OpenShift Container Platform 4.11

Registry

Configuring registries for OpenShift Container Platform
OpenShift Container Platform 4.11 Registry

Configuring registries for OpenShift Container Platform
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 CONTAINER PLATFORM REGISTRY OVERVIEW

- 1.1. INTEGRATED OPENSIFT CONTAINER PLATFORM REGISTRY ........................................... 4
- 1.2. THIRD-PARTY REGISTRIES
  - 1.2.1. Authentication
    - 1.2.1.1. Registry authentication with Podman ...................................................... 4
- 1.3. RED HAT QUAY REGISTRIES
- 1.4. AUTHENTICATION ENABLED RED HAT REGISTRY .................................................. 5

## CHAPTER 2. IMAGE REGISTRY OPERATOR IN OPENSIFT CONTAINER PLATFORM

- 2.1. IMAGE REGISTRY ON CLOUD PLATFORMS AND OPENSTACK ................................. 7
- 2.2. IMAGE REGISTRY ON BARE METAL AND VSMPHIRE
  - 2.2.1. Image registry removed during installation .................................................... 7
- 2.3. IMAGE REGISTRY OPERATOR DISTRIBUTION ACROSS AVAILABILITY ZONES ............... 8
- 2.4. ADDITIONAL RESOURCES ..................................................................................... 9
- 2.5. IMAGE REGISTRY OPERATOR CONFIGURATION PARAMETERS ................................ 9
- 2.6. ENABLE THE IMAGE REGISTRY DEFAULT ROUTE WITH THE CUSTOM RESOURCE DEFINITION ......................................................... 10
- 2.7. CONFIGURING ADDITIONAL TRUST STORES FOR IMAGE REGISTRY ACCESS ............ 11
- 2.8. CONFIGURING STORAGE CREDENTIALS FOR THE IMAGE REGISTRY OPERATOR ... 12
- 2.9. ADDITIONAL RESOURCES ..................................................................................... 12

## CHAPTER 3. SETTING UP AND CONFIGURING THE REGISTRY

- 3.1. CONFIGURING THE REGISTRY FOR AWS USER-PROVISIONED INFRASTRUCTURE .......... 13
  - 3.1.1. Configuring a secret for the Image Registry Operator ........................................ 13
  - 3.1.2. Configuring registry storage for AWS with user-provisioned infrastructure .......... 13
  - 3.1.3. Image Registry Operator configuration parameters for AWS S3 ............................ 14
- 3.2. CONFIGURING THE REGISTRY FOR GCP USER-PROVISIONED INFRASTRUCTURE ....... 15
  - 3.2.1. Configuring a secret for the Image Registry Operator ........................................ 15
  - 3.2.2. Registry storage for GCP with user-provisioned infrastructure ............................ 15
  - 3.2.3. Image Registry Operator configuration parameters for GCP GCS ....................... 15
- 3.3. CONFIGURING THE REGISTRY FOR OPENSTACK USER-PROVISIONED INFRASTRUCTURE ................................................................. 16
  - 3.3.1. Configuring the Image Registry Operator to trust Swift storage .......................... 16
  - 3.3.2. Configuring a secret for the Image Registry Operator ........................................ 16
  - 3.3.3. Registry storage for RHOSP with user-provisioned infrastructure ........................ 17
  - 3.3.4. Image Registry Operator configuration parameters for RHOSP Swift .................. 17
- 3.4. CONFIGURING THE REGISTRY FOR AZURE USER-PROVISIONED INFRASTRUCTURE ........ 18
  - 3.4.1. Configuring a secret for the Image Registry Operator ........................................ 18
  - 3.4.2. Configuring registry storage for Azure ............................................................. 18
  - 3.4.3. Configuring registry storage for Azure Government ........................................... 19
- 3.5. CONFIGURING THE REGISTRY FOR RHOSP .......................................................... 19
  - 3.5.1. Configuring an image registry with custom storage on clusters that run on RHOSP 19
- 3.6. CONFIGURING THE REGISTRY FOR BARE METAL ................................................. 21
  - 3.6.1. Image registry removed during installation ....................................................... 21
  - 3.6.2. Changing the image registry’s management state .............................................. 22
  - 3.6.3. Image registry storage configuration
    - 3.6.3.1. Configuring registry storage for bare metal and other manual installations .... 22
    - 3.6.3.2. Configuring storage for the image registry in non-production clusters ........ 24
    - 3.6.3.3. Configuring block registry storage ............................................................ 24
    - 3.6.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation ................................................................. 25
    - 3.6.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation ................................................................. 27
  - 3.6.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation ................................................................. 27
Foundation
3.6.5. Additional resources
3.7. CONFIGURING THE REGISTRY FOR VSPHERE
3.7.1. Image registry removed during installation
3.7.2. Changing the image registry’s management state
3.7.3. Image registry storage configuration
3.7.3.1. Configuring registry storage for VMware vSphere
3.7.3.2. Configuring storage for the image registry in non-production clusters
3.7.3.3. Configuring block registry storage for VMware vSphere
3.7.3.4. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation
3.7.3.5. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation
3.7.4. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation
3.7.5. Additional resources
3.8. CONFIGURING THE REGISTRY FOR RED HAT OPENSHIFT DATA FOUNDATION
3.8.1. Configuring the Image Registry Operator to use Ceph RGW storage with Red Hat OpenShift Data Foundation
3.8.2. Configuring the Image Registry Operator to use Noobaa storage with Red Hat OpenShift Data Foundation
3.8.3. Configuring the Image Registry Operator to use CephFS storage with Red Hat OpenShift Data Foundation
3.8.4. Additional resources

