and check their status.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- List the pods in the **openshift-image-registry** project and view their status. Example output provided for demonstrative purposes.

```
$ oc get pods -n openshift-image-registry
```

```
NAME READY STATUS RESTARTS AGE
image-registry-79fb4469f6-lrlrn 1/1 Running 0 77m
node-ca-hjksc 1/1 Running 0 73m
node-ca-tftj6 1/1 Running 0 77m
node-ca-wb6ht 1/1 Running 0 77m
node-ca-zvt9q 1/1 Running 0 74m
```

4.4. VIEWING REGISTRY LOGS

You can view the logs for the registry by using the **oc logs** command.

Procedure

- Use the **oc logs** command with deployments to view the logs for the container image registry. Example output provided for demonstrative purposes.

```
$ oc logs deployments/image-registry -n openshift-image-registry
```

```
2015-05-01T19:48:36.300593110Z time="2015-05-01T19:48:36Z" level=info
msg="version=v2.0.0+unknown"
2015-05-01T19:48:36.303294724Z time="2015-05-01T19:48:36Z" level=info msg="redis not
configured" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303422845Z time="2015-05-01T19:48:36Z" level=info msg="using
inmemory layerinfo cache" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303433991Z time="2015-05-01T19:48:36Z" level=info msg="Using
OpenShift Auth handler"
2015-05-01T19:48:36.303439084Z time="2015-05-01T19:48:36Z" level=info msg="listening
on :5000" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
```

4.5. ACCESSING REGISTRY METRICS

The OpenShift Container Registry provides an endpoint for [Prometheus metrics](#). Prometheus is a stand-alone, open source systems monitoring and alerting toolkit. The metrics get exposed at the **/extensions/v2/metrics** path of the registry endpoint. You can access the metrics by running a metrics query that includes a cluster role.

Procedure

1. Create a cluster role if you do not already have one to access the metrics:

```
$ cat <<EOF | oc create -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-scraper
rules:
- apiGroups:
  - image.openshift.io
  resources:
  - registry/metrics
  verbs:
  - get
EOF
```

2. Add the cluster role to a user account by entering the following command:

```
$ oc adm policy add-cluster-role-to-user prometheus-scraper <username>
```

3. For the metrics query, get the user token.

```
openshift:
$ oc whoami -t
```

4. Run a metrics query in node or inside a pod. The following example command and output demonstrate this task.

```
$ curl --insecure -s -u <user>:<secret> \ 1
https://image-registry.openshift-image-registry.svc:5000/extensions/v2/metrics | grep
imageregistry | head -n 20
```

- **<user>:<secret>**: The **<user>** object can be arbitrary, but **<secret>** tag must use the user token.

```
# HELP imageregistry_build_info A metric with a constant '1' value labeled by major,
minor, git commit & git version from which the image registry was built.
# TYPE imageregistry_build_info gauge
imageregistry_build_info{gitCommit="9f72191",gitVersion="v3.11.0+9f72191-135-
dirty",major="3",minor="11+"} 1
# HELP imageregistry_digest_cache_requests_total Total number of requests without
scope to the digest cache.
# TYPE imageregistry_digest_cache_requests_total counter
imageregistry_digest_cache_requests_total{type="Hit"} 5
imageregistry_digest_cache_requests_total{type="Miss"} 24
# HELP imageregistry_digest_cache_scoped_requests_total Total number of scoped
requests to the digest cache.
# TYPE imageregistry_digest_cache_scoped_requests_total counter
imageregistry_digest_cache_scoped_requests_total{type="Hit"} 33
imageregistry_digest_cache_scoped_requests_total{type="Miss"} 44
# HELP imageregistry_http_in_flight_requests A gauge of requests currently being
served by the registry.
# TYPE imageregistry_http_in_flight_requests gauge
```

```
imageregistry_http_in_flight_requests 1
# HELP imageregistry_http_request_duration_seconds A histogram of latencies for
requests to the registry.
# TYPE imageregistry_http_request_duration_seconds summary
imageregistry_http_request_duration_seconds{method="get",quantile="0.5"} 0.01296087
imageregistry_http_request_duration_seconds{method="get",quantile="0.9"}
0.014847248
imageregistry_http_request_duration_seconds{method="get",quantile="0.99"}
0.015981195
imageregistry_http_request_duration_seconds_sum{method="get"}
12.260727916000022
```

