Chapter 7.  Manage Application Updates

[**Container Image Identity and Tags**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: Container Image Identity and Tags**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s02/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Update Application Image and Settings**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s03/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: Update Application Image and Settings**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s04/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Reproducible Deployments with OpenShift Image Streams**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s05/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: Reproducible Deployments with OpenShift Image Streams**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s06/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Automatic Image Updates with OpenShift Image Change Triggers**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s07/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Guided Exercise: Automatic Image Updates with OpenShift Image Change Triggers**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s08/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Lab: Manage Application Updates**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s09/c2358540-87d5-48de-b49e-6f23bdcd629c)

[**Summary**](https://rha.ole.redhat.com/rha/app/courses/do180-4.14/pages/ch07s10/c2358540-87d5-48de-b49e-6f23bdcd629c)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Manage reproducible application updates and rollbacks of code and configurations. |
| **Objectives** | * Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes. * Update applications with minimal downtime by using deployment strategies. * Ensure reproducibility of application deployments by using image streams and short image names. * Ensure automatic update of application pods by using image streams with Kubernetes workload resources. |
| **Sections** | * Container Image Identity and Tags (and Guided Exercise) * Update Application Image and Settings (and Guided Exercise) * Reproducible Deployments with OpenShift Image Streams (and Guided Exercise) * Automatic Image Updates with OpenShift Image Change Triggers (and Guided Exercise) |
| **Lab** | * Manage Application Updates |

Container Image Identity and Tags

Objectives

* Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes.

Kubernetes Image Tags

The full name of a container image is composed of several parts. For example, you can decompose the registry.access.redhat.com/ubi9/nginx-120:1-86 image name into the following elements:

* The registry server is registry.access.redhat.com.
* The namespace is ubi9.
* The name is nginx-120. In this example, the name of the image includes the version of the software, Nginx version 1.20.
* The tag, which points to a specific version of the image, is 1-86. If you omit the tag, then most container tools use the latest tag by default.

Multiple tags can refer to the same image version. The following screen capture of the Red Hat Ecosystem Catalog at https://catalog.redhat.com/software/containers/explore lists the tags for the ubi9/nginx-120 image:

A screenshot of a website

Description automatically generated

In this case, the 1.86, latest, and 1 tags point to the same image version. You can use any of these tags to refer to that version.

The latest and 1 tags are *floating tags*, because they can point to different image versions over time. For example, when developers publish a new version of the image, they change the latest tag to point to that new version. They also update the 1 tag to point to the latest release of that version, such as 1-87 or 1-88.

As a user of the image, by specifying a floating tag, you ensure that you always consume the up-to-date image version that corresponds to the tag.

Floating Tag Issues

Vendors, organizations, and developers who publish images manage their tags and establish their own lifecycle for floating tags. They can reassign a floating tag to a new image version without notice.

As a user of the image, you might not notice that the tag that you were using now points to a different image version.

Suppose that you deploy an application on OpenShift and use the latest tag for the image. The following series of events might occur:

1. When OpenShift deploys the container, it pulls the image with the latest tag from the container registry.
2. Later, the image developer pushes a new version of the image, and reassigns the latest tag to that new version.
3. OpenShift relocates the pod to a different cluster node, for example because the original node fails.
4. On that new node, OpenShift pulls the image with the latest tag, and thereby retrieves the new image version.
5. Now the OpenShift deployment runs with a new version of the application, without your awareness of that version update.

A similar issue is that when you scale up your deployment, OpenShift starts new pods. On the nodes, OpenShift pulls the latest image version for these new pods. As a result, if a new version is available, then your deployment runs with containers that use different versions of the image. Application inconsistencies and unexpected behavior might occur.

To prevent these issues, select an image that is guaranteed not to change over time. You thus gain control over the lifecycle of your application: you can choose when and how OpenShift deploys a new image version.

You can select a static image version in several ways:

* Use a tag that does not change, instead of relying on floating tags.
* Use OpenShift image streams for tight control over the image versions. Another section in this course discusses image streams further.
* Use the *SHA (Secure Hash Algorithm)* image ID instead of a tag when referencing an image version.

The distinction between a floating and non-floating tag is not a technical one, but a convention. Although it is discouraged, there is no mechanism to prevent a developer from pushing a different image to an existing tag. Thus, you must specify the SHA image ID to guarantee that the referenced container image does not change.

Using SHA Image ID

Developers assign tags to images. In contrast, an SHA image ID, or *digest*, is a unique identifier that the container registry computes and assigns to images. The SHA ID is an immutable string that refers to a specific image version. Using the SHA ID for identifying an image is the most secure approach.

To refer to an image by its SHA ID, replace *name*:*tag* with *name*@*SHA-ID* in the image name. The following example uses the SHA image ID instead of a tag.

registry.access.redhat.com/ubi9/nginx-120@**sha256:1be2006abd21735e7684eb4cc6eb62...**

To retrieve the SHA image ID from the tag, use the oc image info command.

**Note**

A multi-architecture image references images for several CPU architectures. Multi-architecture images include an index that points to the images for different platforms and CPU architectures.

For these images, the oc image info command requires you to select an architecture by using the --filter-by-os option:

[user@host ~]$ **oc image info registry.access.redhat.com/ubi9/nginx-120:1-86**

error: the image is a manifest list and contains multiple images - use --filter-by-os to select from:

OS DIGEST

linux/amd64 sha256:1be2006abd21735e7684eb4cc6eb6295346a89411a187e37cd4...

linux/arm64 sha256:d765193e823bb89b878d2d2cb8be0e0073839a6c19073a21485...

linux/ppc64le sha256:0dd0036620f525b3ba9a46f9f1c52ac70414f939446b2ba3a07...

linux/s390x sha256:d8d95cc17764b82b19977bc7ef2f60ff56a3944b3c7c14071dd...

The following example displays the SHA ID for the image that the 1-86 tag currently points to.

[user@host ~]$ **oc image info --filter-by-os linux/amd64 \**

**registry.access.redhat.com/ubi9/nginx-120:1-86**

Name: registry.access.redhat.com/ubi9/nginx-120:1-86

**Digest: sha256:1be2006abd21735e7684eb4cc6eb​6295346a89411a187e37cd4a3aa2f1bd13a5**

Manifest List: sha256:5bc635dc946fedb4ba391470e8f84f9860e06a1709e30206a95ed9955...

Media Type: application/vnd.docker.distribution.manifest.v2+json

*...output omitted...*

You can also use the skopeo inspect command. The output format differs from the oc image info command, although both commands report similar data.

If you use the oc debug node/*node-name* command to connect to a compute node, then you can list the locally available images by running the crictl images --digests --no-trunc command. The --digests option instructs the command to display the SHA image IDs, and the --no-trunc option instructs the command to display the full SHA string; otherwise, the command displays only the first characters.

[user@host ~]$ **oc debug node/*node-name***

Temporary namespace openshift-debug-csn2p is created for debugging node...

Starting pod/***node-name***-debug ...

To use host binaries, run `chroot /host`

Pod IP: 192.168.50.10

If you don't see a command prompt, try pressing enter.

sh-4.4# **chroot /host**

sh-4.4# **crictl images --digests --no-trunc \**

**registry.access.redhat.com/ubi9/nginx-120:1-86**

IMAGE TAG DIGEST IMAGE ID ...

registry.access.redhat.com/ubi9/nginx-120 1-86 **sha256:1be2...13a5** 2e68...949e ...

The IMAGE ID column displays the local image identifier that the container engine assigns to the image. This identifier is not related to the SHA ID.

The container image format relies on SHA-256 hashes to identify several image components, such as the image layers or the image metadata. Because some commands also report these SHA-256 strings, ensure that you use the SHA-256 hash that corresponds to the SHA image ID. Commands often refer to the SHA image ID as the image digest.

Selecting a Pull Policy

When you deploy an application, OpenShift selects a compute node to run the pod. On that node, OpenShift pulls the image and then starts the container.

By setting the imagePullPolicy attribute in the deployment resource, you can control how OpenShift pulls the image.

The following example shows the myapp deployment resource. The pull policy is set to IfNotPresent.

[user@host ~]$ **oc get deployment myapp -o yaml**

apiVersion: apps/v1

kind: Deployment

*...output omitted...*

template:

metadata:

creationTimestamp: null

labels:

app: myapp

spec:

containers:

- image: registry.access.redhat.com/ubi9/nginx-120:1-86

**imagePullPolicy: IfNotPresent**

name: nginx-120

*...output omitted...*

The imagePullPolicy attribute can take the following values:

**IfNotPresent**

If the image is already on the compute node, because another container is using it or because OpenShift pulled the image during a preceding pod run, then OpenShift uses that local image. Otherwise, OpenShift pulls the image from the container registry.

If you use a floating tag in your deployment, and the image with that tag is already on the node, then OpenShift does not pull the image again, even if the floating tag might point to a newer image in the source container registry.

OpenShift sets the imagePullPolicy attribute to IfNotPresent by default when you use a tag or the SHA ID to identify the image.

**Always**

OpenShift always verifies whether an updated version of the image is available on the source container registry. To do so, OpenShift retrieves the SHA ID of the image from the registry. If a local image with that same SHA ID is already on the compute node, then OpenShift uses that image. Otherwise, OpenShift pulls the image.

If you use a floating tag in your deployment, and an image with that tag is already on the node, then OpenShift queries the registry anyway to ensure that the tag still points to the same image version. However, if the developer pushed a new version of the image and updated the floating tag, then OpenShift retrieves that new image version.

OpenShift sets the imagePullPolicy attribute to Always by default when you use the latest tag, or when you do not specify a tag.

**Never**

OpenShift does not pull the image, and expects the image to be already available on the node. Otherwise, the deployment fails.

To use this option, you must prepopulate your compute nodes with the images that you plan to use. You use this mechanism to improve speed or to avoid relying on a container registry for these images.

Pruning Images from Cluster Nodes

When OpenShift deletes a pod from a compute node, it does not remove the associated image. OpenShift can reuse the images without having to pull them again from the remote registry.

Because the images consume disk space on the compute nodes, OpenShift needs to remove, or *prune*, the unused images when disk space becomes sparse. The kubelet process, which runs on the compute nodes, includes a garbage collector that runs every five minutes. If the usage of the file system that stores the images is above 85%, then the garbage collector removes the oldest unused images. Garbage collection stops when the file system usage drops below 80%.

The reference documentation at the end of this lecture includes instructions to adjust these default thresholds.

From a compute node, you can run the crictl imagefsinfo command to retrieve the name of the file system that stores the images:

[user@host ~]$ **oc debug node/*node-name***

Temporary namespace openshift-debug-csn2p is created for debugging node...

Starting pod/***node-name***-debug ...

To use host binaries, run `chroot /host`

Pod IP: 192.168.50.10

If you don't see a command prompt, try pressing enter.

sh-4.4# **chroot /host**

sh-4.4# **crictl imagefsinfo**

{

"status": {

"timestamp": "1674465624446958511",

"fsId": {

**"mountpoint": "/var/lib/containers/storage/overlay-images"**

},

**"usedBytes"**: {

"value": "**1318560**"

},

"inodesUsed": {

"value": "446"

}

}

}

From the preceding command output, the file system that stores the images is /var/lib/containers/storage/overlay-images. The images consume 1318560 bytes of disk space.

From the compute node, you can use the crictl rmi to remove an unused image. However, pruning objects by using the crictl command might interfere with the garbage collector and the kubelet process.

It is recommended that you rely on the garbage collector to prune unused objects, images, and containers from the compute nodes. The garbage collector is configurable to better fulfill custom needs that you might have.

Guided Exercise: Container Image Identity and Tags

Update an application by changing its deployment to reference a newer image tag, and find the hashes of the old and new application images.

**Outcomes**

You should be able to inspect container images, list images of containers that run on compute nodes, and deploy applications by using image tags or SHA IDs.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise. It also creates the updates-ids project and the /home/student/DO180/labs/updates-ids/resources.txt file. The resources.txt file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

[student@workstation ~]$ **lab start updates-ids**

**Instructions**

1. Log in to the OpenShift cluster as the developer user with the developer password. Use the updates-ids project.
   1. Log in to the OpenShift cluster.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

*...output omitted...*

* 1. Set the updates-ids project as the active project.
  2. [student@workstation ~]$ **oc project updates-ids**

*...output omitted...*

1. Inspect the two versions of the registry.ocp4.example.com:8443/ubi8/httpd-24 image from the classroom container registry. The classroom setup copied that image from the Red Hat Ecosystem Catalog. The original image is registry.access.redhat.com/ubi8/httpd-24.
   1. Use the oc image info command to inspect the image version that the 1-209 tag references. Notice the unique SHA ID that identifies the image version.

**Note**

To improve readability, the instructions truncate the SHA-256 strings.

On your system, the commands return the full SHA-256 strings. Also, you must type the full SHA-256 string, to provide such a parameter to a command.

