Reproducible Deployments with OpenShift Image Streams

Objectives

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

Image Streams

Image streams provide a crucial abstraction layer for managing container images within a Red Hat OpenShift Container Platform (RHOCP) environment. For enterprises, this abstraction is key to establishing secure, consistent, and efficient software supply chains. By decoupling application definitions from the specific location and version of an image, image streams enable robust CI/CD pipelines, simplify image promotion across environments, and enhance security by controlling which images are available for deployment.

The distinction between a floating and a non-floating tag is a convention rather than technical. A developer is discouraged, although not prevented, from pushing a different image to an existing tag. Therefore, a pod that references an image tag is not guaranteed to retrieve the same image when the pod is restarted. Image streams resolve this problem and also provide essential rollback capabilities.

Image streams are one of the main differentiators between OpenShift and upstream Kubernetes. Whereas Kubernetes resources reference container images directly, OpenShift resources, such as build configurations, reference image streams. OpenShift also extends Kubernetes resources, such as Kubernetes Deployments, with annotations that enable them to use 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.

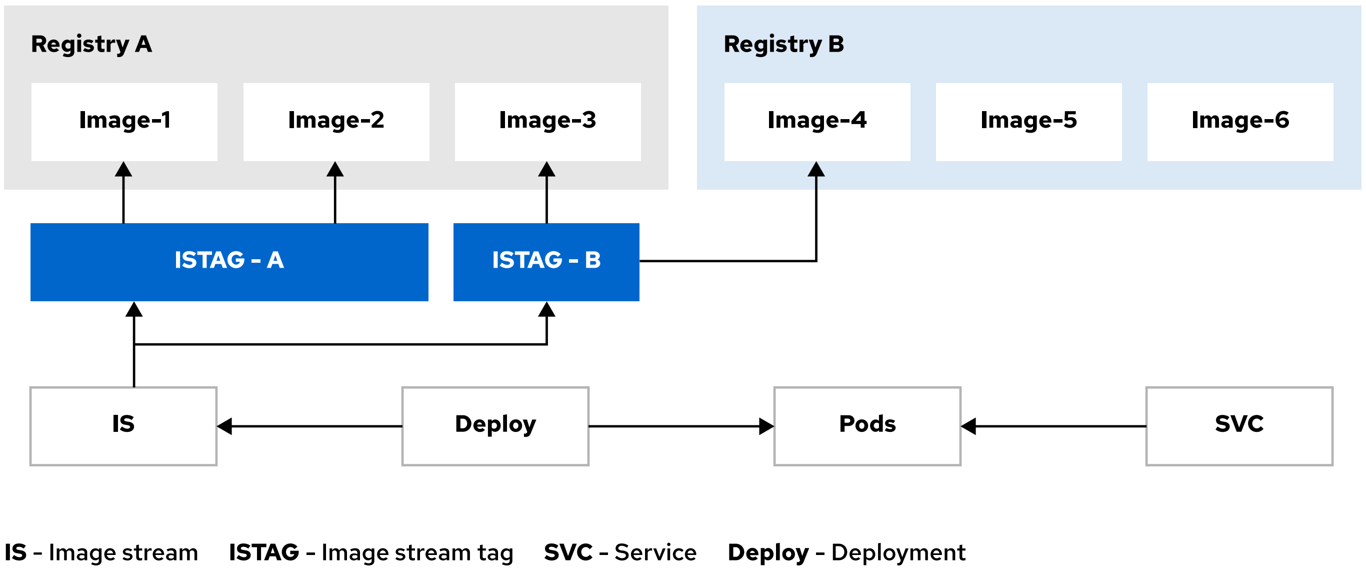
Image streams provide a stable, short name to reference a container image that is independent of any registry server and container runtime configuration.

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, an image stream can have multiple image stream tags that reference container images from different registry servers and 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.

In the following illustration, a deployment that uses the `Image-4` image can be rolled back to using the `Image-3` image, because the `ISTAG-B` image stream tag keeps a history of used images:



An image stream tag stores a copy of the metadata about its current container image, including the SHA image ID. The SHA image ID is a unique identifier that the container registry computes and assigns to images. 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, reference the internal registry as the source registry of the image.

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. Although the image name is sometimes interpreted as consisting of multiple components, such as `registry-host-name/repository-or-organization-or-user-name/image-name:tag-name`, splitting the image name into its components is a matter of convention, not of structure.

An SHA image ID is an SHA-256 hash that uniquely identifies an immutable container image. You cannot modify a container image. Instead, you create a container image with 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. The image stream tag behavior is crucial to scaling applications, because it isolates OpenShift from changes that happen on a registry server.

Suppose that you deploy an application from an external registry, and after a few days of testing with some 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 pods by using the same image ID and avoid any incompatibility between the original and the 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.

Image Stream Use Cases

Image streams provide an evolution approach for the container workflows of an organization, and can improve deployment reliability and resilience.

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 reduce 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 those users would need either to change the registry names in their commands, or to change their container engine configurations to search for the local mirror first.

A common enterprise use case for image streams is to manage the promotion of application images across different environments, such as for development, testing, and production. A CI/CD pipeline can build an image and push it to an image stream tag such as `myapp:latest-dev`. After automated tests pass, the same immutable image digest can be tagged into the testing environment's image stream, for example, as `myapp:qa-stable`. This use of the image stream tags ensures that the same artifact is tested. Finally, after successful QA, the image digest is tagged as `myapp:prod-v1.2`, triggering a production deployment. This process provides traceability and consistency throughout the application lifecycle.