4.6. ADDITIONAL RESOURCES

- [Allowing pods to reference images across projects](#)
- [Removing the kubeadmin user](#)
- [Understanding identity provider configuration](#)

CHAPTER 5. EXPOSING THE REGISTRY

By default, the OpenShift image registry is secured during cluster installation so that it serves traffic through the Transport Layer Security (TLS) protocol. Unlike previous versions of OpenShift Container Platform, the registry is not exposed outside of the cluster at the time of installation.

5.1. EXPOSING A DEFAULT REGISTRY MANUALLY

Instead of logging in to the default OpenShift image registry from within the cluster, you can gain external access to the OpenShift image registry by exposing the registry with a route. With this external access, you can log in to the registry from outside the cluster by using the route address. You can then tag and push images to an existing project by using the route host.

Prerequisites

- The following prerequisites are automatically performed:
 - Deploy the Registry Operator.
 - Deploy the Ingress Operator.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. To expose the registry by using the **defaultRoute** parameter that exists in the **configs.imageregistry.operator.openshift.io** resource, set `defaultRoute`` to **true** by running the following command:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute":true}}' --type=merge
```

2. Get the default registry route by running the following command:

```
$ HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

3. Get the certificate of the Ingress Operator by running the following command:

```
$ oc extract secret/$(oc get ingresscontroller -n openshift-ingress-operator default -o json | jq '.spec.defaultCertificate.name // "router-certs-default"' -r) -n openshift-ingress --confirm
```

4. Move the extracted certificate to the trusted CA directory of the system by running the following command:

```
$ sudo mv tls.crt /etc/pki/ca-trust/source/anchors/
```

5. Enable the default certificate of the cluster to trust the route by running the following command:

```
$ sudo update-ca-trust enable
```

6. Log in with podman with the default route by running the following command:

```
$ sudo podman login -u kubeadmin -p $(oc whoami -t) $HOST
```

5.2. EXPOSING A SECURE REGISTRY MANUALLY

Instead of logging in to the OpenShift image registry from within the cluster, you can gain external access to the OpenShift image registry by exposing the registry with a route. With this external access, you can log in to the registry from outside the cluster by using the route address. You can then tag and push images to an existing project by using the route host.

You can expose the route by using **DefaultRoute** parameter in the **configs.imageregistry.operator.openshift.io** resource or by using custom routes.

Prerequisites

- The following prerequisites are automatically performed:
 - Deploy the Registry Operator.
 - Deploy the Ingress Operator.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. To expose the registry using **DefaultRoute** parameter, set **DefaultRoute** to **True**:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute": true}}' --type=merge
```

2. Log in with **podman** by entering the following command:

```
$ HOST=$(oc get route default-route -n openshift-image-registry --template='{ { .spec.host } }')
```

```
$ podman login -u kubeadmin -p $(oc whoami -t) --tls-verify=false $HOST
```

- **--tls-verify=false**: Set this parameter to **false** if the default certificate of the cluster for routes is untrusted. You can set a custom, trusted certificate as the default certificate with the Ingress Operator.
3. To expose the registry using custom routes, create a secret with your route's TLS keys. This step is optional. If you do not create a secret, the route uses the default TLS configuration from the Ingress Operator.

```
$ oc create secret tls public-route-tls \
  -n openshift-image-registry \
  --cert=</path/to/tls.crt> \
  --key=</path/to/tls.key>
```

4. On the Registry Operator, enter the following command:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

```
spec:
  routes:
  - name: public-routes
    hostname: myregistry.mycorp.organization
    secretName: public-route-tls
  ...
```



NOTE

Only set **secretName** if you are providing a custom TLS configuration for the route of the registry.

Troubleshooting

- [Error creating TLS secret](#)