[student@workstation ~]$ **oc image info \**

**registry.ocp4.example.com:8443/ubi8/httpd-24:1-209**

Name: registry.ocp4.example.com:8443/ubi8/httpd-24:**1-209**

Digest: **sha256:b1e3...f876**

*...output omitted...*

* 1. Inspect the image version that the 1-215 tag references. Notice that the SHA ID, or digest, differs from the preceding image version.
  2. [student@workstation ~]$ **oc image info \**
  3. **registry.ocp4.example.com:8443/ubi8/httpd-24:1-215**
  4. Name: registry.ocp4.example.com:8443/ubi8/httpd-24:**1-215**
  5. Digest: **sha256:91ad...fd83**

*...output omitted...*

* 1. For inspecting images, you can also use the skopeo inspect command. The output format differs from the oc image info command, although both commands report similar data.

Log in to the registry as the developer user with the developer password by using the skopeo login command. Then, use the skopeo inspect command to inspect the 1-215 image tag.

[student@workstation ~]$ **skopeo login registry.ocp4.example.com:8443 -u developer**

Password: **developer**

Login Succeeded!

[student@workstation ~]$ **skopeo inspect \**

**docker://registry.ocp4.example.com:8443/ubi8/httpd-24:1-215**

{

"Name": "registry.ocp4.example.com:8443/ubi8/httpd-24",

"Digest": "**sha256:91ad...fd83**",

"RepoTags": [

"1-209",

"1-215"

],

*...output omitted...*

}

The skopeo inspect command also shows other existing image tags.

1. Deploy an application from the image version that the 1-209 tag references.
   1. Use the oc create deployment command to deploy the application. Set the name of the deployment to httpd1.
   2. [student@workstation ~]$ **oc create deployment httpd1 \**
   3. **--image registry.ocp4.example.com:8443/ubi8/httpd-24:1-209**

deployment.apps/httpd1 created

* 1. Wait for the pod to start, and then retrieve the name of the cluster node that runs it. You might have to rerun the command several times for the pod to report a Running status. The name of the pod on your system probably differs.
  2. [student@workstation ~]$ **oc get pods -o wide**
  3. NAME READY STATUS RESTARTS AGE IP NODE ...

httpd1-6dff796d99-pm2x6 1/1 **Running** 0 19s 10.8.0.104 **master01** ...

* 1. Retrieve the name of the container that is running inside the pod. The crictl ps command that you run in a following step takes the container name as an argument.
  2. [student@workstation ~]$ **oc get deployment httpd1 -o wide**
  3. NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS ...

httpd1 1/1 1 1 1m10s **httpd-24** ...

1. Access the cluster node and then retrieve the image that the container is using.
   1. You must log in as the admin user to access the cluster node. Use the redhatocp password.
   2. [student@workstation ~]$ **oc login -u admin -p redhatocp**
   3. Login successful.

*...output omitted...*

* 1. Use the oc debug node command to access the cluster node.
  2. [student@workstation ~]$ **oc debug node/master01**
  3. Temporary namespace openshift-debug-flz4d is created for debugging node...
  4. Starting pod/master01-debug ...
  5. To use host binaries, run `chroot /host`
  6. Pod IP: 192.168.50.10

If you don't see a command prompt, try pressing enter.

* 1. In the remote shell, run the chroot /﻿host command.
  2. sh-4.4# **chroot /host**

sh-4.4#

* 1. Use the crictl ps command to confirm that the httpd-24 container is running. Add the -o yaml option to display the container details in YAML format.
  2. sh-4.4# **crictl ps --name httpd-24 -o yaml**
  3. containers:
  4. - annotations:
  5. *...output omitted...*
  6. image:
  7. annotations: {}
  8. image: registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876
  9. **imageRef: registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876**
  10. labels:
  11. *...output omitted...*

state: **CONTAINER\_RUNNING**

Notice that the command refers to the image by its SHA ID, and not by the tag that you specified when you created the deployment resource.

* 1. Use the crictl images command to list the locally available images on the node. The registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 is in that list, because the local container engine pulled it when you deployed the httpd1 application.

**Note**

The IMAGE ID column displays the local image identifier that the container engine assigns to the image. This identifier is not related to the SHA image ID that the container registry assigned to the image.

Most crictl commands, such as crictl images or crictl rmi, accept a local image identifier instead of the full image name. For example, you can run the crictl images 8ee59251acc93 command as a short version of the crictl images registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 command.

sh-4.4# **crictl images**

IMAGE TAG **IMAGE ID** SIZE

quay.io/openshift-release-dev/ocp-release <none> d52324cb88017 444MB

quay.io/openshift-release-dev/ocp-v4.0-art-dev <none> 22e6e45df32af 468MB

quay.io/openshift-release-dev/ocp-v4.0-art-dev <none> e798432938c49 503MB

quay.io/openshift-release-dev/ocp-v4.0-art-dev <none> 3ca084e53b321 873MB

*...output omitted...*

**registry.ocp4.example.com:8443/ubi8/httpd-24 1-209 8ee59251acc93** 461MB

*...output omitted...*

* 1. The preceding crictl images command does not display the SHA image IDs by default. Rerun the command and add the --digests option to display the SHA IDs. Also add the local image ID to the command to limit the output to the registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 image.

The command reports only the first characters of the SHA image ID. These characters match the SHA ID of the image that the httpd-24 container is using. Therefore, the httpd-24 container is using the expected image.

sh-4.4# **crictl images --digests *8ee59251acc93***

IMAGE TAG DIGEST IMAGE ID ...

registry.ocp4.example.com:8443/ubi8/httpd-24 1-209 **b1e3c572516d1** 8ee59251acc93 ...

* 1. Disconnect from the cluster node.
  2. sh-4.4# **exit**
  3. exit
  4. sh-4.4# **exit**
  5. exit
  6. Removing debug pod ...
  7. Temporary namespace openshift-debug-flz4d was removed.

[student@workstation ~]$

1. Log in as the developer user and then deploy another application by using the SHA ID of the image as the digest.
   1. Log in to the OpenShift cluster as the developer user.
   2. [student@workstation ~]$ **oc login -u developer -p developer**
   3. Login successful.

*...output omitted...*

* 1. Rerun the oc image info command to retrieve the SHA ID of the image version that the 1-209 tag references. Specify the JSON format for the command output. Parse the JSON output with the jq -r command to retrieve the value of the .digest object. Export the SHA ID as the $IMAGE environment variable.
  2. [student@workstation ~]$ **oc image info \**
  3. **registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 -o json | \**
  4. **jq -r .digest**

sha256:b1e3...f876

[student@workstation ~]$ **IMAGE=sha256:b1e3...f876**

* 1. Use the oc create deployment command to deploy the application. Set the name of the deployment to httpd2.
  2. [student@workstation ~]$ **oc create deployment httpd2 \**
  3. **--image registry.ocp4.example.com:8443/ubi8/httpd-24@$IMAGE**

deployment.apps/httpd2 created

* 1. Confirm that the new deployment refers to the image version by its SHA ID.
  2. [student@workstation ~]$ **oc get deployment httpd2 -o wide**
  3. NAME READY ... CONTAINERS IMAGES ...

httpd2 1/1 ... httpd-24 registry.../ubi8/httpd-24@**sha256:b1e3...f876** ...

1. Update the httpd2 application by using a more recent image version.
   1. In the httpd2 deployment, update the httpd-24 container to use the image version that the 1-215 tag references.
   2. [student@workstation ~]$ **oc set image deployment/httpd2 \**
   3. **httpd-24=registry.ocp4.example.com:8443/ubi8/httpd-24:1-215**

deployment.apps/httpd2 image updated

* 1. Confirm that the deployment refers to the new image version.
  2. [student@workstation ~]$ **oc get deployment httpd2 -o wide**
  3. NAME READY ... IMAGES ...

httpd2 1/1 ... **registry.ocp4.example.com:8443/ubi8/httpd-24:1-215** ...

* 1. Confirm that the deployment finished redeploying the pod. You might have to rerun the command several times for the pod to report a Running status. The pod names probably differ on your system.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE
  4. httpd1-6dff796d99-pm2x6 1/1 Running 0 118m

**httpd2-998d9b9b9-5859j** 1/1 **Running** 0 21s

* 1. Inspect the pod to confirm that the container is using the new image. Replace the pod name with your own from the previous step.
  2. [student@workstation ~]$ **oc get pod httpd2-*998d9b9b9-5859j* \**
  3. **-o jsonpath='{.spec.containers[0].image}{"\n"}'**

registry.ocp4.example.com:8443/ubi8/httpd-24:1-215

1. Add the latest tag to the image version that the 1-209 tag already references. Deploy an application from the image with the latest tag.
   1. Use the skopeo login command to log in to the classroom container registry as the developer user. Use developer for the password.
   2. [student@workstation ~]$ **skopeo login -u developer -p developer \**
   3. **registry.ocp4.example.com:8443**

Login Succeeded!

* 1. Use the skopeo copy command to add the latest tag to the image.
  2. [student@workstation ~]$ **skopeo copy \**
  3. **docker://registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 \**
  4. **docker://registry.ocp4.example.com:8443/ubi8/httpd-24:latest**
  5. Getting image source signatures
  6. *...output omitted...*
  7. Writing manifest to image destination

Storing signatures

* 1. Use the oc image info command to confirm that both tags refer to the same image. The two commands report the same SHA image ID, which indicates that the tags point to the same image version.
  2. [student@workstation ~]$ **oc image info \**
  3. **registry.ocp4.example.com:8443/ubi8/httpd-24:1-209**
  4. Name: registry.ocp4.example.com:8443/ubi8/httpd-24:1-209
  5. Digest: **sha256:b1e3...f876**

*...output omitted...*

[student@workstation ~]$ **oc image info \**

**registry.ocp4.example.com:8443/ubi8/httpd-24:latest**

Name: registry.ocp4.example.com:8443/ubi8/httpd-24:latest

Digest: **sha256:b1e3...f876**

*...output omitted...*

* 1. Use the oc create deployment command to deploy another application. Set the name of the deployment to httpd3. To confirm that by default the command selects the latest tag, do not provide the tag part in the image name.
  2. [student@workstation ~]$ **oc create deployment httpd3 \**
  3. **--image registry.ocp4.example.com:8443/ubi8/httpd-24**

deployment.apps/httpd3 created

* 1. Confirm that the pod is running. You might have to rerun the command several times for the pod to report a Running status. The pod names probably differ on your system.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE
  4. httpd1-6dff796d99-pm2x6 1/1 Running 0 150m
  5. httpd2-998d9b9b9-5859j 1/1 Running 0 32m

**httpd3-85b978d758-fvqdr** 1/1 **Running** 0 42s

* 1. Confirm that the pod is using the expected image. Notice that the SHA image ID corresponds to the image that the 1-209 tag references. You retrieved that SHA image ID in a preceding step when you ran the oc image info command.
  2. [student@workstation ~]$ **oc describe pod httpd3-*85b978d758-fvqdr***
  3. *...output omitted...*
  4. Containers:
  5. httpd-24:
  6. Container ID: cri-o://2cee...3a68
  7. Image: registry.ocp4.example.com:8443/ubi8/httpd-24
  8. Image ID: registry.ocp4.example.com:8443/ubi8/httpd-24@**sha256:b1e3...f876**

*...output omitted...*

1. Assign the latest tag to a different image version. This operation simulates a developer who pushes a new version of an image and assigns the latest tag to that new image version.
   1. Use the skopeo copy command to add the latest tag to the image version that the 1-215 tag already references. The command automatically removes the latest tag from the earlier image.
   2. [student@workstation ~]$ **skopeo copy \**
   3. **docker://registry.ocp4.example.com:8443/ubi8/httpd-24:1-215 \**
   4. **docker://registry.ocp4.example.com:8443/ubi8/httpd-24:latest**
   5. Getting image source signatures
   6. *...output omitted...*
   7. Writing manifest to image destination

Storing signatures

* 1. Log out of the classroom container registry.
  2. [student@workstation ~]$ **skopeo logout registry.ocp4.example.com:8443**

Removed login credentials for registry.ocp4.example.com:8443

**Note**

The skopeo logout command logs out of a specified registry server by deleting the cached credentials that are stored in the ${XDG\_RUNTIME\_DIR}/containers/auth.json file.

Red Hat recommends removing cached credentials that are no longer required.

* 1. Even though the latest tag is now referencing a different image version, OpenShift does not redeploy the pods that are running with the previous image version.

Rerun the oc describe pod command to confirm that the pod still uses the preceding image.

