Chapter 7. Manage Application Updates

Abstract

Goal	Manage reproducible application updates and rollbacks of code and configurations.
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

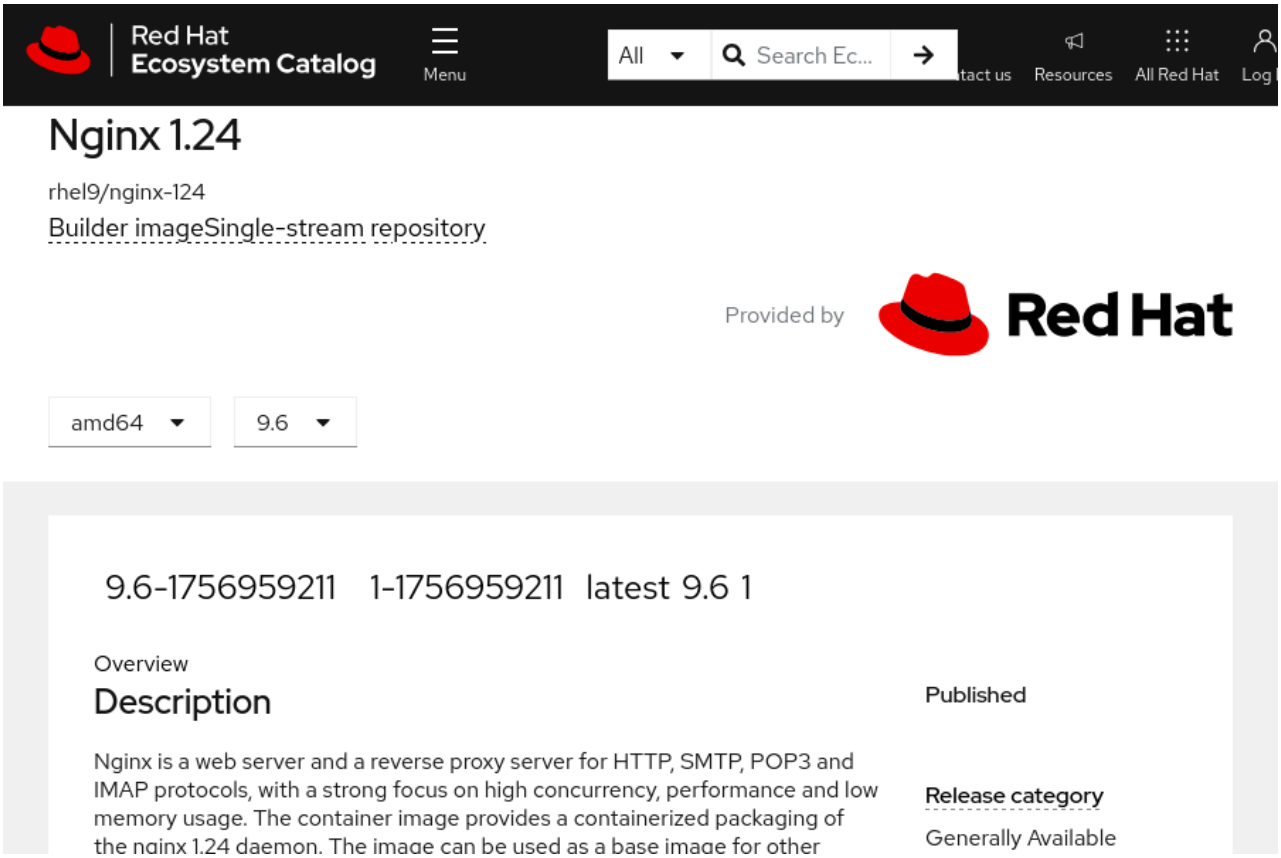
- Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes.

Kubernetes Image Tags

The full name of a container image is composed of several parts. For example, the `registry.redhat.io/rhel9/nginx-124:9.6` image name is composed of the following elements:

- The registry server is `registry.redhat.io`.
- The namespace is `rhel9`.
- The name is `nginx-124`. In this example, the name of the image includes the version of the software, which is Nginx version 1.20.
- The tag, which points to a specific version of the image, is `9.6`. 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 screenshot of the Red Hat Ecosystem Catalog at https://catalog.redhat.com/software/containers/explore lists the tags for the `rhel9/nginx-124` image:



In this case, the `9.6`, `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 `1-1756959211`.

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 *Secure Hash Algorithm* (SHA) 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. A developer is discouraged, although not prevented, 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 an 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.redhat.io/rhel9/nginx-124@sha256:126903359331eb9f2c87950f952442...
```

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.redhat.io/rhel9/nginx-124:9.6
> error: the image is a manifest list and contains multiple images - use --filter-by-os to select from:
>
>   OS              DIGEST
>   linux/amd64     sha256:c83789b0c7765e172bee5d36948e63e5db43248d8a3d69a3bb1...
>   linux/arm64/v8  sha256:dd239a25567a7b3887848b927bf9376fc76c4c4a57290afee55...
>   linux/ppc64le   sha256:0e9df5ef304708e1244697dc127816f07ece420bc47ea388cf1...
>   linux/s390x     sha256:0dd7ddd965e11c8a70fbcc66499b72de4e508afb0e4dfca48e8...
> ```

The following example displays the SHA ID for the image that the `9.6` tag currently points to:

```
[user@host ~]$ oc image info --filter-by-os linux/amd64 \
registry.redhat.io/rhel9/nginx-124:9.6
Name:      registry.redhat.io/rhel9/nginx-124:9.6
Digest:    sha256:c83789b0c7765e172bee5d36948e63e5db43248d8a3d69a3bb1... (1)
Manifest List: sha256:126903359331eb9f2c87950f952442db456750190956b9301d0...
Media Type:    application/vnd.oci.image.manifest.v1+json
...output omitted...
```

(1) The SHA image ID, or digest, for the specified image tag and architecture.

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.redhat.io/rhel9/nginx-124:9.6
IMAGE                              TAG   DIGEST              IMAGE ID    ...
registry.redhat.io/rhel9/nginx-124:  9.6   sha256:c837...71de   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:

```
[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.redhat.io/rhel9/nginx-124:9.6
        imagePullPolicy: IfNotPresent (1)
        name: nginx-124
...output omitted...
```

(1) The image pull policy is set to `IfNotPresent`.

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 references section 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"
    }
  }
}
```

NOTE

The `oc debug node` and `crictl` commands require cluster administrator privileges.

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` command 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 any custom needs. The garbage collector thresholds are configurable via the `KubeletConfig` Custom Resource (CR).

REFERENCES

`skopeo-inspect`(1) and `podman-system-prune`(1) man pages

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

For more information about image names, refer to the *Overview of Images* 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#about-containers-images-and-image-streams

For more information about pull policies, refer to the *Image Pull Policy Overview* section in the *Managing Images* 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-image-pull-policy-overview_image-pull-policy

For more information about garbage collection, refer to the *Understanding How Terminated Containers Are Removed Through Garbage Collection* section in the *Working with Nodes* chapter in the Red Hat OpenShift Container Platform 4.18 *Nodes* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/nodes/index#nodes-nodes-garbage-collection-containers_nodes-nodes-configuring