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

Red Hat provides hardened, supported container images, such as the MariaDB database, that work mostly as drop-in replacements of container images from popular open source projects. Replacing unreliable image sources with supported Red Hat alternatives, when available, is a preferred use of image streams.

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 an image stream 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, you can add image stream tags. A common method is to import an image from a remote registry, which creates the image stream if it does not exist and adds the tag. Use the `oc import-image` command for this purpose. The following example adds the `25.0` tag to the `keycloak` image stream. In this example, the image stream tag refers to the `quay.io/keycloak/keycloak:25.0.2` image from the Quay.io public repository.

```
[user@host \~]$ oc import-image keycloak:25.0 \
  --from=quay.io/keycloak/keycloak:25.0.2 --confirm
```

Alternatively, use the `oc tag` *`SOURCE-IMAGE IMAGE-STREAM-TAG`* command to add image stream tags to an existing image stream.

```
[user@host ~]$ oc tag quay.io/keycloak/keycloak:25.0.2 keycloak:25.0
```

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

```
[user@host ~]$ oc tag quay.io/keycloak/keycloak:19.0 keycloak:19.0
```

Use the `oc tag` command to update an image stream tag with a new source image reference. The following example changes the `keycloak:25.0` image stream tag to point to the `quay.io/keycloak/keycloak:25.0.3` image:

```
[user@host ~]$ oc tag quay.io/keycloak/keycloak:25.0.3 keycloak:25.0
```

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

25.0
  tagged from quay.io/keycloak/keycloak:25.0.3

  * 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
```

Inspecting an Image Stream Definition

To learn the structure of an image stream, you can inspect its YAML definition.

```
[user@host \~]$ oc get is/php -o yaml -n openshift
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
...output omitted...
  name: php 1
  namespace: openshift 2
...output omitted...
spec:
  lookupPolicy:
    local: false 3
  tags: 4
  - annotations:
      description: Build and run PHP 7.4 applications on UBI 8. For more information
        about using this builder image, including OpenShift considerations, see http
s://github.com/sclorg/s2i-php-container/blob/master/7.4/README.md.
...output omitted...
      version: "7.4"
    from:
      kind: DockerImage
      name: registry.ocp4.example.com:8443/ubi8/php-74:latest
    generation: 2
    importPolicy:
      importMode: Legacy
    name: 7.4-ubi8
    referencePolicy:
      type: Local
...output omitted...
status:
5 dockerImageRepository: image-registry.openshift-image-registry.svc:5000/openshift/php
  tags: 6
  - items:
    - created: "2025-09-12T09:34:46Z"
      dockerImageReference: registry.ocp4.example.com:8443/ubi8/php-74@sha256:32cb...f6
91 7
      generation: 2
      image: sha256:32cb...f691
    tag: 7.4-ubi8
  - items:
    - created: "2025-09-12T09:34:46Z"
      dockerImageReference: registry.ocp4.example.com:8443/ubi8/php-80@sha256:0194...af
f6
      generation: 2
      image: sha256:0194...aff6
    tag: 8.0-ubi8
 ...output omitted...
```

1 The name of the `ImageStream` object.

2 The project where the image stream resides.

3 The `lookupPolicy` controls whether this image stream can be used to satisfy short image names in pod specifications within the same project.

4. The `spec.tags` section defines the intended state, and lists each tag and its source image.
5. The `status.dockerImageRepository` field shows the path to this image stream in the internal RHOCP registry.
6. The `status.tags` section shows the current state, including the history of image digests for each tag.
7. The `dockerImageReference` in the `status` section records the immutable digest (SHA ID) of the image that was imported for the tag.

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 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 must manually 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 refresh automatically.

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 refresh regularly.

OpenShift can periodically verify whether a new image version is available. When OpenShift 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:25.0.3 keycloak:25.0 --scheduled
```

By default, OpenShift verifies the image every 15 minutes. The refresh 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 the event 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:25.0.3 keycloak:25.0 \
  --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:25.0`, 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 considers only the image streams with an enabled local lookup policy.

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:25.0
```

When you use a short name, OpenShift looks for a matching image stream in the current project. OpenShift considers only the image streams with an enabled local lookup policy. 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, which you can create by using the following command:

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

- `CronJob` objects, which you can create by using the following command:

  ```
  [user@host ~]$ oc create cronjob NAME --image IMAGE-STREAM-TAG \
    --schedule CRON-SYNTAX -- COMMAND
  ```

- `Pod` objects, which 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. The *Automatic Image Updates with OpenShift Image Change Triggers* chapter in this course discusses how updating an image stream tag triggers automatic rollouts of associated deployments.

REFERENCES

For more information about using image streams, refer to the *Managing Image Streams* chapter in the Red Hat OpenShift Container Platform 4.18 *Images* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/images/index#managing-image-streams

For more information about using image streams with deployments, refer to the *Using Image Streams with Kubernetes Resources* chapter in the Red Hat OpenShift Container Platform 4.18 *Images* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/images/index#using-imagestreams-with-kube-resources

For more information on how to configure container registries in OpenShift, refer to the *Image Controller Configuration Parameters* chapter in the Red Hat OpenShift Container Platform 4.18 *Images* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/images/index#images-configuration-parameters_image-configuration

How to Simplify Container Image Management in Kubernetes with OpenShift Image Streams