[student@workstation ~]$ **oc describe pod httpd3-*85b978d758-fvqdr***

*...output omitted...*

Containers:

httpd-24:

Container ID: cri-o://2cee...3a68

Image: registry.ocp4.example.com:8443/ubi8/httpd-24

Image ID: registry.ocp4.example.com:8443/ubi8/httpd-24@**sha256:b1e3...f876**

*...output omitted...*

1. Scale the httpd3 deployment to two pods.
   1. Use the oc scale command to add a new pod to the deployment.
   2. [student@workstation ~]$ **oc scale deployment/httpd3 --replicas 2**

deployment.apps/httpd3 scaled

* 1. List the pods to confirm that two pods are running for the httpd3 deployment. The pod names probably differ on your system.
  2. [student@workstation ~]$ **oc get pods**
  3. httpd1-6dff796d99-pm2x6 1/1 Running 0 75m
  4. httpd2-998d9b9b9-5859j 1/1 Running 0 30m
  5. **httpd3-85b978d758-f98jh** 1/1 Running 0 54s

**httpd3-85b978d758-fvqdr** 1/1 Running 0 11m

* 1. Retrieve the SHA image ID for the pod that the deployment initially created. The ID did not change. The container is still using the original image version.
  2. [student@workstation ~]$ **oc describe pod httpd3-*85b978d758-fvqdr***
  3. *...output omitted...*
  4. Containers:
  5. httpd-24:
  6. Container ID: cri-o://2cee...3a68
  7. Image: registry.ocp4.example.com:8443/ubi8/httpd-24
  8. Image ID: registry.ocp4.example.com:8443/ubi8/httpd-24@**sha256:b1e3...f876**

*...output omitted...*

* 1. Retrieve the SHA image ID for the additional pod. Notice that the ID is different. The additional pod is using the image that the latest tag is currently referencing.
  2. [student@workstation ~]$ **oc describe pod httpd3-*85b978d758-f98jh***
  3. *...output omitted...*
  4. Containers:
  5. httpd-24:
  6. Container ID: cri-o://d254...c893
  7. Image: registry.ocp4.example.com:8443/ubi8/httpd-24
  8. Image ID: registry.ocp4.example.com:8443/ubi8/httpd-24@**sha256:91ad...fd83**

*...output omitted...*

The state of the deployment is inconsistent. The two replicated pods use a different image version. Consequently, the scaled application might not behave correctly. Red Hat recommends that you use a less volatile tag than latest in production environments, or that you tightly control the tag assignments in your container registry.

Update Application Image and Settings

Objectives

* Update applications with minimal downtime by using deployment strategies.

Application Code, Configuration, and Data

Modern applications loosely couple code, configuration, and data. Configuration files and data are not hard-coded as part of the software. Instead, the software loads the configuration and data from an external source. This externalization enables deploying an application to different environments without requiring a change to the application source code.

OpenShift provides configuration map, secret, and volume resources to store the application configuration and data. The application code is available through container images.

Because OpenShift deploys applications from container images, developers must build a new version of the image when they update the code of their application. Organizations usually use a continuous integration and continuous delivery (CI/CD) pipeline to automatically build the image from the application source code, and then to push the resulting image to a container registry.

You use OpenShift resources, such as configuration maps and secrets, to update the configuration of the application. To control the deployment process of a new image version, you use a Deployment object.

Deployment Strategies

Deploying functional application changes or new versions to users is a significant phase of the CI/CD pipelines, where you add value to the development process.

Introducing application changes carries risks, such as downtime during the deployment, bugs, or reduced application performance. You can reduce or mitigate some risks with testing and validation stages in your pipelines.

Application or service downtime can result in lost business, disruption to other services that depend on yours, and violations of service level agreements, among others. To reduce downtime and minimize risks in deployments, use a *deployment strategy*. A deployment strategy changes or upgrades an application in a way that minimizes the impact of those changes.

In OpenShift, you use Deployment objects to define deployments and deployment strategies. The RollingUpdate and the Recreate strategies are the main OpenShift deployment strategies.

To select the RollingUpdate or Recreate strategies, you set the .spec.strategy.type property of the Deployment object. The following snippet shows a Deployment object that uses the Recreate strategy:

apiVersion: apps/v1

kind: Deployment

metadata:

*...output omitted...*

spec:

progressDeadlineSeconds: 600

replicas: 10

revisionHistoryLimit: 10

selector:

matchLabels:

app: myapp2

**strategy:**

**type: Recreate**

template:

*...output omitted...*

Rolling Update Strategy

The RollingUpdate strategy consists of updating a version of an application in stages. It replaces one instance after another until all instances are replaced.

In this strategy, both versions of the application run simultaneously, and it scales down instances of the previous version only when the new version is ready. The main drawback is that this strategy requires compatibility between the versions in the deployment.

The following graphic shows the deployment of a new version of an application by using the RollingUpdate strategy:

1. Some application instances run a code version that needs updating (v1). OpenShift scales up a new instance with the updated application version (v2). Because the new instance with version v2 is not ready, only the version v1 instances fulfill customer requests.
2. The instance with v2 is ready and accepts customer requests. OpenShift scales down an instance with version v1, and scales up a new instance with version v2. Both versions of the application fulfill customer requests.
3. The new instance with v2 is ready and accepts customer requests. OpenShift scales down the remaining instance with version v1.
4. No instances remain to replace. The application update was successful, and without downtime.

The RollingUpdate strategy supports continuous deployment, and eliminates application downtime during deployments. You can use this strategy if the different versions of your application can run at the same time.

**Note**

The RollingUpdate strategy is the default strategy if you do not specify a strategy on the Deployment objects.

The following snippet shows a Deployment object that uses the RollingUpdate strategy:

apiVersion: apps/v1

kind: Deployment

metadata:

*...output omitted...*

spec:

progressDeadlineSeconds: 600

replicas: 10

revisionHistoryLimit: 10

selector:

matchLabels:

app: myapp2

**strategy:**

**rollingUpdate:**

**maxSurge: 25%**

**maxUnavailable: 50%**

**type: RollingUpdate**

template:

*...output omitted...*

Out of many parameters to configure the RollingUpdate strategy, the preceding snippet shows the maxSurge and maxUnavailable parameters.

During a rolling update, the number of pods for the application varies, because OpenShift starts new pods for the new revision, and removes pods from the previous revision. The maxSurge parameter indicates how many pods OpenShift can create above the normal number of replicas. The maxUnavailable parameter indicates how many pods OpenShift can remove below the normal number of replicas. You can express these parameters as percentages or as a number of pods.

If you do not configure a readiness probe for your deployment, then during a rolling update, OpenShift starts sending client traffic to new pods as soon as they are running. However, the application inside a container might not be immediately ready to accept client requests. The application might have to load files to cache, establish a network connection to a database, or perform initial tasks that might take time to complete. Consequently, OpenShift redirects client requests to a container that is not yet ready, and these requests fail.

Adding a readiness probe to your deployment prevents OpenShift from sending traffic to new pods that are not ready.

Recreate Strategy

In this strategy, all the instances of an application are killed first, and are then replaced with new ones. The major drawback of this strategy is that it causes a downtime in your services. For a period, no application instances are available to fulfill requests.

The following graphic shows the deployment of a new version of an application that uses the Recreate strategy:

1. The application has some instances that run a code version to update (v1).
2. OpenShift scales down the running instances to zero. This action causes application downtime, because no instances are available to fulfill requests.
3. OpenShift scales up new instances with a new version of the application (v2). When the new instances are booting, the downtime continues.
4. The new instances finished booting, and are ready to fulfill requests. This step is the last step of the Recreate strategy, and it resolves the application outage.

You can use this strategy when your application cannot have different simultaneously running code versions. You might also use it to execute data migrations or data transformations before the new code starts. This strategy is not recommended for applications that need high availability, for example, medical systems.

Rolling out Applications

When you update a Deployment object, OpenShift automatically rolls out the application. If you apply several modifications in a row, such as modifying the image version, updating environment variables, and configuring the readiness probe, then OpenShift rolls out the application for each modification.

To prevent these multiple deployments, pause the rollout, apply all your modifications to the Deployment object, and then resume the rollout. OpenShift then performs a single rollout to apply all your modifications:

* Use the oc rollout pause command to pause the rollout of the myapp deployment:

[user@host ~]$ **oc rollout pause deployment/myapp**

* Apply all your modifications to the Deployment object. The following example modifies the image, an environment variable, and the readiness probe.
* [user@host ~]$ **oc set image deployment/myapp \**
* **nginx-120=registry.access.redhat.com/ubi9/nginx-120:1-86**
* [user@host ~]$ **oc set env deployment/myapp NGINX\_LOG\_TO\_VOLUME=1**

[user@host ~]$ **oc set probe deployment/myapp --readiness --get-url http://:8080**

* Resume the rollout:

[user@host ~]$ **oc rollout resume deployment/myapp**

OpenShift rolls out the application to apply all your modifications to the Deployment object.

You can follow a similar process when you create and configure a new deployment:

* Create the deployment, and set the number of replicas to zero. This way, OpenShift does not roll out your application, and no pods are running.
* [user@host ~]$ **oc create deployment myapp2 \**
* **--image registry.access.redhat.com/ubi9/nginx-120:1-86 --replicas 0**
* [user@host ~]$ **oc get deployment/myapp2**
* NAME READY UP-TO-DATE AVAILABLE AGE

myapp2 **0/0 0 0** 9s

* Apply the configuration to the Deployment object. The following example adds a readiness probe.

[user@host ~]$ **oc set probe deployment/myapp2 --readiness --get-url http://:8080**

* Scale up the deployment. OpenShift rolls out the application.
* [user@host ~]$ **oc scale deployment/myapp2 --replicas 10**
* [user@host ~]$ **oc get deployment/myapp2**
* NAME READY UP-TO-DATE AVAILABLE AGE

myapp2 **10/10 10 10** 18s

Monitoring Replica Sets

Whenever OpenShift rolls out an application from a Deployment object, it creates a ReplicaSet object. Replica sets are responsible for creating and monitoring the pods. If a pod fails, then the ReplicaSet object deploys a new one.

To deploy pods, replica sets use the pod template definition from the Deployment object. OpenShift copies the template definition from the Deployment object when it creates the ReplicaSet object.

When you update the Deployment object, OpenShift does not update the existing ReplicaSet object. Instead, it creates another ReplicaSet object with the new pod template definition. Then, OpenShift rolls out the application according to the update strategy.

Thus, several ReplicaSet objects for a deployment can exist at the same time on your system. During a rolling update, the old and the new ReplicaSet objects coexist and coordinate the rollout of the new application version. After the rollout completes, OpenShift keeps the old ReplicaSet object so that you can roll back if the new application version does not operate correctly.

The following graphic shows a Deployment object and two ReplicaSet objects. The old ReplicaSet object for version 1 of the application does not run any pods. The current ReplicaSet object for version 2 of the application manages three replicated pods.

Do not directly change or delete ReplicaSet objects, because OpenShift manages them through the associated Deployment objects. The .spec.revisionHistoryLimit attribute in Deployment objects specifies how many ReplicaSet objects OpenShift keeps. OpenShift automatically deletes the extra ReplicaSet objects. Also, when you delete a Deployment object, OpenShift deletes all the associated ReplicaSet objects.

Run the oc get replicaset command to list the ReplicaSet objects. OpenShift uses the Deployment object name as a prefix for the ReplicaSet objects.

[user@host ~]$ **oc get replicaset**

NAME DESIRED CURRENT READY AGE

myapp2-574968dd59 0 0 0 3m27s

myapp2-76679885b9 10 10 10 22s

myapp2-786cbf9bc8 0 0 0 114s

The preceding output shows three ReplicaSet objects for the myapp2 deployment. Whenever you modified the myapp2 deployment, OpenShift created a ReplicaSet object. The second object in the list is active and monitors 10 pods. The other ReplicaSet objects do not manage any pods. They represent the previous versions of the Deployment object.

During a rolling update, two ReplicaSet objects are active. The old ReplicaSet object is scaling down, and at the same time the new object is scaling up:

[user@host ~]$ **oc get replicaset**

NAME DESIRED CURRENT READY AGE

myapp2-574968dd59 0 0 0 13m

myapp2-5fb5766df5 4 4 2 21s

myapp2-76679885b9 8 8 8 10m

myapp2-786cbf9bc8 0 0 0 11m

|  |  |
| --- | --- |
|  | The new ReplicaSet object already started four pods, but the READY column shows that the readiness probe succeeded for only two pods so far. These two pods are likely to receive client traffic. |
|  | The ReplicaSet object already scaled down from 10 to 8 pods. |

Managing Rollout

Because OpenShift preserves ReplicaSet objects from earlier deployment versions, you can roll back if you notice that the new version of the application does not work.

Use the oc rollout undo command to roll back to the preceding deployment version. The command uses the existing ReplicaSet object for that version to roll back the pods. The command also reverts the Deployment object to the preceding version.

[user@host ~]$ **oc rollout undo deployment/myapp2**

Use the oc rollout status command to control the rollout process:

[user@host ~]$ **oc rollout status deployment/myapp2**

deployment "myapp2" successfully rolled out

If the rollout operation fails, because you specify a wrong container image name or the readiness probe fails, then OpenShift does not automatically roll back your deployment. In this case, run the oc rollout undo command to revert to the preceding working configuration.

In contrast to the Deployment resource, from Kubernetes, the DeploymentConfig resource, from OpenShift, supports an automated rollback. In addition, the oc rollout cancel command works only with DeploymentConfig objects, and supports canceling a stuck rollout operation. Another command, oc rollback, also works only with OpenShift DeploymentConfig resources. Red Hat recommends that you use the oc rollout command instead, which provides similar functionalities.

By default, the oc rollout undo command rolls back to the preceding deployment version. If you need to roll back to an earlier revision, then list the available revisions and add the --to-revision *rev* option to the oc rollout undo command.