CHAPTER 4. ACCESSING THE REGISTRY ............................................................... 43
4.1. PREREQUISITES 43
4.2. ACCESSING REGISTRY DIRECTLY FROM THE CLUSTER 43
4.3. CHECKING THE STATUS OF THE REGISTRY PODS 45
4.4. VIEWING REGISTRY LOGS 45
4.5. ACCESSING REGISTRY METRICS 46
4.6. ADDITIONAL RESOURCES 47

CHAPTER 5. EXPOSING THE REGISTRY ............................................................... 48
5.1. EXPOSING A DEFAULT REGISTRY MANUALLY 48
5.2. EXPOSING A SECURE REGISTRY MANUALLY 48
CHAPTER 1. OPENSOURCE CONTAINER PLATFORM REGISTRY

OVERVIEW

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

1.1. INTEGRATED OPENSOURCE CONTAINER PLATFORM 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. It provides an out-of-the-box solution for users to manage the images that run their workloads, and runs on top of the existing cluster infrastructure. This registry can be scaled up or down like any other cluster workload and does not require specific infrastructure provisioning. In addition, it is integrated into the cluster user authentication and authorization system, which means that access to create and retrieve images is controlled by defining user permissions on the image resources.

The registry is typically used as a publication target for images built on the cluster, as well as being a source of images for workloads running on the cluster. When a new image is pushed to the registry, the cluster is notified of the new image and 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 imagestreams.

Additional resources

- Image Registry Operator in OpenShift Container Platform

1.2. THIRD-PARTY REGISTRIES

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

1.2.1. Authentication

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

1.2.1.1. Registry authentication with Podman

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
following is a generic example of authenticating the registry with Podman.

**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. Login by running the following command and entering your username and password to authenticate:

   ```
   $ 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>
   ```

**1.3. 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 you can install in your own data center or cloud environment. Advanced registry features in Red Hat Quay include geo-replication, image scanning, and the ability to roll back images.

Visit the [Quay.io](https://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 like any remote container image registry.

**Additional resources**

- [Red Hat Quay product documentation](#)

**1.4. 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. Following the move to the new registry, the existing registry will be available for a period of time.

**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 it is possible to use this authentication method with OpenShift Container Platform, it is not supported for production deployments. Restrict this authentication method to stand-alone 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 imagestreams 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 imagestreams 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**
2.1. IMAGE REGISTRY ON CLOUD PLATFORMS AND OPENSTACK

The Image Registry Operator installs a single instance of the OpenShift Container Platform 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, GCP, Azure, or OpenStack.

When you install or upgrade an installer-provisioned infrastructure cluster on AWS or Azure, 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, the Operator creates a default `configs.imageregistry.operator.openshift.io` resource instance based on configuration detected in the cluster.

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

The Image Registry Operator runs in the `openshift-image-registry` namespace, and manages the registry instance in that location as well. All configuration and workload resources for the registry reside in that namespace.

