Red Hat OpenShift Service on AWS 4

Registry

Red Hat OpenShift Service on AWS can build images from your source code, deploy them, and manage their lifecycle.

Last Updated: 2023-05-25

Red Hat OpenShift Service on AWS 4 Registry

Red Hat OpenShift Service on AWS can build images from your source code, deploy them, and manage their lifecycle.

Legal Notice

Abstract

Red Hat OpensShift Service on AWS provides an internal, integrated container image registry that can be deployed in your Red Hat OpenShift Service on AWS environment to locally manage images.

Table of Contents

CHAPTER 1. OPENSHIFT IMAGE REGISTRY OVERVIEW

Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS environment to locally manage images. This overview contains reference information and links for registries commonly used with Red Hat OpenShift Service on AWS, with a focus on the OpenShift image registry.

1.1. GLOSSARY OF COMMON TERMS FOR OPENSHIFT IMAGE REGISTRY

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

container

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

Image Registry Operator

The Image Registry Operator runs in the **openshift-image-registry** namespace, and manages the registry instance in that location.

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 Red Hat OpenShift Service on AWS images.

namespace

A namespace isolates groups of resources within a single cluster.

pod

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

private registry

A registry is a server that implements the container image registry API. A private registry is a registry that requires authentication to allow users access its contents.

public registry

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

Quay.io

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

OpenShift image registry

OpenShift image registry is the registry provided by Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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 OPENSHIFT IMAGE REGISTRY

Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS

1.3. THIRD-PARTY REGISTRIES

Red Hat OpenShift Service on AWS 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 image registry. In this situation, Red Hat OpenShift Service on AWS 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.

1.3.1. Authentication

Red Hat OpenShift Service on AWS can communicate with registries to access private image repositories using credentials supplied by the user. This allows Red Hat OpenShift Service on AWS to push and pull images to and from private repositories.

1.3.1.1. 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. Log in 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>
```

1.4. 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 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 Red Hat OpenShift Service on AWS like any remote container image registry.

Additional resources

- Red Hat Quay product documentation

1.5. 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 Red Hat OpenShift Service on AWS. Following the move to the new registry, the existing registry will be available for a period of time.

NOTE

Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS, it is not supported for production deployments. Restrict this authentication method to stand-alone projects outside Red Hat OpenShift Service on AWS.

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

CHAPTER 2. IMAGE REGISTRY OPERATOR IN RED HAT OPENSHIFT SERVICE ON AWS

2.1. IMAGE REGISTRY ON RED HAT OPENSHIFT SERVICE ON AWS

The Image Registry Operator installs a single instance of the OpenShift image registry, and manages all registry configuration, including setting up registry 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 metatdata in etcd.
- **Unmanaged**: the **--prune-registry** flag for the image pruner is set to **false**.

2.2. 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

```
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
```

A cluster administrator can override the default **topologySpreadConstraints** by configuring the **configs.imageregistry.operator.openshift.io/cluster** spec file. In that case, only the constraints you provide apply.

2.3. IMAGE REGISTRY OPERATOR CONFIGURATION PARAMETERS

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

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 supported values for **logLevel** are: • **Normal** • **Debug** • **Trace** • **TraceAll**
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.
storage	**Storagetype**: Details for configuring registry storage, for example S3 bucket coordinates. Normally configured by default.

Parameter	Description
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**.
spec.storage.managementState	The Image Registry Operator sets the **spec.storage.managementState** parameter to **Managed** on new installations or upgrades of clusters on AWS. • **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**.

2.4. ENABLE THE IMAGE REGISTRY DEFAULT ROUTE WITH THE CUSTOM RESOURCE DEFINITION

In Red Hat OpenShift Service on AWS, the **Registry** Operator controls the OpenShift image 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:

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

2.5. 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 PEM certificate content is the value, for each additional registry CA to trust.

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: | 1
    -----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:

```
$ 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
```

2.6. 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 Red Hat OpenShift Service on AWS 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
```

CHAPTER 3. 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.

3.1. 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, 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.

3.2. ACCESSING THE 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, change your root directory to **/host**:

```
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:

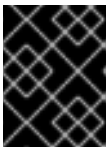
```
Login Succeeded!
```

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:

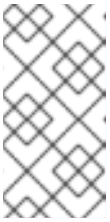
Component	Value
<registry_ip>	**172.30.124.220**
<port>	**5000**
<project>	**openshift**
<image>	**image**
<tag>	omitted (defaults to **latest**)

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 Red Hat OpenShift Service on AWS 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>
```

3.3. 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-llrln 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
```

3.4. 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

```
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
```

3.5. 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

You can access the metrics by running a metrics query using a cluster role.

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
```

2. Add this role to a user, run the following command:

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

Metrics query

1. Get the user token.

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

2. Run a metrics query in node or inside a pod, for example:

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

Example output

```
# 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
```

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

CHAPTER 4. EXPOSING THE REGISTRY

By default, the OpenShift image registry is secured during cluster installation so that it serves traffic through TLS. Unlike previous versions of Red Hat OpenShift Service on AWS, the registry is not exposed outside of the cluster at the time of installation.

4.1. 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 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.
- You have access to the cluster as a user with the **cluster-admin** role.

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**:

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

2. Get the default registry route:

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

3. Get the certificate of the Ingress Operator:

```
$ 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:

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

5. Log in with podman using the default route:

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

4.2. EXPOSING A SECURE REGISTRY MANUALLY

Instead of logging in to the OpenShift image registry from within the cluster, you can gain external access to 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.
- You have access to the cluster as a user with the **cluster-admin** role.

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 (1)
```

(1) **--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:

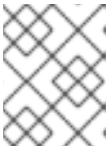
```
$ oc create secret tls public-route-tls \
    -n openshift-image-registry \
    --cert=</path/to/tls.crt> \
    --key=</path/to/tls.key>
```

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.