* Use the oc rollout history command to list the available revisions:
* [user@host ~]$ **oc rollout history deployment/myapp2**
* deployment.apps/myapp2
* REVISION CHANGE-CAUSE
* 1 <none>
* 3 <none>
* 4 <none>

5 <none>

**Note**

The CHANGE-CAUSE column provides a user-defined message that describes the revision. You can store the message in the kubernetes.io/change-cause deployment annotation after every rollout:

[user@host ~]$ **oc annotate deployment/myapp2 \**

**kubernetes.io/change-cause="Image updated to 1-86"**

deployment.apps/myapp2 annotated

[user@host ~]$ **oc rollout history deployment/myapp2**

deployment.apps/myapp2

REVISION CHANGE-CAUSE

1 <none>

3 <none>

4 <none>

5 Image updated to 1-86

* Add the --revision option to the oc rollout history command for more details about a specific revision:
* [user@host ~]$ **oc rollout history deployment/myapp2 --revision 1**
* deployment.apps/myapp2 with revision #1
* Pod Template:
* Labels: app=myapp2
* **pod-template-hash=574968dd59**
* Containers:
* nginx-120:
* Image: registry.access.redhat.com/ubi9/nginx-120:1-86
* Port: <none>
* Host Port: <none>
* Environment: <none>
* Mounts: <none>

Volumes: <none>

The pod-template-hash attribute is the suffix of the associated ReplicaSet object. You can inspect that ReplicaSet object for more details by using the oc describe replicaset myapp2-574968dd59 command, for example.

* Roll back to a specific revision by adding the --to-revision option to the oc rollout undo command:

[user@host ~]$ **oc rollout undo deployment/myapp2 --to-revision 1**

If you use floating tags to refer to container image versions in deployments, then the resulting image when you roll back a deployment might have changed in the container registry. Thus, the image that you run after the rollback might not be the original one that you used.

To prevent this issue, use OpenShift image streams for referencing images instead of floating tags. Another section in this course discusses image streams further.

Reproducible Deployments with OpenShift Image Streams

Objectives

* Ensure reproducibility of application deployments by using image streams and short image names.

Image Streams

*Image streams* are one of the main differentiators between OpenShift and upstream Kubernetes. Kubernetes resources reference container images directly, but OpenShift resources, such as deployment configurations and build configurations, reference image streams. OpenShift also extends Kubernetes resources, such as Kubernetes Deployments, with annotations that make them work with OpenShift image streams.

With image streams, OpenShift can ensure reproducible, stable deployments of containerized applications and also rollbacks of deployments to their latest known-good state.

Image streams provide a stable, short name to reference a container image that is independent of any registry server and container runtime configuration.

As an example, an organization could start by downloading container images directly from the Red Hat public registry and later set up an enterprise registry as a mirror of those images to save bandwidth. OpenShift users would not notice any change, because they still refer to these images by using the same image stream name. Users of the RHEL container tools would notice the change, because they would be required either to change the registry names in their commands, or to change their container engine configurations to search for the local mirror first.

In other scenarios, the indirection that an image stream provides can be helpful. Suppose that you start with a database container image that has security issues, and the vendor takes too long to update the image with fixes. Later, you find an alternative vendor who provides an alternative container image for the same database, with those security issues already fixed, and even better, with a track record of providing timely updates to them. If those container images are compatible regarding configuration of environment variables and volumes, you could change your image stream to point to the image from the alternative vendor.

Red Hat provides hardened, supported container images that work mostly as drop-in replacements of container images from some popular open source projects, such as the MariaDB database.

Image Stream Tags

An image stream represents one or more sets of container images. Each set, or stream, is identified by an *image stream tag*. Unlike container images in a registry server, which have multiple tags from the same image repository (or user or organization), an image stream can have multiple image stream tags that reference container images from different registry servers and from different image repositories.

An image stream provides default configurations for a set of image stream tags. Each image stream tag references one stream of container images, and can override most configurations from its associated image stream.

An image stream tag stores a copy of the metadata about its current container image. Storing metadata supports faster search and inspection of container images, because you do not need to reach its source registry server.

You can also configure an image stream tag to store the source image layers in the OpenShift internal container registry, which acts as a local image cache. Storing image layers locally avoids the need to fetch these layers from their source registry server. Consumers of the cached image, such as pods and deployments, just reference the internal registry as the source registry of the image.

For some other OpenShift resource types that relate to image streams, you can usually dismiss them as implementation details of the internal registry, and focus only on image streams and image stream tags.

To better visualize the relationship between image streams and image stream tags, you can explore the openshift project that is pre-created in all OpenShift clusters. You can see many image streams in that project, including the php image stream:

[user@host ~]$ **oc get is -n openshift -o name**

*...output omitted...*

imagestream.image.openshift.io/nodejs

imagestream.image.openshift.io/perl

**imagestream.image.openshift.io/php**

imagestream.image.openshift.io/postgresql

imagestream.image.openshift.io/python

*...output omitted...*

Several tags exist for the php image stream, and an image stream tag resource exists for each tag:

[user@host ~]$ **oc get istag -n openshift | grep php**

8.0-ubi9 image-registry ... 6 days ago

8.0-ubi8 image-registry ... 6 days ago

7.4-ubi8 image-registry ... 6 days ago

7.3-ubi7 image-registry ... 6 days ago

The oc describe command on an image stream shows information from both the image stream and its image stream tags:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

Tags: 5

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

**8.0-ubi8** (latest)

tagged from registry.access.redhat.com/ubi8/php-80:latest

*...output omitted...*

**7.4-ubi8**

tagged from registry.access.redhat.com/ubi8/php-74:latest

*...output omitted...*

**7.3-ubi7**

tagged from registry.access.redhat.com/ubi7/php-73:latest

*...output omitted...*

In the previous example, each of the php image stream tags refers to a different image name.

Image Names, Tags, and IDs

The textual name of a container image is a string. This name is sometimes interpreted as being made of multiple components, such as registry-host-name/repository-or-organization-or-user-name/image-name:tag-name, but splitting the image name into its components is a matter of convention, not of structure.

A SHA image ID is a SHA-256 hash that uniquely identifies an immutable container image. You cannot modify a container image. Instead, you create a container image that has a new ID. When you push a new container image to a registry server, the server associates the existing textual name with the new image ID.

When you start a container from an image name, you download the image that is currently associated with that image name. The image ID behind that name might change at any moment, and the next container that you start might have a different image ID. If the image that is associated with an image name has any issues, and you know only the image name, then you cannot roll back to an earlier image.

OpenShift image stream tags keep a history of the latest image IDs that they fetched from a registry server. The history of image IDs is the stream of images from an image stream tag. You can use the history inside an image stream tag to roll back to a previous image, if for example a new container image causes a deployment error.

Updating a container image in an external registry does not automatically update an image stream tag. The image stream tag keeps the reference to the last image ID that it fetched. This behavior is crucial to scaling applications, because it isolates OpenShift from changes that happen at a registry server.

Suppose that you deploy an application from an external registry, and after a few days of testing with a few users, you decide to scale its deployment to enable a larger user population. In the meantime, your vendor updates the container image on the external registry. If OpenShift had no image stream tags, then the new pods would get the new container image, which is different from the image on the original pod. Depending on the changes, this new image could cause your application to fail. Because OpenShift stores the image ID of the original image in an image stream tag, it can create new pods by using the same image ID and avoid any incompatibility between the original and updated image.

OpenShift keeps the image ID of the first pod, and ensures that new pods use the same image ID. OpenShift ensures that all pods use the same image.

To better visualize the relationship between an image stream, an image stream tag, an image name, and an image ID, refer to the following oc describe is command, which shows the source image and current image ID for each image stream tag:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

\* registry.access.redhat.com/ubi9/php-80@sha256:**2b82...f544**

2 days ago

**8.0-ubi8** (latest)

tagged from registry.access.redhat.com/ubi8/php-80:latest

\* registry.access.redhat.com/ubi8/php-80@sha256:**2c74...5ef4**

2 days ago

*...output omitted...*

If your OpenShift cluster administrator already updated the php:8.0-ubi9 image stream tag, the oc describe is command shows multiple image IDs for that tag:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

\* registry.access.redhat.com/ubi9/php-80@sha256:**2b82...f544**

2 days ago

registry.access.redhat.com/ubi9/php-80@sha256:**8840...94f0**

5 days ago

registry.access.redhat.com/ubi9/php-80@sha256:**506c...5d90**

9 days ago

In the previous example, the asterisk (\*) shows which image ID is the current one for each image stream tag. It is usually the latest one to be imported, and the first one that is listed.

When an OpenShift image stream tag references a container image from an external registry, you must explicitly update the image stream tag to get new image IDs from the external registry. By default, OpenShift does not monitor external registries for changes to the image ID that is associated with an image name.

You can configure an image stream tag to check the external registry for updates on a defined schedule. By default, new image stream tags do not check for updated images.

Creating Image Streams and Tags

In addition to the image streams in the openshift project, you can create image streams in your project so that the resources in that project, such as Deployment objects, can use them.

Use the oc create is command to create image streams in the current project. The following example creates an image stream named keycloak:

[user@host ~]$ **oc create is keycloak**

After you create the image stream, use the oc create istag command to add image stream tags. The following example adds the 20.0 tag to the keycloak image stream. In this example, the image stream tag refers to the quay.io/keycloak/keycloak:20.0.2 image from the Quay.io public repository.

[user@host ~]$ **oc create istag keycloak:20.0 \**

**--from-image quay.io/keycloak/keycloak:20.0.2**

Repeat the preceding command if you need more image stream tags:

[user@host ~]$ **oc create istag keycloak:19.0 \**

**--from-image quay.io/keycloak/keycloak:19.0**

Use the oc tag *SOURCE-IMAGE* *IMAGE-STREAM-TAG* command to update an image stream tag with a new source image reference. The following example changes the keycloak:20.0 image stream tag to point to the quay.io/keycloak/keycloak:20.0.3 image:

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0**

Use the oc describe is command to verify that the image stream tag points to the SHA ID of the source image:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

Created: 5 minutes ago

Labels: <none>

Annotations: openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z

Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak

Image Lookup: local=false

Unique Images: 3

Tags: 2

20.0

tagged from quay.io/keycloak/keycloak:20.0.3

\* quay.io/keycloak/keycloak@sha256:**c167...62e9**

47 seconds ago

quay.io/keycloak/keycloak@sha256:**5569...b311**

5 minutes ago

19.0

tagged from quay.io/keycloak/keycloak:19.0

\* quay.io/keycloak/keycloak@sha256:**40cc...ffde**

5 minutes ago

Importing Image Stream Tags Periodically

When you create an image stream tag, OpenShift configures it with the SHA ID of the source image that you specify. After that initial creation, the image stream tag does not change, even if the developer pushes a new version of the source image.

By using image stream tags, you are in control of the images that your applications are using. If you want to use a new image version, then you manually need to update the image stream tag to point to that new version.

However, for some container registries that you trust, or for some specific images, you might prefer the image stream tags to automatically refresh.

For example, Red Hat regularly updates the images from the Red Hat Ecosystem Catalog with bug and security fixes. To benefit from these updates as soon as Red Hat releases them, you can configure your image stream tags to regularly refresh.

OpenShift can periodically verify whether a new image version is available. When it detects a new version, it automatically updates the image stream tag. To activate that periodic refresh, add the --scheduled option to the oc tag command.

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 --scheduled**

By default, OpenShift verifies the image every 15 minutes. This period is a setting that your cluster administrators can adapt.

Configuring Image Pull-through

When OpenShift starts a pod that uses an image stream tag, it pulls the corresponding image from the source container registry.

When the image comes from a registry on the internet, pulling the image can take time, or even fail in case of a network outage. Some public registries have bandwidth throttling rules that can slow down your downloads further.

To mitigate these issues, you can configure your image stream tags to cache the images in the OpenShift internal container registry. The first time that OpenShift pulls the image, it downloads the image from the source repository and then stores the image in its internal registry. After that initial pull, OpenShift retrieves the image from the internal registry.

To activate image pull-through, add the --reference-policy local option to the oc tag command.

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 \**

**--reference-policy local**

Using Image Streams in Deployments

When you create a Deployment object, you can specify an image stream instead of a container image from a registry. Using an image stream in Kubernetes workload resources, such as deployments, requires preparation:

* Create the image stream object in the same project as the Deployment object.
* Enable the local lookup policy in the image stream object.
* In the Deployment object, reference the image stream tag by its name, such as keycloak:20.0, and not by the full image name from the source registry.

Enabling the Local Lookup Policy

When you use an image stream in a Deployment object, OpenShift looks for that image stream in the current project. However, OpenShift searches only the image streams that you enabled the local lookup policy for.