**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 it only prunes image metadata in etcd.
- **Unmanaged**: the `--prune-registry` flag for the image pruner is set to `false`.

2.2. IMAGE REGISTRY ON BARE METAL 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.

**NOTE**

The Prometheus console provides an ImageRegistryRemoved alert, for example:

"Image Registry has been removed. ImageStreamTags, BuildConfigs and DeploymentConfigs which reference ImageStreamTags may not work as expected. Please configure storage and update the config to Managed state by editing configs.imageregistry.operator.openshift.io."

### 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.

The Image Registry Operator defaults to the following when deployed with a zone-related topology constraint:

**Image Registry Operator deployed with a zone related topology constraint**

```yaml
- 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
```

The Image Registry Operator defaults to the following when deployed without a zone-related topology constraint, which applies to bare metal and vSphere instances:

**Image Registry Operator deployed without a zone related topology constraint**

```yaml
- labelSelector:
```
A cluster administrator can override the default topology Spread Constraints by configuring the configs.imageregistry.operator.openshift.io/cluster spec file. In that case, only the constraints you provide apply.

### 2.4. ADDITIONAL RESOURCES

- Configuring pod topology spread constraints

### 2.5. IMAGE REGISTRY OPERATOR CONFIGURATION PARAMETERS

The configs.imageregistry.operator.openshift.io resource offers the following configuration parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| managementState             | **Managed**: The Operator updates the registry as configuration resources are updated.  
                              | **Unmanaged**: The Operator ignores changes to the configuration resources.      
<pre><code>                          | **Removed**: The Operator removes the registry instance and tear down any storage that the Operator provisioned. |
</code></pre>
<p>| logLevel                    | Sets logLevel of the registry instance. Defaults to Normal.                  |
| httpSecret                  | Value needed by the registry to secure uploads, generated by default.         |
| proxy                       | Defines the Proxy to be used when calling master API and upstream registries. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>storage</strong></td>
<td><strong>Storagetype</strong>: Details for configuring registry storage, for example S3 bucket coordinates. Normally configured by default.</td>
</tr>
<tr>
<td><strong>readOnly</strong></td>
<td>Indicates whether the registry instance should reject attempts to push new images or delete existing ones.</td>
</tr>
<tr>
<td><strong>requests</strong></td>
<td>API Request Limit details. Controls how many parallel requests a given registry instance will handle before queuing additional requests.</td>
</tr>
<tr>
<td><strong>defaultRoute</strong></td>
<td>Determines whether or not an external route is defined using the default hostname. If enabled, the route uses re-encrypt encryption. Defaults to <strong>false</strong>.</td>
</tr>
<tr>
<td><strong>routes</strong></td>
<td>Array of additional routes to create. You provide the hostname and certificate for the route.</td>
</tr>
<tr>
<td><strong>replicas</strong></td>
<td>Replica count for the registry.</td>
</tr>
<tr>
<td><strong>disableRedirect</strong></td>
<td>Controls whether to route all data through the registry, rather than redirecting to the back end. Defaults to <strong>false</strong>.</td>
</tr>
<tr>
<td><strong>spec.storage.managementState</strong></td>
<td>The Image Registry Operator sets the <strong>spec.storage.managementState</strong> parameter to <strong>Managed</strong> on new installations or upgrades of clusters using installer-provisioned infrastructure on AWS or Azure.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Managed</strong>: Determines that the Image Registry Operator manages underlying storage. If the Image Registry Operator's <strong>managementState</strong> is set to <strong>Removed</strong>, then the storage is deleted.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>managementState</strong> is set to <strong>Managed</strong>, the Image Registry Operator attempts to apply some default configuration on the underlying storage unit. For example, if set to <strong>Managed</strong>, 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 <strong>managementState</strong> is set to <strong>Unmanaged</strong>.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Unmanaged</strong>: Determines that the Image Registry Operator ignores the storage settings. If the Image Registry Operator’s <strong>managementState</strong> is set to <strong>Removed</strong>, then the storage is not deleted. If you provided an underlying storage unit configuration, such as a bucket or container name, and the <strong>spec.storage.managementState</strong> is not yet set to any value, then the Image Registry Operator configures it to <strong>Unmanaged</strong>.</td>
</tr>
</tbody>
</table>

### 2.6. ENABLE THE IMAGE REGISTRY DEFAULT ROUTE WITH THE CUSTOM RESOURCE DEFINITION

In OpenShift Container Platform, the **Registry** Operator controls the registry feature. The Operator is defined by 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:

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

2.7. CONFIGURING ADDITIONAL TRUST STORES FOR IMAGE REGISTRY ACCESS

The `image.config.openshift.io/cluster` custom resource can contain a reference to a config map that contains additional certificate authorities to be trusted during image registry access.