Use the oc set image-lookup command to enable the local lookup policy for an image stream:

[user@host ~]$ **oc set image-lookup keycloak**

Use the oc describe is command to verify that the policy is active:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

Created: 3 hours ago

Labels: <none>

Annotations: openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z

Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak

**Image Lookup: local=true**

Unique Images: 3

Tags: 2

*...output omitted...*

You can also retrieve the local lookup policy status for all the image streams in the current project by running the oc set image-lookup command without parameters:

[user@host ~]$ **oc set image-lookup**

NAME LOCAL

keycloak true

zabbix-agent false

nagios false

To disable the local lookup policy, add the --enabled=false option to the oc set image-lookup command:

[user@host ~]$ **oc set image-lookup keycloak --enabled=false**

Configuring Image Streams in Deployments

When you create a Deployment object by using the oc create deployment command, use the --image option to specify the image stream tag:

[user@host ~]$ **oc create deployment mykeycloak --image keycloak:20.0**

When you use a short name, OpenShift looks for a matching image stream in the current project. OpenShift considers only the image streams that you enabled the local lookup policy for. If it does not find an image stream, then OpenShift looks for a regular container image in the allowed container registries. The reference documentation at the end of this lecture describes how to configure these allowed registries.

You can also use image streams with other Kubernetes workload resources:

* Job objects that you can create by using the following command:

[user@host ~]$ **oc create job *NAME* --image *IMAGE-STREAM-TAG* -- *COMMAND***

* CronJob objects that you can create by using the following command:
* [user@host ~]$ **oc create cronjob *NAME* --image *IMAGE-STREAM-TAG* \**

**--schedule *CRON-SYNTAX* -- *COMMAND***

* Pod objects that you can create by using the following command:

[user@host ~]$ **oc run *NAME* --image *IMAGE-STREAM-TAG***

Another section in this course discusses how changing an image stream tag can automatically roll out the associated deployments.

Reproducible Deployments with OpenShift Image Streams

Objectives

* Ensure reproducibility of application deployments by using image streams and short image names.

Image Streams

*Image streams* are one of the main differentiators between OpenShift and upstream Kubernetes. Kubernetes resources reference container images directly, but OpenShift resources, such as deployment configurations and build configurations, reference image streams. OpenShift also extends Kubernetes resources, such as Kubernetes Deployments, with annotations that make them work with OpenShift image streams.

With image streams, OpenShift can ensure reproducible, stable deployments of containerized applications and also rollbacks of deployments to their latest known-good state.

Image streams provide a stable, short name to reference a container image that is independent of any registry server and container runtime configuration.

As an example, an organization could start by downloading container images directly from the Red Hat public registry and later set up an enterprise registry as a mirror of those images to save bandwidth. OpenShift users would not notice any change, because they still refer to these images by using the same image stream name. Users of the RHEL container tools would notice the change, because they would be required either to change the registry names in their commands, or to change their container engine configurations to search for the local mirror first.

In other scenarios, the indirection that an image stream provides can be helpful. Suppose that you start with a database container image that has security issues, and the vendor takes too long to update the image with fixes. Later, you find an alternative vendor who provides an alternative container image for the same database, with those security issues already fixed, and even better, with a track record of providing timely updates to them. If those container images are compatible regarding configuration of environment variables and volumes, you could change your image stream to point to the image from the alternative vendor.

Red Hat provides hardened, supported container images that work mostly as drop-in replacements of container images from some popular open source projects, such as the MariaDB database.

Image Stream Tags

An image stream represents one or more sets of container images. Each set, or stream, is identified by an *image stream tag*. Unlike container images in a registry server, which have multiple tags from the same image repository (or user or organization), an image stream can have multiple image stream tags that reference container images from different registry servers and from different image repositories.

An image stream provides default configurations for a set of image stream tags. Each image stream tag references one stream of container images, and can override most configurations from its associated image stream.

An image stream tag stores a copy of the metadata about its current container image. Storing metadata supports faster search and inspection of container images, because you do not need to reach its source registry server.

You can also configure an image stream tag to store the source image layers in the OpenShift internal container registry, which acts as a local image cache. Storing image layers locally avoids the need to fetch these layers from their source registry server. Consumers of the cached image, such as pods and deployments, just reference the internal registry as the source registry of the image.

For some other OpenShift resource types that relate to image streams, you can usually dismiss them as implementation details of the internal registry, and focus only on image streams and image stream tags.

To better visualize the relationship between image streams and image stream tags, you can explore the openshift project that is pre-created in all OpenShift clusters. You can see many image streams in that project, including the php image stream:

[user@host ~]$ **oc get is -n openshift -o name**

*...output omitted...*

imagestream.image.openshift.io/nodejs

imagestream.image.openshift.io/perl

**imagestream.image.openshift.io/php**

imagestream.image.openshift.io/postgresql

imagestream.image.openshift.io/python

*...output omitted...*

Several tags exist for the php image stream, and an image stream tag resource exists for each tag:

[user@host ~]$ **oc get istag -n openshift | grep php**

8.0-ubi9 image-registry ... 6 days ago

8.0-ubi8 image-registry ... 6 days ago

7.4-ubi8 image-registry ... 6 days ago

7.3-ubi7 image-registry ... 6 days ago

The oc describe command on an image stream shows information from both the image stream and its image stream tags:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

Tags: 5

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

**8.0-ubi8** (latest)

tagged from registry.access.redhat.com/ubi8/php-80:latest

*...output omitted...*

**7.4-ubi8**

tagged from registry.access.redhat.com/ubi8/php-74:latest

*...output omitted...*

**7.3-ubi7**

tagged from registry.access.redhat.com/ubi7/php-73:latest

*...output omitted...*

In the previous example, each of the php image stream tags refers to a different image name.

Image Names, Tags, and IDs

The textual name of a container image is a string. This name is sometimes interpreted as being made of multiple components, such as registry-host-name/repository-or-organization-or-user-name/image-name:tag-name, but splitting the image name into its components is a matter of convention, not of structure.

A SHA image ID is a SHA-256 hash that uniquely identifies an immutable container image. You cannot modify a container image. Instead, you create a container image that has a new ID. When you push a new container image to a registry server, the server associates the existing textual name with the new image ID.

When you start a container from an image name, you download the image that is currently associated with that image name. The image ID behind that name might change at any moment, and the next container that you start might have a different image ID. If the image that is associated with an image name has any issues, and you know only the image name, then you cannot roll back to an earlier image.

OpenShift image stream tags keep a history of the latest image IDs that they fetched from a registry server. The history of image IDs is the stream of images from an image stream tag. You can use the history inside an image stream tag to roll back to a previous image, if for example a new container image causes a deployment error.

Updating a container image in an external registry does not automatically update an image stream tag. The image stream tag keeps the reference to the last image ID that it fetched. This behavior is crucial to scaling applications, because it isolates OpenShift from changes that happen at a registry server.

Suppose that you deploy an application from an external registry, and after a few days of testing with a few users, you decide to scale its deployment to enable a larger user population. In the meantime, your vendor updates the container image on the external registry. If OpenShift had no image stream tags, then the new pods would get the new container image, which is different from the image on the original pod. Depending on the changes, this new image could cause your application to fail. Because OpenShift stores the image ID of the original image in an image stream tag, it can create new pods by using the same image ID and avoid any incompatibility between the original and updated image.

OpenShift keeps the image ID of the first pod, and ensures that new pods use the same image ID. OpenShift ensures that all pods use the same image.

To better visualize the relationship between an image stream, an image stream tag, an image name, and an image ID, refer to the following oc describe is command, which shows the source image and current image ID for each image stream tag:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

\* registry.access.redhat.com/ubi9/php-80@sha256:**2b82...f544**

2 days ago

**8.0-ubi8** (latest)

tagged from registry.access.redhat.com/ubi8/php-80:latest

\* registry.access.redhat.com/ubi8/php-80@sha256:**2c74...5ef4**

2 days ago

*...output omitted...*

If your OpenShift cluster administrator already updated the php:8.0-ubi9 image stream tag, the oc describe is command shows multiple image IDs for that tag:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

\* registry.access.redhat.com/ubi9/php-80@sha256:**2b82...f544**

2 days ago

registry.access.redhat.com/ubi9/php-80@sha256:**8840...94f0**

5 days ago

registry.access.redhat.com/ubi9/php-80@sha256:**506c...5d90**

9 days ago

In the previous example, the asterisk (\*) shows which image ID is the current one for each image stream tag. It is usually the latest one to be imported, and the first one that is listed.

When an OpenShift image stream tag references a container image from an external registry, you must explicitly update the image stream tag to get new image IDs from the external registry. By default, OpenShift does not monitor external registries for changes to the image ID that is associated with an image name.

You can configure an image stream tag to check the external registry for updates on a defined schedule. By default, new image stream tags do not check for updated images.

Creating Image Streams and Tags

In addition to the image streams in the openshift project, you can create image streams in your project so that the resources in that project, such as Deployment objects, can use them.

Use the oc create is command to create image streams in the current project. The following example creates an image stream named keycloak:

[user@host ~]$ **oc create is keycloak**

After you create the image stream, use the oc create istag command to add image stream tags. The following example adds the 20.0 tag to the keycloak image stream. In this example, the image stream tag refers to the quay.io/keycloak/keycloak:20.0.2 image from the Quay.io public repository.

[user@host ~]$ **oc create istag keycloak:20.0 \**

**--from-image quay.io/keycloak/keycloak:20.0.2**

Repeat the preceding command if you need more image stream tags:

[user@host ~]$ **oc create istag keycloak:19.0 \**

**--from-image quay.io/keycloak/keycloak:19.0**

Use the oc tag *SOURCE-IMAGE* *IMAGE-STREAM-TAG* command to update an image stream tag with a new source image reference. The following example changes the keycloak:20.0 image stream tag to point to the quay.io/keycloak/keycloak:20.0.3 image:

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0**

Use the oc describe is command to verify that the image stream tag points to the SHA ID of the source image:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

Created: 5 minutes ago

Labels: <none>

Annotations: openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z

Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak

Image Lookup: local=false

Unique Images: 3

Tags: 2

20.0

tagged from quay.io/keycloak/keycloak:20.0.3

\* quay.io/keycloak/keycloak@sha256:**c167...62e9**

47 seconds ago

quay.io/keycloak/keycloak@sha256:**5569...b311**

5 minutes ago

19.0

tagged from quay.io/keycloak/keycloak:19.0

\* quay.io/keycloak/keycloak@sha256:**40cc...ffde**

5 minutes ago

Importing Image Stream Tags Periodically

When you create an image stream tag, OpenShift configures it with the SHA ID of the source image that you specify. After that initial creation, the image stream tag does not change, even if the developer pushes a new version of the source image.

By using image stream tags, you are in control of the images that your applications are using. If you want to use a new image version, then you manually need to update the image stream tag to point to that new version.

However, for some container registries that you trust, or for some specific images, you might prefer the image stream tags to automatically refresh.

For example, Red Hat regularly updates the images from the Red Hat Ecosystem Catalog with bug and security fixes. To benefit from these updates as soon as Red Hat releases them, you can configure your image stream tags to regularly refresh.

OpenShift can periodically verify whether a new image version is available. When it detects a new version, it automatically updates the image stream tag. To activate that periodic refresh, add the --scheduled option to the oc tag command.

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 --scheduled**

By default, OpenShift verifies the image every 15 minutes. This period is a setting that your cluster administrators can adapt.

Configuring Image Pull-through

When OpenShift starts a pod that uses an image stream tag, it pulls the corresponding image from the source container registry.

When the image comes from a registry on the internet, pulling the image can take time, or even fail in case of a network outage. Some public registries have bandwidth throttling rules that can slow down your downloads further.

To mitigate these issues, you can configure your image stream tags to cache the images in the OpenShift internal container registry. The first time that OpenShift pulls the image, it downloads the image from the source repository and then stores the image in its internal registry. After that initial pull, OpenShift retrieves the image from the internal registry.

To activate image pull-through, add the --reference-policy local option to the oc tag command.

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 \**

**--reference-policy local**

Using Image Streams in Deployments

When you create a Deployment object, you can specify an image stream instead of a container image from a registry. Using an image stream in Kubernetes workload resources, such as deployments, requires preparation:

* Create the image stream object in the same project as the Deployment object.
* Enable the local lookup policy in the image stream object.
* In the Deployment object, reference the image stream tag by its name, such as keycloak:20.0, and not by the full image name from the source registry.

Enabling the Local Lookup Policy

When you use an image stream in a Deployment object, OpenShift looks for that image stream in the current project. However, OpenShift searches only the image streams that you enabled the local lookup policy for.

Use the oc set image-lookup command to enable the local lookup policy for an image stream:

[user@host ~]$ **oc set image-lookup keycloak**

Use the oc describe is command to verify that the policy is active:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

Created: 3 hours ago

Labels: <none>

Annotations: openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z

Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak

**Image Lookup: local=true**

Unique Images: 3

Tags: 2

*...output omitted...*

You can also retrieve the local lookup policy status for all the image streams in the current project by running the oc set image-lookup command without parameters:

[user@host ~]$ **oc set image-lookup**

NAME LOCAL

keycloak true

zabbix-agent false

nagios false

To disable the local lookup policy, add the --enabled=false option to the oc set image-lookup command:

[user@host ~]$ **oc set image-lookup keycloak --enabled=false**

Configuring Image Streams in Deployments

When you create a Deployment object by using the oc create deployment command, use the --image option to specify the image stream tag:

[user@host ~]$ **oc create deployment mykeycloak --image keycloak:20.0**

When you use a short name, OpenShift looks for a matching image stream in the current project. OpenShift considers only the image streams that you enabled the local lookup policy for. If it does not find an image stream, then OpenShift looks for a regular container image in the allowed container registries. The reference documentation at the end of this lecture describes how to configure these allowed registries.

You can also use image streams with other Kubernetes workload resources:

* Job objects that you can create by using the following command:

[user@host ~]$ **oc create job *NAME* --image *IMAGE-STREAM-TAG* -- *COMMAND***

* CronJob objects that you can create by using the following command:
* [user@host ~]$ **oc create cronjob *NAME* --image *IMAGE-STREAM-TAG* \**

**--schedule *CRON-SYNTAX* -- *COMMAND***

* Pod objects that you can create by using the following command:

[user@host ~]$ **oc run *NAME* --image *IMAGE-STREAM-TAG***

Another section in this course discusses how changing an image stream tag can automatically roll out the associated deployments.

Reproducible Deployments with OpenShift Image Streams

Objectives

* Ensure reproducibility of application deployments by using image streams and short image names.

Image Streams

*Image streams* are one of the main differentiators between OpenShift and upstream Kubernetes. Kubernetes resources reference container images directly, but OpenShift resources, such as deployment configurations and build configurations, reference image streams. OpenShift also extends Kubernetes resources, such as Kubernetes Deployments, with annotations that make them work with OpenShift image streams.

With image streams, OpenShift can ensure reproducible, stable deployments of containerized applications and also rollbacks of deployments to their latest known-good state.

Image streams provide a stable, short name to reference a container image that is independent of any registry server and container runtime configuration.

As an example, an organization could start by downloading container images directly from the Red Hat public registry and later set up an enterprise registry as a mirror of those images to save bandwidth. OpenShift users would not notice any change, because they still refer to these images by using the same image stream name. Users of the RHEL container tools would notice the change, because they would be required either to change the registry names in their commands, or to change their container engine configurations to search for the local mirror first.

In other scenarios, the indirection that an image stream provides can be helpful. Suppose that you start with a database container image that has security issues, and the vendor takes too long to update the image with fixes. Later, you find an alternative vendor who provides an alternative container image for the same database, with those security issues already fixed, and even better, with a track record of providing timely updates to them. If those container images are compatible regarding configuration of environment variables and volumes, you could change your image stream to point to the image from the alternative vendor.

Red Hat provides hardened, supported container images that work mostly as drop-in replacements of container images from some popular open source projects, such as the MariaDB database.

Image Stream Tags

An image stream represents one or more sets of container images. Each set, or stream, is identified by an *image stream tag*. Unlike container images in a registry server, which have multiple tags from the same image repository (or user or organization), an image stream can have multiple image stream tags that reference container images from different registry servers and from different image repositories.

An image stream provides default configurations for a set of image stream tags. Each image stream tag references one stream of container images, and can override most configurations from its associated image stream.

An image stream tag stores a copy of the metadata about its current container image. Storing metadata supports faster search and inspection of container images, because you do not need to reach its source registry server.

You can also configure an image stream tag to store the source image layers in the OpenShift internal container registry, which acts as a local image cache. Storing image layers locally avoids the need to fetch these layers from their source registry server. Consumers of the cached image, such as pods and deployments, just reference the internal registry as the source registry of the image.

For some other OpenShift resource types that relate to image streams, you can usually dismiss them as implementation details of the internal registry, and focus only on image streams and image stream tags.

To better visualize the relationship between image streams and image stream tags, you can explore the openshift project that is pre-created in all OpenShift clusters. You can see many image streams in that project, including the php image stream:

[user@host ~]$ **oc get is -n openshift -o name**

*...output omitted...*

imagestream.image.openshift.io/nodejs

imagestream.image.openshift.io/perl

**imagestream.image.openshift.io/php**

imagestream.image.openshift.io/postgresql

imagestream.image.openshift.io/python

*...output omitted...*

Several tags exist for the php image stream, and an image stream tag resource exists for each tag:

[user@host ~]$ **oc get istag -n openshift | grep php**

8.0-ubi9 image-registry ... 6 days ago

8.0-ubi8 image-registry ... 6 days ago

7.4-ubi8 image-registry ... 6 days ago

7.3-ubi7 image-registry ... 6 days ago

The oc describe command on an image stream shows information from both the image stream and its image stream tags:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

Tags: 5

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

**8.0-ubi8** (latest)

tagged from registry.access.redhat.com/ubi8/php-80:latest

*...output omitted...*

**7.4-ubi8**

tagged from registry.access.redhat.com/ubi8/php-74:latest

*...output omitted...*

**7.3-ubi7**

tagged from registry.access.redhat.com/ubi7/php-73:latest

*...output omitted...*

In the previous example, each of the php image stream tags refers to a different image name.

Image Names, Tags, and IDs

The textual name of a container image is a string. This name is sometimes interpreted as being made of multiple components, such as registry-host-name/repository-or-organization-or-user-name/image-name:tag-name, but splitting the image name into its components is a matter of convention, not of structure.

A SHA image ID is a SHA-256 hash that uniquely identifies an immutable container image. You cannot modify a container image. Instead, you create a container image that has a new ID. When you push a new container image to a registry server, the server associates the existing textual name with the new image ID.

When you start a container from an image name, you download the image that is currently associated with that image name. The image ID behind that name might change at any moment, and the next container that you start might have a different image ID. If the image that is associated with an image name has any issues, and you know only the image name, then you cannot roll back to an earlier image.

OpenShift image stream tags keep a history of the latest image IDs that they fetched from a registry server. The history of image IDs is the stream of images from an image stream tag. You can use the history inside an image stream tag to roll back to a previous image, if for example a new container image causes a deployment error.

Updating a container image in an external registry does not automatically update an image stream tag. The image stream tag keeps the reference to the last image ID that it fetched. This behavior is crucial to scaling applications, because it isolates OpenShift from changes that happen at a registry server.

Suppose that you deploy an application from an external registry, and after a few days of testing with a few users, you decide to scale its deployment to enable a larger user population. In the meantime, your vendor updates the container image on the external registry. If OpenShift had no image stream tags, then the new pods would get the new container image, which is different from the image on the original pod. Depending on the changes, this new image could cause your application to fail. Because OpenShift stores the image ID of the original image in an image stream tag, it can create new pods by using the same image ID and avoid any incompatibility between the original and updated image.

OpenShift keeps the image ID of the first pod, and ensures that new pods use the same image ID. OpenShift ensures that all pods use the same image.

To better visualize the relationship between an image stream, an image stream tag, an image name, and an image ID, refer to the following oc describe is command, which shows the source image and current image ID for each image stream tag:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

\* registry.access.redhat.com/ubi9/php-80@sha256:**2b82...f544**

2 days ago

**8.0-ubi8** (latest)

tagged from registry.access.redhat.com/ubi8/php-80:latest

\* registry.access.redhat.com/ubi8/php-80@sha256:**2c74...5ef4**

2 days ago

*...output omitted...*

If your OpenShift cluster administrator already updated the php:8.0-ubi9 image stream tag, the oc describe is command shows multiple image IDs for that tag:

[user@host ~]$ **oc describe is php -n openshift**

Name: php

Namespace: openshift

*...output omitted...*

**8.0-ubi9**

tagged from registry.access.redhat.com/ubi9/php-80:latest

*...output omitted...*

\* registry.access.redhat.com/ubi9/php-80@sha256:**2b82...f544**

2 days ago

registry.access.redhat.com/ubi9/php-80@sha256:**8840...94f0**

5 days ago

registry.access.redhat.com/ubi9/php-80@sha256:**506c...5d90**

9 days ago

In the previous example, the asterisk (\*) shows which image ID is the current one for each image stream tag. It is usually the latest one to be imported, and the first one that is listed.

When an OpenShift image stream tag references a container image from an external registry, you must explicitly update the image stream tag to get new image IDs from the external registry. By default, OpenShift does not monitor external registries for changes to the image ID that is associated with an image name.

You can configure an image stream tag to check the external registry for updates on a defined schedule. By default, new image stream tags do not check for updated images.

Creating Image Streams and Tags

In addition to the image streams in the openshift project, you can create image streams in your project so that the resources in that project, such as Deployment objects, can use them.

Use the oc create is command to create image streams in the current project. The following example creates an image stream named keycloak:

[user@host ~]$ **oc create is keycloak**

After you create the image stream, use the oc create istag command to add image stream tags. The following example adds the 20.0 tag to the keycloak image stream. In this example, the image stream tag refers to the quay.io/keycloak/keycloak:20.0.2 image from the Quay.io public repository.

[user@host ~]$ **oc create istag keycloak:20.0 \**

**--from-image quay.io/keycloak/keycloak:20.0.2**

Repeat the preceding command if you need more image stream tags:

[user@host ~]$ **oc create istag keycloak:19.0 \**

**--from-image quay.io/keycloak/keycloak:19.0**

Use the oc tag *SOURCE-IMAGE* *IMAGE-STREAM-TAG* command to update an image stream tag with a new source image reference. The following example changes the keycloak:20.0 image stream tag to point to the quay.io/keycloak/keycloak:20.0.3 image:

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0**

Use the oc describe is command to verify that the image stream tag points to the SHA ID of the source image:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

Created: 5 minutes ago

Labels: <none>

Annotations: openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z

Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak

Image Lookup: local=false

Unique Images: 3

Tags: 2

20.0

tagged from quay.io/keycloak/keycloak:20.0.3

\* quay.io/keycloak/keycloak@sha256:**c167...62e9**

47 seconds ago

quay.io/keycloak/keycloak@sha256:**5569...b311**

5 minutes ago

19.0

tagged from quay.io/keycloak/keycloak:19.0

\* quay.io/keycloak/keycloak@sha256:**40cc...ffde**

5 minutes ago

Importing Image Stream Tags Periodically

When you create an image stream tag, OpenShift configures it with the SHA ID of the source image that you specify. After that initial creation, the image stream tag does not change, even if the developer pushes a new version of the source image.

By using image stream tags, you are in control of the images that your applications are using. If you want to use a new image version, then you manually need to update the image stream tag to point to that new version.

However, for some container registries that you trust, or for some specific images, you might prefer the image stream tags to automatically refresh.

For example, Red Hat regularly updates the images from the Red Hat Ecosystem Catalog with bug and security fixes. To benefit from these updates as soon as Red Hat releases them, you can configure your image stream tags to regularly refresh.

OpenShift can periodically verify whether a new image version is available. When it detects a new version, it automatically updates the image stream tag. To activate that periodic refresh, add the --scheduled option to the oc tag command.

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 --scheduled**

By default, OpenShift verifies the image every 15 minutes. This period is a setting that your cluster administrators can adapt.

Configuring Image Pull-through

When OpenShift starts a pod that uses an image stream tag, it pulls the corresponding image from the source container registry.

When the image comes from a registry on the internet, pulling the image can take time, or even fail in case of a network outage. Some public registries have bandwidth throttling rules that can slow down your downloads further.

To mitigate these issues, you can configure your image stream tags to cache the images in the OpenShift internal container registry. The first time that OpenShift pulls the image, it downloads the image from the source repository and then stores the image in its internal registry. After that initial pull, OpenShift retrieves the image from the internal registry.

To activate image pull-through, add the --reference-policy local option to the oc tag command.

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 \**

**--reference-policy local**

Using Image Streams in Deployments

When you create a Deployment object, you can specify an image stream instead of a container image from a registry. Using an image stream in Kubernetes workload resources, such as deployments, requires preparation:

* Create the image stream object in the same project as the Deployment object.
* Enable the local lookup policy in the image stream object.
* In the Deployment object, reference the image stream tag by its name, such as keycloak:20.0, and not by the full image name from the source registry.

Enabling the Local Lookup Policy

When you use an image stream in a Deployment object, OpenShift looks for that image stream in the current project. However, OpenShift searches only the image streams that you enabled the local lookup policy for.

Use the oc set image-lookup command to enable the local lookup policy for an image stream:

[user@host ~]$ **oc set image-lookup keycloak**

Use the oc describe is command to verify that the policy is active:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

Created: 3 hours ago

Labels: <none>

Annotations: openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z

Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak

**Image Lookup: local=true**

Unique Images: 3

Tags: 2

*...output omitted...*

You can also retrieve the local lookup policy status for all the image streams in the current project by running the oc set image-lookup command without parameters:

[user@host ~]$ **oc set image-lookup**

NAME LOCAL

keycloak true

zabbix-agent false

nagios false

To disable the local lookup policy, add the --enabled=false option to the oc set image-lookup command:

[user@host ~]$ **oc set image-lookup keycloak --enabled=false**

Configuring Image Streams in Deployments

When you create a Deployment object by using the oc create deployment command, use the --image option to specify the image stream tag:

[user@host ~]$ **oc create deployment mykeycloak --image keycloak:20.0**

When you use a short name, OpenShift looks for a matching image stream in the current project. OpenShift considers only the image streams that you enabled the local lookup policy for. If it does not find an image stream, then OpenShift looks for a regular container image in the allowed container registries. The reference documentation at the end of this lecture describes how to configure these allowed registries.