Prerequisites

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

Procedure

You can create a config map in the `openshift-config` namespace and use its name in `AdditionalTrustedCA` in the `image.config.openshift.io` custom resource to provide additional CAs that should be trusted when contacting external registries.

The config map key is the hostname of a registry with the port for which this CA is to be trusted, and the base64-encoded certificate is the value, for each additional registry CA to trust.

Image registry CA config map example

```yaml
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-----
```

1 If the registry has the port, such as `registry-with-port.example.com:5000`, : should be replaced with ..

You can configure additional CAs with the following procedure.

1. To configure an additional CA:
2.8. CONFIGURING STORAGE CREDENTIALS FOR THE IMAGE REGISTRY OPERATOR

In addition to the `configs.imageregistry.operator.openshift.io` and ConfigMap resources, storage credential 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.

**Procedure**

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

```
$ oc create secret generic image-registry-private-configuration-user --from-file=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 GCP user-provisioned infrastructure
- Configuring the registry for Azure user-provisioned infrastructure
- Configuring the registry for bare metal
- Configuring the registry for vSphere
CHAPTER 3. SETTING UP AND CONFIGURING THE REGISTRY

3.1. Configuring the Registry for AWS User-Provisioned Infrastructure

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.

  
  ```bash
  $ 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.

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

Use the following procedure if the Registry Operator cannot create an S3 bucket and automatically configure storage.

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**

```
storage:
  s3:
    bucket: <bucket-name>
    region: <region-name>
```

---

**WARNING**

To secure your registry images in AWS, block public access to the S3 bucket.

---

### 3.1.3. Image Registry Operator configuration parameters for AWS S3

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

`ImageRegistryConfigStorageS3` 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bucket</strong></td>
<td>Bucket is the bucket name in which you want to store the registry's data. It is optional and is generated if not provided.</td>
</tr>
<tr>
<td><strong>region</strong></td>
<td>Region is the AWS region in which your bucket exists. It is optional and is set based on the installed AWS Region.</td>
</tr>
<tr>
<td><strong>regionEndpoint</strong></td>
<td>RegionEndpoint is the endpoint for S3 compatible storage services. It is optional and defaults based on the Region that is provided.</td>
</tr>
<tr>
<td><strong>virtualHostedStyle</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>encrypt</strong></td>
<td>Encrypt specifies whether or not the registry stores the image in encrypted format. It is optional and defaults to false.</td>
</tr>
<tr>
<td><strong>keyID</strong></td>
<td>KeyID is the KMS key ID to use for encryption. It is optional. Encrypt must be true, or this parameter is ignored.</td>
</tr>
<tr>
<td><strong>ImageRegistryConfigStorageS3CloudFront</strong></td>
<td>CloudFront configures Amazon Cloudfront as the storage middleware in a registry. It is optional.</td>
</tr>
</tbody>
</table>
3.2. CONFIGURING THE REGISTRY FOR GCP USER-PROVISIONED INFRASTRUCTURE

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

- `REGISTRY_STORAGE_GCS_KEYFILE`

**Procedure**

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

  ```sh
  $ 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. Registry storage for GCP with user-provisioned infrastructure

You must set up the storage medium manually and configure the settings in the registry custom resource (CR).

**Prerequisites**

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

3.2.3. Image Registry Operator configuration parameters for GCP GCS

**Procedure**

- regionEndpoint: http://rook-ceph-rgw-ocs-storagecluster-cephobjectstore.openshift-storage.svc.cluster.local
The following configuration parameters are available for GCP GCS registry storage.

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

### 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 the Image Registry Operator to trust Swift storage

You must configure the Image Registry Operator to trust Red Hat OpenStack Platform (RHOSP) Swift storage.

**Procedure**

- From a command line, enter the following command to change the value of the `spec.disableRedirect` field in the `config.imageregistry` object to `true`:

  ```bash
  $ 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_USER`
• **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_USER=<username> --from-literal=REGISTRY_STORAGE_SWIFT_PASSWORD=<password> -n openshift-image-registry
  ```

**3.3.3. Registry storage for RHOSP with user-provisioned infrastructure**

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_USER**
  
  - **REGISTRY_STORAGE_SWIFT_PASSWORD**

**3.3.4. Image Registry Operator configuration parameters for RHOSP Swift**

The following configuration parameters are available for Red Hat OpenStack Platform (RHOSP) Swift registry storage.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authURL</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>authVersion</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>container</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>domain</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>domainID</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>tenant</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>tenantID</td>
<td>This value is optional.</td>
</tr>
<tr>
<td>regionName</td>
<td>This value is optional.</td>
</tr>
</tbody>
</table>
3.4. CONFIGURING THE REGISTRY FOR AZURE USER-PROVISIONED INFRASTRUCTURE

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.

  ```bash
  $ 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`:

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

   Example configuration

   ```yaml
   storage:
   azure:
   ```
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**

   ```yaml
   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

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:

   ```yaml
   apiVersion: storage.k8s.io/v1
   ```
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>
```

1. Enter the namespace `openshift-image-registry`. This namespace allows the Cluster Image Registry Operator to consume the PVC.

2. Optional: Adjust the volume size.

3. Enter the name of the storage class that you created.

4. From a command line, apply the configuration:

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

persistentvolumeclaim/csi-pvc-imageregistry created

5. 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:

1. 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
   ...

2. 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
   STORAGETYPE            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

**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.

NOTE

The Prometheus console provides an ImageRegistryRemoved alert, for example:

"Image Registry has been removed. ImageStreamTags, BuildConfigs and DeploymentConfigs which reference ImageStreamTags may not work as expected. Please configure storage and update the config to Managed state by editing configs.imageregistry.operator.openshift.io."

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:

  ```bash
  $ 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 using 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
   ```

   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
   ```
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

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 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.

Procedure

1. 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:

   ```bash
   $ 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.

3. Edit the registry configuration so that it references the correct 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 internal image registry:

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

This document outlines the 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:

   ```bash
   cat <<EOF | oc apply -f -
   apiVersion: objectbucket.io/v1alpha1
   kind: ObjectBucketClaim
   metadata:
     name: rgwtest
     namespace: openshift-storage
   spec:
   ```
2. Get the bucket name by entering the following command:

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

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

   $ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwtest -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 rgwtest -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. Create a encryption route for Ceph RGW by entering the following command:

   $ oc create route reencrypt <route_name> --service=rook-ceph-rgw-ocs-storagecluster-cephobjectstore --port=https -n openshift-storage

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

      $ route_host=$(oc get route <route_name> -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/router-certs-default -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":true,"trustedCA":{"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 internal image registry:

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

This document outlines 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
   
   $ bucket_name=$(oc get obc -n openshift-storage noobaatest -o jsonpath='{.spec.bucketName}')
   
   $ 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)
   
   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:
5. Get the route host by entering the following command:

```bash
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:

```bash
oc extract secret/router-certs-default -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:

```bash
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":true,"trustedCA":{"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 internal image registry:

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

This document outlines the 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 are 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:

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

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

```bash
$ 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

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.

**NOTE**

The Prometheus console provides an ImageRegistryRemoved alert, for example:

"Image Registry has been removed. ImageStreamTags, BuildConfigs and DeploymentConfigs which reference ImageStreamTags may not work as expected. Please configure storage and update the config to Managed state by editing configs.imageregistry.operator.openshift.io."

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

   **NOTE**
   
   When using shared storage, review your security settings to prevent outside access.

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

   ```bash
   $ 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:

   ```bash
   $ oc edit configs.imageregistry.operator.openshift.io
   
   Example output
   
   storage:
   pvc:
     claim: 1
   
   Leave the `claim` field blank to allow the automatic creation of an `image-registry-storage` PVC.