You can also use image streams with other Kubernetes workload resources:

* Job objects that you can create by using the following command:

[user@host ~]$ **oc create job *NAME* --image *IMAGE-STREAM-TAG* -- *COMMAND***

* CronJob objects that you can create by using the following command:
* [user@host ~]$ **oc create cronjob *NAME* --image *IMAGE-STREAM-TAG* \**

**--schedule *CRON-SYNTAX* -- *COMMAND***

* Pod objects that you can create by using the following command:

[user@host ~]$ **oc run *NAME* --image *IMAGE-STREAM-TAG***

Another section in this course discusses how changing an image stream tag can automatically roll out the associated deployments.

Guided Exercise: Reproducible Deployments with OpenShift Image Streams

Deploy an application that references container images indirectly by using image streams.

**Outcomes**

You should be able to create image streams and image stream tags, and deploy applications that use image stream tags.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise. It also creates the updates-imagestreams project and the /home/student/DO180/labs/updates-imagestreams/resources.txt file. The resources.txt file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

[student@workstation ~]$ **lab start updates-imagestreams**

**Instructions**

1. Log in to the OpenShift cluster as the developer user with the developer password. Use the updates-imagestreams project.
   1. Log in to the OpenShift cluster.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

*...output omitted...*

* 1. Set the updates-imagestreams project as the active project.
  2. [student@workstation ~]$ **oc project updates-imagestreams**

*...output omitted...*

1. Create the versioned-hello image stream and the v1.0 image stream tag from the registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0 image.
   1. Use the oc create is command to create the image stream.
   2. [student@workstation ~]$ **oc create is versioned-hello**

imagestream.image.openshift.io/versioned-hello created

* 1. Use the oc create istag command to create the image stream tag.
  2. [student@workstation ~]$ **oc create istag versioned-hello:v1.0 \**
  3. **--from-image registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0**

imagestreamtag.image.openshift.io/versioned-hello:v1.0 created

1. Enable image stream resolution for the versioned-hello image stream so that Kubernetes resources in the current project can use it.
   1. Use the oc set image-lookup command to enable image lookup resolution.
   2. [student@workstation ~]$ **oc set image-lookup versioned-hello**

imagestream.image.openshift.io/versioned-hello image lookup updated

* 1. Run the oc set image-lookup command without any arguments to verify your work.
  2. [student@workstation ~]$ **oc set image-lookup**
  3. NAME LOCAL

versioned-hello **true**

1. Review the image stream and confirm that the image stream tag refers to the source image by its SHA ID. Verify that the source image in the registry.ocp4.example.com:8443 registry has the same SHA ID.
   1. Retrieve the details of the versioned-hello image stream.

**Note**

To improve readability, the instructions truncate the SHA-256 strings.

[student@workstation ~]$ **oc describe is versioned-hello**

Name: versioned-hello

Namespace: updates-imagestreams

Created: 7 minutes ago

*...output omitted...*

v1.0

tagged from registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0

\* registry.ocp4.example.com:8443/.../versioned-hello@**sha256:66e0...105e**

7 minutes ago

* 1. Use the oc image info command to query the image from the classroom container registry. The SHA image ID is the same as the one from the image stream tag.
  2. [student@workstation ~]$ **oc image info \**
  3. **registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0**
  4. Name: registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0
  5. Digest: **sha256:66e0...105e**
  6. Media Type: application/vnd.docker.distribution.manifest.v2+json

*...output omitted...*

1. Create a deployment named version that uses the versioned-hello:v1.0 image stream tag.
   1. Use the oc create deployment command to create the object.
   2. [student@workstation ~]$ **oc create deployment version --image versioned-hello:v1.0**

deployment.apps/version created

* 1. Wait for the pod to start. You might have to rerun the command several times for the pod to report a Running status. The name of the pod on your system probably differs.
  2. [student@workstation ~]$ **oc get pods**
  3. NAME READY STATUS RESTARTS AGE

version-744bf7694b-bzhd2 1/1 **Running** 0 2m11s

1. Confirm that both the deployment and the pod refer to the image by its SHA ID.
   1. Retrieve the image that the deployment uses. The deployment refers to the image from the source registry by its SHA ID. The v1.0 image stream tag also points to that SHA image ID.
   2. [student@workstation ~]$ **oc get deployment -o wide**
   3. ... IMAGES ...

... registry.ocp4.example.com:8443/.../versioned-hello@**sha256:66e0...105e** ...

* 1. Retrieve the image that the pod is using. The pod is also referring to the image by its SHA ID.
  2. [student@workstation ~]$ **oc get pod version-*744bf7694b-bzhd2* \**
  3. **-o jsonpath='{.spec.containers[0].image}{"\n"}'**

registry.ocp4.example.com:8443/redhattraining/versioned-hello@**sha256:66e0...105e**

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish updates-imagestreams**

Automatic Image Updates with OpenShift Image Change Triggers

Objectives

* Ensure automatic update of application pods by using image streams with Kubernetes workload resources.

Using Triggers to Manage Images

Image stream tags record the SHA ID of the source container image. Thus, an image stream tag always points to an immutable image.

If a new version of the source image becomes available, then you can change the image stream tag to point to that new image. However, a Deployment object that uses the image stream tag does not roll out automatically. For an automatic rollout, you must configure the Deployment object with an image trigger.

If you update an image stream tag to point to a new image version, and you notice that this version does not work as expected, then you can revert the image stream tag. Deployment objects for which you configured a trigger automatically roll back to that previous image.

Other Kubernetes workloads also support image triggers, such as Pod, CronJob, and Job objects.

Configuring Image Trigger for Deployments

Before you can configure image triggers for a Deployment object, ensure that the Deployment object is using image stream tags for its containers:

* Create the image stream object in the same project as the Deployment object.
* Enable the local lookup policy in the image stream object by using the oc set image-lookup command.
* In the Deployment object, reference the image stream tags by their names, such as keycloak:20, and not by the full image names from the source registry.

Image triggers apply at the container level. If your Deployment object includes several containers, then you can specify a trigger for each one. Before you can set triggers, retrieve the container names:

[user@host ~]$ **oc get deployment mykeycloak -o wide**

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS ...

mykeycloak 0/1 1 0 6s **keycloak** ...

Use the oc set triggers command to configure an image trigger for the container inside the Deployment object. Use the --from-image option to specify the image stream tag to watch.

[user@host ~]$ **oc set triggers deployment/mykeycloak --from-image keycloak:20 \**

**--containers keycloak**

OpenShift DeploymentConfig objects natively support image streams, and support automatic rollout on image change. DeploymentConfig resources have attributes to support image streams and triggers.

**Note**

As of Red Hat OpenShift Container Platform 4.14, DeploymentConfig objects are deprecated. DeploymentConfig objects are still supported, but are not recommended for new installations.

Instead, use Deployment objects or another alternative to provide declarative updates for pods.

In contrast, Kubernetes Deployment resources do not natively support image streams, and do not have attributes to store the related configuration. To provide automatic image rollout for Deployment objects, OpenShift adds the image.openshift.io/triggers annotation to store the configuration in JSON format.

The following example retrieves the content of the image.openshift.io/triggers annotation, and then uses the jq command to display the configuration in a more readable format:

[user@host ~]$ **oc get deployment mykeycloak \**

**-o jsonpath='{.metadata.annotations.image\.openshift\.io/triggers}' | jq .**

[

{

"from": {

"kind": "ImageStreamTag",

"name": "keycloak:20"

},

"fieldPath": "spec.template.spec.containers[?(@.name==\"keycloak\")].image"

}

]

The fieldPath attribute is a JSONPath expression that OpenShift uses to locate the attribute that stores the container image name. OpenShift updates that attribute with the new image name and SHA ID whenever the image stream tag changes.

For a more concise view, use the oc set triggers command with the name of the Deployment object as an argument:

[user@host ~]$ **oc set triggers deployment/mykeycloak**

NAME TYPE VALUE AUTO

deployments/mykeycloak config true

deployments/mykeycloak image keycloak:20 (keycloak) true

|  |  |
| --- | --- |
|  | OpenShift uses the configuration trigger to roll out the deployment whenever you change its configuration, such as to update environment variables or to configure the readiness probe. |
|  | OpenShift watches the keycloak:20 image stream tag that the keycloak container uses. |

The true value under the AUTO column indicates that the trigger is enabled.

You can disable the configuration trigger by using the oc rollout pause command, and you can re-enable it by using the oc rollout resume command.

You can disable the image trigger by adding the --manual option to the oc set triggers command:

[user@host ~]$ **oc set triggers deployment/mykeycloak --manual \**

**--from-image keycloak:20 --containers keycloak**

You re-enable the trigger by using the --auto option:

[user@host ~]$ **oc set triggers deployment/mykeycloak --auto \**

**--from-image keycloak:20 --containers keycloak**

You can remove the triggers from all the containers in the Deployment object by adding the --remove-all option to the command:

[user@host ~]$ **oc set triggers deployment/mykeycloak --remove-all**

Rolling out Deployments

A Deployment object with an image trigger automatically rolls out when the image stream tag changes.

The image stream tag might change because you manually update it to point to a new version of the source image. The image stream tag might also change automatically if you configure it for periodic refresh, by adding the --scheduled option to the oc tag command. When the image stream tag automatically changes, all the Deployment objects with a trigger that refers to that image stream tag also roll out.

Rolling Back Deployments

OpenShift DeploymentConfig objects and Kubernetes Deployment objects behave differently when you need to roll back because of a malfunctioning image.

For OpenShift DeploymentConfig objects, use the oc rollout undo command. The command rolls back the object and disables the image trigger. The command disables the trigger, because otherwise, after rolling back, OpenShift would notice that a new image is available, the malfunctioning image, and would then roll out. You must manually re-enable the image trigger after you fix the issue.

For Kubernetes Deployment objects, you cannot use the oc rollout undo command in the event of malfunctioning images, because the command does not disable the image triggers. If you use the command, then OpenShift notices that the deployment is already using the image that the trigger points to, and therefore does not roll back the application.

For Kubernetes Deployment objects, instead of rolling back the deployment, you revert the image stream tag. By reverting the image stream tag, OpenShift rolls out the Deployment object to use the previous image that the image stream tag is pointing to again.

Managing Image Stream Tags

You can create image streams and image stream tags in several ways. The following commands perform the same operation. They all create the keycloak image stream if it does not exist, and then create the keycloak:20.0.2 image stream tag:

[user@host ~]$ **oc create istag keycloak:20.0.2 \**

**--from-image quay.io/keycloak/keycloak:20.0.2**

[user@host ~]$ **oc import-image keycloak:20.0.2 \**

**--from quay.io/keycloak/keycloak:20.0.2 --confirm**

[user@host ~]$ **oc tag quay.io/keycloak/keycloak:20.0.2 keycloak:20.0.2**

You can rerun the oc import-image and oc tag commands to update the image stream tag from the source image. If the source image changes, then the commands update the image stream tag to point to that new version. However, you can use the oc create istag command only to initially create the image stream tag. You cannot update tags by using that command.

Use the --help option for more details about the commands.

You can create several image stream tags that point to the same image. The following command creates the keycloak:20 image stream tag, which points to the same image as the keycloak:20.0.2 image stream tag. In other words, the keycloak:20 image stream tag is an alias for the keycloak:20.0.2 image stream tag.

[user@host ~]$ **oc tag --alias keycloak:20.0.2 keycloak:20**

The oc describe is command reports that both tags point to the same image:

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

*...output omitted...*

**20.0.2 (20)**

tagged from quay.io/keycloak/keycloak:20.0.2

\* quay.io/keycloak/keycloak@sha256:5569...b311

3 minutes ago

Using aliases is a similar concept to floating tags for container images. Suppose that a new image version is available in the Quay.io repository. You could create an image stream tag for that new image:

[user@host ~]$ **oc create istag keycloak:20.0.3 \**

**--from-image quay.io/keycloak/keycloak:20.0.3**

imagestreamtag.image.openshift.io/keycloak:20.0.3 created

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

*...output omitted...*

**20.0.3**

tagged from quay.io/keycloak/keycloak:20.0.3

\* quay.io/keycloak/keycloak@sha256:c167...62e9

36 seconds ago

20.0.2 **(20)**

tagged from quay.io/keycloak/keycloak:20.0.2

\* quay.io/keycloak/keycloak@sha256:5569...b311

About an hour ago

The keycloak:20 image stream tag does not change. Therefore, the Deployment objects that use that tag do not roll out.

After testing the new image, you can move the keycloak:20 tag to point to the new image stream tag:

[user@host ~]$ **oc tag --alias keycloak:20.0.3 keycloak:20**

Tag keycloak:20 set up to track keycloak:20.0.3.

[user@host ~]$ **oc describe is keycloak**

Name: keycloak

Namespace: myproject

*...output omitted...*

**20.0.3 (20)**

tagged from quay.io/keycloak/keycloak:20.0.3

\* quay.io/keycloak/keycloak@sha256:c167...62e9

10 minutes ago

20.0.2

tagged from quay.io/keycloak/keycloak:20.0.2

\* quay.io/keycloak/keycloak@sha256:5569...b311

About an hour ago

Because the keycloak:20 image stream tag points to a new image, OpenShift rolls out all the Deployment objects that use that tag.

If the new application does not work as expected, then you can roll back the deployments by resetting the keycloak:20 tag to the previous image stream tag:

[user@host ~]$ **oc tag --alias keycloak:20.0.2 keycloak:20**

By providing a level of indirection, image streams give you control over managing the container images that you use in your OpenShift cluster.

Guided Exercise: Automatic Image Updates with OpenShift Image Change Triggers

Update an application that references container images indirectly though image streams.

**Outcomes**

* Add an image trigger to a deployment.
* Modify an image stream tag to point to a new image.
* Watch the rollout of the application.
* Roll back a deployment to the previous image.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise. It also creates the updates-triggers project and deploys a web application with 10 replicas.

The command creates the /home/student/DO180/labs/updates-triggers/resources.txt file. The resources.txt file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

[student@workstation ~]$ **lab start updates-triggers**

**Instructions**

1. Log in to the OpenShift cluster as the developer user with the developer password. Use the updates-triggers project.
   1. Log in to the OpenShift cluster.
   2. [student@workstation ~]$ **oc login -u developer -p developer \**
   3. **https://api.ocp4.example.com:6443**
   4. Login successful.

*...output omitted...*

* 1. Set the updates-triggers project as the active project.
  2. [student@workstation ~]$ **oc project updates-triggers**

*...output omitted...*

1. Inspect the versioned-hello image stream that the lab command created.
   1. Verify that the lab command enabled the local lookup policy for the versioned-hello image stream.
   2. [student@workstation ~]$ **oc set image-lookup**
   3. NAME LOCAL

versioned-hello **true**

* 1. Verify that the lab command created the versioned-hello:1 image stream tag. The image stream tag refers to the image in the classroom registry by its SHA ID.

**Note**

To improve readability, the instructions truncate the SHA-256 strings.

On your system, the commands return the full SHA-256 strings. Also, you must type the full SHA-256 string, to provide such a parameter to a command.

[student@workstation ~]$ **oc get istag**

NAME IMAGE REFERENCE ...

**versioned-hello:1** ...:8443/redhattraining/versioned-hello@**sha256:66e0...105e** ...

* 1. Verify that the lab command created the versioned-hello:1 image stream tag from the registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-123 image.
  2. [student@workstation ~]$ **oc get istag versioned-hello:1 \**
  3. **-o jsonpath='{.tag.from.name}{"\n"}'**

registry.ocp4.example.com:8443/redhattraining/versioned-hello:**1-123**

1. Inspect the Deployment object that the lab command created. Verify that the application is available from outside the cluster.
   1. List the Deployment objects. The version deployment retrieved the SHA image ID from the versioned-hello:1 image stream tag. The Deployment object includes a container named versioned-hello. You use that information in a later step, when you configure the trigger.
   2. [student@workstation ~]$ **oc get deployment -o wide**
   3. NAME READY ... CONTAINERS IMAGES ...

version 10/10 ... **versioned-hello** .../versioned-hello@**sha256:66e0...105e** ...

* 1. Open a new terminal.
  2. Run the /home/student/DO180/labs/updates-triggers/curl\_loop.sh script that the lab command prepared. The script sends web requests to the application in a loop. Leave the script running and do not interrupt it.
  3. [student@workstation ~]$ **/home/student/DO180/labs/updates-triggers/curl\_loop.sh**
  4. Hi!
  5. Hi!
  6. Hi!
  7. Hi!

*...output omitted...*

1. Add an image trigger to the Deployment object.
   1. Switch back to the first terminal window, and then use the oc set triggers command to add the trigger for the versioned-hello:1 image stream tag to the versioned-hello container.
   2. [student@workstation ~]$ **oc set triggers deployment/version \**
   3. **--from-image versioned-hello:1 --containers versioned-hello**

deployment.apps/version triggers updated

* 1. Review the definition of the trigger from the image.openshift.io/triggers annotation.
  2. [student@workstation ~]$ **oc get deployment version \**
  3. **-o jsonpath='{.metadata.annotations.image\.openshift\.io/triggers}' | jq .**
  4. [
  5. {
  6. "from": {
  7. "kind": "ImageStreamTag",
  8. "name": "versioned-hello:1"
  9. },
  10. "fieldPath": "spec.template.spec.containers[?(@.name==\"versioned-hello\")].image"
  11. }

]

1. Update the versioned-hello:1 image stream tag to point to the 1-125 tag of the registry.ocp4.example.com:8443/redhattraining/versioned-hello image. Watch the output of the curl\_loop.sh script to verify that the Deployment object automatically rolls out.
   1. Use the oc tag command to update the versioned-hello:1 tag.
   2. [student@workstation ~]$ **oc tag \**
   3. **registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-125 \**
   4. **versioned-hello:1**

Tag versioned-hello:1 set to registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-125.

* 1. Changing the image stream tag triggered a rolling update. Watch the output of the curl\_loop.sh script in the second terminal.

Before the update, only pods that use the earlier version of the image reply. During the rolling updates, both old and new pods respond. After the update, only pods that run the latest version of the image reply. The following output probably differs on your system.

*...output omitted...*

Hi!

Hi!

Hi!

Hi!

Hi! v1.1

Hi! v1.1

Hi!

Hi! v1.1

Hi!

Hi! v1.1

Hi! v1.1

Hi! v1.1

Hi! v1.1

*...output omitted...*

Do not stop the script.

1. Inspect the Deployment object and the image stream.
   1. List the version deployment and notice that the image changed.
   2. [student@workstation ~]$ **oc get deployment version -o wide**
   3. NAME READY ... CONTAINERS IMAGES ...

version 10/10 ... versioned-hello .../versioned-hello@**sha256:834d...fcb4** ...

* 1. Display the details of the versioned-hello image stream. The versioned-hello:1 image stream tag points to the image with the same SHA ID as in the Deployment object.

Notice that the preceding image is still available. In the following step, you roll back to that image by specifying its SHA ID.

[student@workstation ~]$ **oc describe is versioned-hello**

Name: versioned-hello

Namespace: updates-triggers

*...output omitted...*

**1**

tagged from registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-125

**\* registry.ocp4.example.com:8443/.../versioned-hello@sha256:834d...fcb4**

6 minutes ago

registry.ocp4.example.com:8443/.../versioned-hello@sha256:66e0...105e

37 minutes ago

1. Roll back the Deployment object by reverting the versioned-hello:1 image stream tag.
   1. Use the oc tag command. For the source image, copy and paste the old image name and the SHA ID from the output of the preceding command.
   2. [student@workstation ~]$ **oc tag \**
   3. **registry.ocp4.example.com:8443/redhattraining/versioned-hello@sha256:66e0...105e \**
   4. **versioned-hello:1**

Tag versioned-hello:1 set to registry.ocp4.example.com:8443/redhattraining/versioned-hello@sha256:66e0...105e.

* 1. Watch the output of the curl\_loop.sh script in the second terminal. The pods that run the v1.0 version of the application are responding again. The following output probably differs on your system.
  2. *...output omitted...*
  3. Hi! v1.1
  4. Hi! v1.1
  5. Hi! v1.1
  6. Hi! v1.1
  7. Hi!
  8. Hi! v1.1
  9. Hi! v1.1
  10. Hi! v1.1
  11. Hi!
  12. Hi! v1.1
  13. Hi! v1.1
  14. Hi!
  15. Hi!
  16. Hi!

*...output omitted...*

Press **Ctrl**+**C** to quit the script. Close that second terminal when done.

**Finish**

On the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish updates-triggers**

Lab: Manage Application Updates

Update two live applications to their latest releases as identified by non-floating tags.

**Outcomes**

You should be able to configure Deployment objects with images and triggers, and configure image stream tags and aliases.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise. It also creates the updates-review project and deploys two applications, app1 and app2, in that project.

The command creates the /home/student/DO180/labs/updates-review/resources.txt file. The resources.txt file contains the name of the images that you use during the exercise. You can use the file to copy and paste these image names.

[student@workstation ~]$ **lab start updates-review**

**Instructions**

The API URL of your OpenShift cluster is https://api.ocp4.example.com:6443, and the oc command is already installed on your workstation machine.

Log in to the OpenShift cluster as the developer user with the developer password. Use the updates-review project for your work.

1. Your team created the app1 deployment in the updates-review project from the registry.ocp4.example.com:8443/redhattraining/php-ssl:latest container image. Recently, a developer in your organization pushed a new version of the image and then reassigned the latest tag to that version.

Reconfigure the app1 deployment to use the 1-222 static tag instead of the latest floating tag, to prevent accidental redeployment of your application with untested image versions that your developers can publish at any time.

* 1. Log in to the OpenShift cluster.
  2. [student@workstation ~]$ **oc login -u developer -p developer \**
  3. **https://api.ocp4.example.com:6443**
  4. Login successful.

*...output omitted...*

* 1. Set the updates-review project as the active project.
  2. [student@workstation ~]$ **oc project updates-review**

*...output omitted...*

* 1. Verify that the app1 deployment uses the latest tag. Retrieve the container name.
  2. [student@workstation ~]$ **oc get deployment/app1 -o wide**
  3. NAME READY ... CONTAINERS IMAGES ...

app1 1/1 ... **php-ssl** registry...:8443/redhattraining/php-ssl:**latest** ...

* 1. In the Deployment object, change the image to registry.ocp4.example.com:8443/redhattraining/php-ssl:1-222.
  2. [student@workstation ~]$ **oc set image deployment/app1 \**
  3. **php-ssl=registry.ocp4.example.com:8443/redhattraining/php-ssl:1-222**

deployment.apps/app1 image updated

* 1. Verify your work.
  2. [student@workstation ~]$ **oc get deployment/app1 -o wide**
  3. NAME READY ... CONTAINERS IMAGES ...

app1 1/1 ... php-ssl registry...:8443/redhattraining/php-ssl:**1-222** ...

Hide Solution

1. The app2 deployment is using the php-ssl:1 image stream tag, which is an alias for the php-ssl:1-222 image stream tag.

Enable image triggering for the app2 deployment, so that whenever the php-ssl:1 image stream tag changes, OpenShift rolls out the application. You test your configuration in a later step, when you reassign the php-ssl:1 alias to a new image stream tag.

* 1. Retrieve the container name from the Deployment object.
  2. [student@workstation ~]$ **oc get deployment/app2 -o wide**
  3. NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS ...

app2 1/1 1 1 21m **php-ssl** ...

* 1. Add the image trigger to the Deployment object.
  2. [student@workstation ~]$ **oc set triggers deployment/app2 \**
  3. **--from-image php-ssl:1 --containers php-ssl**

deployment.apps/app2 triggers updated

* 1. Verify your work.
  2. [student@workstation ~]$ **oc set triggers deployment/app2**
  3. NAME TYPE VALUE AUTO
  4. deployments/app2 config true

deployments/app2 image **php-ssl:1 (php-ssl)** true

Hide Solution

1. A new image version, registry.ocp4.example.com:8443/redhattraining/php-ssl:1-234, is available in the container registry. Your QA team tested and approved that version. It is ready for production.

Create the php-ssl:1-234 image stream tag that points to the new image. Move the php-ssl:1 image stream tag alias to the new php-ssl:1-234 image stream tag. Verify that the app2 application redeploys.

* 1. Create the php-ssl:1-234 image stream tag.
  2. [student@workstation ~]$ **oc create istag php-ssl:1-234 \**
  3. **--from-image registry.ocp4.example.com:8443/redhattraining/php-ssl:1-234**

imagestreamtag.image.openshift.io/php-ssl:1-234 created

* 1. Move the php-ssl:1 alias to the new php-ssl:1-234 image stream tag.
  2. [student@workstation ~]$ **oc tag --alias php-ssl:1-234 php-ssl:1**

Tag php-ssl:1 set up to track php-ssl:1-234.

* 1. Verify that the app2 application rolls out. The names of the replica sets on your system probably differ.
  2. [student@workstation ~]$ **oc describe deployment/app2**
  3. Name: app2
  4. Namespace: updates-review
  5. *...output omitted...*
  6. Events:
  7. Type ... Age Message
  8. ---- ---- -------
  9. Normal ... 33m ... Scaled up replica set app2-7dd589f6d5 to 1
  10. **Normal ... 3m30s ... Scaled up replica set app2-7bf5b7787 to 1**

**Normal ... 3m28s ... Scaled down replica set app2-7dd589f6d5 to 0 from 1**

Hide Solution

**Evaluation**

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

[student@workstation ~]$ **lab grade updates-review**

**Finish**

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

[student@workstation ~]$ **lab finish updates-review**

Summary

* You reference container images by tags or by SHA IDs. Image developers assign tags to images. Container registries compute and assign SHA IDs to images.
* Deployment objects have an imagePullPolicy attribute that specifies how compute nodes pull the image from the registry.
* A deployment strategy is a way of changing or upgrading your application to minimize the impact of those changes.
* Deployment objects support the rolling update and the re-create deployment strategies.
* OpenShift image stream and image stream tag resources provide stable references to container images.
* Kubernetes workload resources, such as Deployments and Jobs, can use image streams. You must create the image streams in the same project as the Kubernetes resources, and you must enable the local lookup policy in the image streams to use them.
* You can configure image monitoring in deployments so that OpenShift rolls out the application whenever the image stream tag changes.