4. Check the `clusteroperator` status:

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

  ```bash
  $ 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

1. 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:

   ```bash
   $ 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.

b. Create a `pvc.yaml` file with the following contents to define a VMware vSphere PersistentVolumeClaim object:

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

1. A unique name that represents the PersistentVolumeClaim object.
2. The namespace for the PersistentVolumeClaim object, which is `openshift-image-registry`.
3. The access mode of the persistent volume claim. With `ReadWriteOnce`, the volume can be mounted with read and write permissions by a single node.
4. The size of the persistent volume claim.

b. Create the PersistentVolumeClaim object from the file:

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

3. 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. Creating a custom PVC allows you to leave the `claim` field blank for the default automatic creation of an `image-registry-storage` PVC.

For instructions about configuring registry storage so that it references the correct PVC, see Configuring the registry for vSphere.

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

For instructions on using the Ceph RGW storage with Red Hat OpenShift Data Foundation, refer to the official documentation.
Red Hat OpenShift Data Foundation integrates multiple storage types that you can use with the internal image registry:

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

This document outlines the 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](https://docs.openshift.com/container-platform/4.11/data-platform/operator-overview.html) 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:

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

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

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

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

```bash
$ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwtest -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 rgwtest -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:

```bash
$ 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. Create a encryption route for Ceph RGW by entering the following command:

```
$ oc create route reencrypt <route_name> --service=rook-ceph-rgw-ocs-storagecluster-cephobjectstore --port=https -n openshift-storage
```

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

```
$route_host=$(oc get route <route_name> -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/router-certs-default -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":true,"trustedCA": "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 internal image registry:

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

This document outlines 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](https://openshift.io/data-foundation) 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:

```yaml
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
  bucketName: noobaatest
EOF
```

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

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

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

```bash
$ 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:

```bash
$ 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:

```bash
$ 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:

```bash
$ oc extract secret/router-certs-default -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:

```bash
$ oc patch config.image/cluster -p "{"spec":
  "managementState":"Managed","replicas":2,"storage":
  {"managementState":"Unmanaged","s3":{"bucket":\"${bucket_name}\"","region":"us-east--
```
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 internal image registry:

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

This document outlines the 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 are 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:

```yaml
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:
3.7.5. Additional resources

- Recommended configurable storage technology
- Configuring Image Registry to use OpenShift Data Foundation

3.8. CONFIGURING THE REGISTRY FOR RED HAT OPENSSHIFT DATA FOUNDATION

To configure the internal 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 internal image registry:

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

This document outlines the 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: rgwtest
  namespace: openshift-storage
spec:
```

```bash
$ oc patch config.image/cluster -p '":{"spec":
  {"managementState":"Managed","replicas":2,"storage":
  {"managementState":"Unmanaged","pvc":{"claim":"registry-storage-pvc"}}}}' --type=merge
```
2. Get the bucket name by entering the following command:

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

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

   ```
   $ AWS_ACCESS_KEY_ID=$(oc get secret -n openshift-storage rgwtest -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 rgwtest -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. Create a encryption route for Ceph RGW by entering the following command:

   ```
   $ oc create route reencrypt <route_name> --service=rook-ceph-rgw-ocs-storagecluster-cephobjectstore --port=https -n openshift-storage
   
a. Get the route host by entering the following command:

   ```
   $ route_host=$(oc get route <route_name> -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/router-certs-default -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":true,"trustedCA":{"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 internal image registry:

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

This document outlines 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:

   ```bash
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
```

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

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

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

   ```bash
   $ 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:

   ```bash
   ```
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/router-certs-default -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 Noobaa 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":true,"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 internal image registry:

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

This document outlines the 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 are 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](https://openshift.io/) to provide object storage and CephFS file storage.
Procedure

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

```bash
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:

```bash
$ 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)](#)
CHAPTER 4. ACCESSING THE REGISTRY

Use the following sections for instructions on accessing the registry, including viewing logs and metrics, as well as securing and exposing the registry.

You can access the registry directly to invoke podman commands. This allows you to push images to or pull them from the integrated registry directly using operations like podman push or podman pull. To do so, you must be logged in to the registry using the podman login command. The operations you can perform depend on your user permissions, as described in the following sections.

4.1. PREREQUISITES

- 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, for example when using the podman push command:
  
  - The user must have 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 REGISTRY DIRECTLY FROM THE CLUSTER

You can access the registry from inside the cluster.

Procedure

Access the registry from the cluster by using internal routes:

1. Access the node by getting the node’s name:

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

2. To enable access to tools such as oc and podman on the node, run the following command:

   ```
   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
   ```
You should see a message confirming login, such as:

- Login Succeeded!

**NOTE**

You can pass any value for the user name; the token contains all necessary information. Passing a user name that contains colons will result in a login failure.

Since the Image Registry Operator creates the route, it will likely be similar to `default-route-openshift-image-registry.<cluster_name>`.

4. Perform `podman pull` and `podman push` operations against your registry:

**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.

In the following examples, use:

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;registry_ip&gt;</code></td>
<td>172.30.124.220</td>
</tr>
<tr>
<td><code>&lt;port&gt;</code></td>
<td>5000</td>
</tr>
<tr>
<td><code>&lt;project&gt;</code></td>
<td>openshift</td>
</tr>
<tr>
<td><code>&lt;image&gt;</code></td>
<td>image</td>
</tr>
<tr>
<td><code>&lt;tag&gt;</code></td>
<td>omitted (defaults to <code>latest</code>)</td>
</tr>
</tbody>
</table>

a. Pull an arbitrary image:

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

b. Tag the new image with the form `<registry_ip>:<port>/<project>/<image>`. The project name must appear in this 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
```

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

1. List the pods in the openshift-image-registry project and view their status:

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

Example output

```
NAME READY STATUS RESTARTS AGE
cluster-image-registry-operator-764bd7f846-qqtpb 1/1 Running 0 78m
image-registry-79fb4469f6-lrln 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

1. Use the oc logs command with deployments to view the logs for the container image registry:

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

Example output

```
```
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 are exposed at the /extensions/v2/metrics path of the registry endpoint.

Procedure

There are two ways in which you can access the metrics, running a metrics query or using the cluster role.

Metrics query

1. Run a metrics query, for example:

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

Example output

```text
# 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{method="get"} 12.260727916000022
```
<user> can be arbitrary, but <secret> must match the value specified in the registry configuration.

Cluster role

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
   
   $ oc adm policy add-cluster-role-to-user prometheus-scraper <username>
   
   2. Add this role to a user, run the following command:

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

3. Access the metrics using cluster role. The part of the configuration file responsible for metrics should look like this:

   ```
   openshift:
   version: 1.0
   metrics:
     enabled: true
   ...
   ```

4.6. ADDITIONAL RESOURCES

- For more information on allowing pods in a project to reference images in another project, see [Allowing pods to reference images across projects](#).

- A **kubeadmin** can access the registry until deleted. See [Removing the kubeadmin user](#) for more information.

- For more information on configuring an identity provider, see [Understanding identity provider configuration](#).
CHAPTER 5. EXPOSING THE REGISTRY

By default, the OpenShift Container Platform registry is secured during cluster installation so that it serves traffic through TLS. 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 Container Platform registry from within the cluster, you can gain external access to it by exposing it with a route. This external access enables you to log in to the registry from outside the cluster using the route address and to 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.

Procedure

You can expose the route by using the `defaultRoute` parameter in the `configs.imageregistry.operator.openshift.io` resource.

To expose the registry using the `defaultRoute`:

1. Set `defaultRoute` to `true`:

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

2. Get the default registry route:

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

3. Get the certificate of the Ingress Operator:

   ```bash
   $ oc get secret -n openshift-ingress router-certs-default -o go-template='{{index .data
   "tls.crt"}}' | base64 -d | sudo tee /etc/pki/ca-trust/source/anchors/$HOST.crt > /dev/null
   ```

4. Enable the cluster’s default certificate to trust the route using the following commands:

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

5. Log in with podman using the default route:

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

5.2. EXPOSING A SECURE REGISTRY MANUALLY

Instead of logging in to the OpenShift Container Platform registry from within the cluster, you can gain
external access to it by exposing it with a route. This allows you to log in to the registry from outside the cluster using the route address, and to 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.

Procedure

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

To expose the registry using DefaultRoute:

1. Set DefaultRoute to True:

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

2. Log in with podman:

   ```
   $ 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 is needed if the cluster’s default certificate for routes is untrusted. You can set a custom, trusted certificate as the default certificate with the Ingress Operator.

To expose the registry using custom routes:

1. Create a secret with your route’s TLS keys:

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

   This step is optional. If you do not create a secret, the route uses the default TLS configuration from the Ingress Operator.

2. On the Registry Operator:

   ```
   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 registry's route.