Find and Inspect Container Images

Objectives

- Find containerized applications in container registries and get information about the runtime parameters of supported and community container images.

Container Image Overview

A *container* is an isolated runtime environment where applications are executed as isolated processes. The isolation of the runtime environment ensures that applications do not interfere with other containers or system processes.

A *container image* contains a packaged version of your application, with all the necessary dependencies for the application to run. Images can exist without containers. However, containers depend on images, because containers use container images to build a runtime environment to execute applications.

Containers can be split into two similar but distinct concepts: *container images* and *container instances*. A *container image* contains immutable data that defines an application and its libraries. You can use container images to create *container instances*, which are running processes that are isolated by a set of kernel namespaces.

You can use each container image many times to create distinct container instances. These replicas can be split across multiple hosts. The application within a container is independent of the host environment.

Container Image Registries

Image registries are services that offer container images to download. Image creators and maintainers can store and distribute container images in a controlled manner to public or private audiences. Some examples of image registries include Quay.io, Red Hat Registry, Docker Hub, and Amazon ECR.

Red Hat Registry

Red Hat distributes container images through the Red Hat Ecosystem Catalog, https://catalog.redhat.com/, which provides a centralized searching utility. You can search the Red Hat Ecosystem Catalog for technical details about container images. The catalog hosts a large set of container images, including from major open source projects, such as Apache, MySQL, and Jenkins. Although many of the container images are publicly available, authentication to the website is required to access the container images that are not available in the public domain.

Because the Red Hat Ecosystem Catalog also contains software products other than container images, go to `https://catalog.redhat.com/software/containers/explore` to search for specific container images.

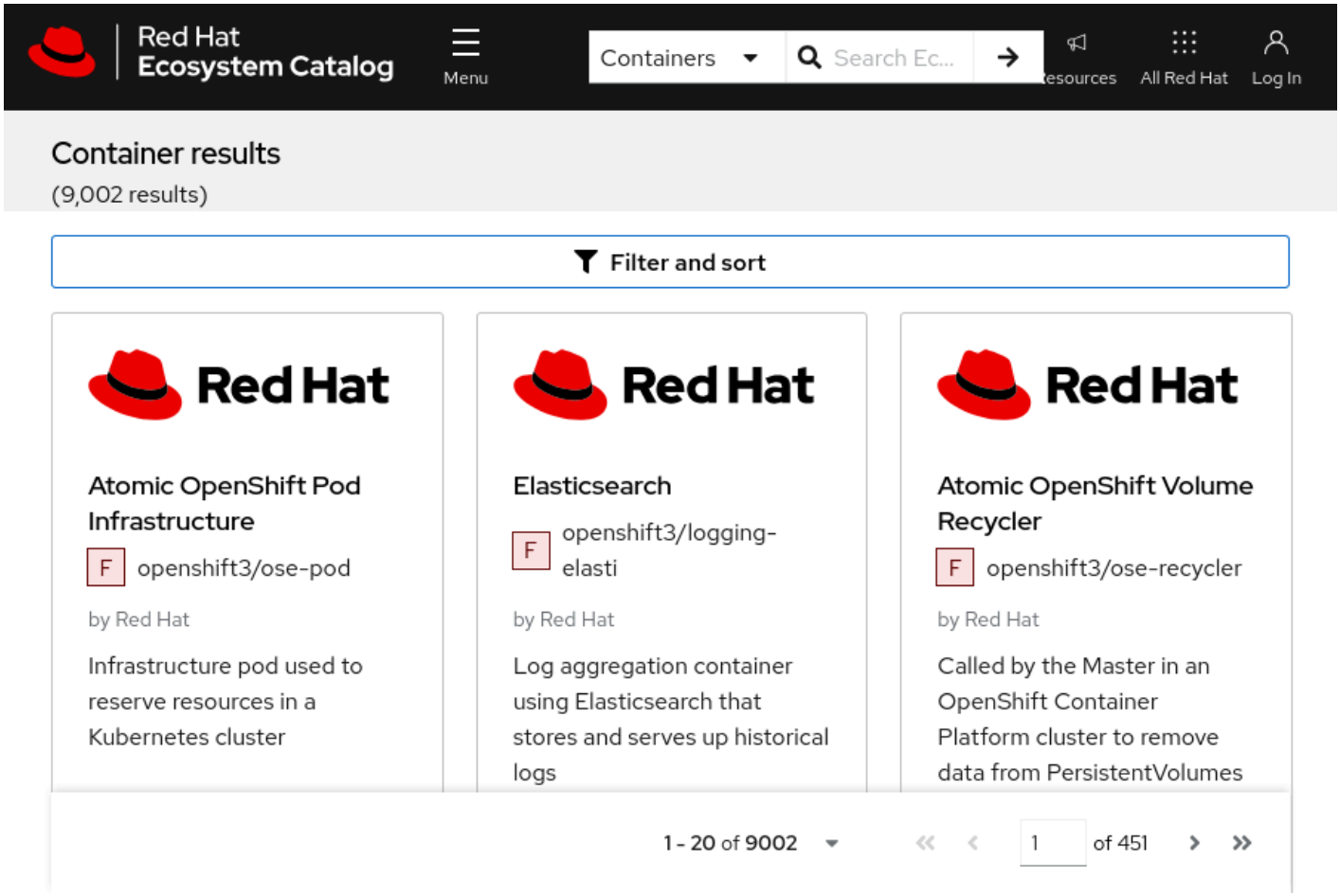

Figure 3.4: Red Hat Ecosystem Catalog

The details page of a container image includes relevant information, such as technical data, the installed packages within the image, or a security scan. You can navigate through these options by using the tabs on the website. You can also change the image version by selecting a specific tag.

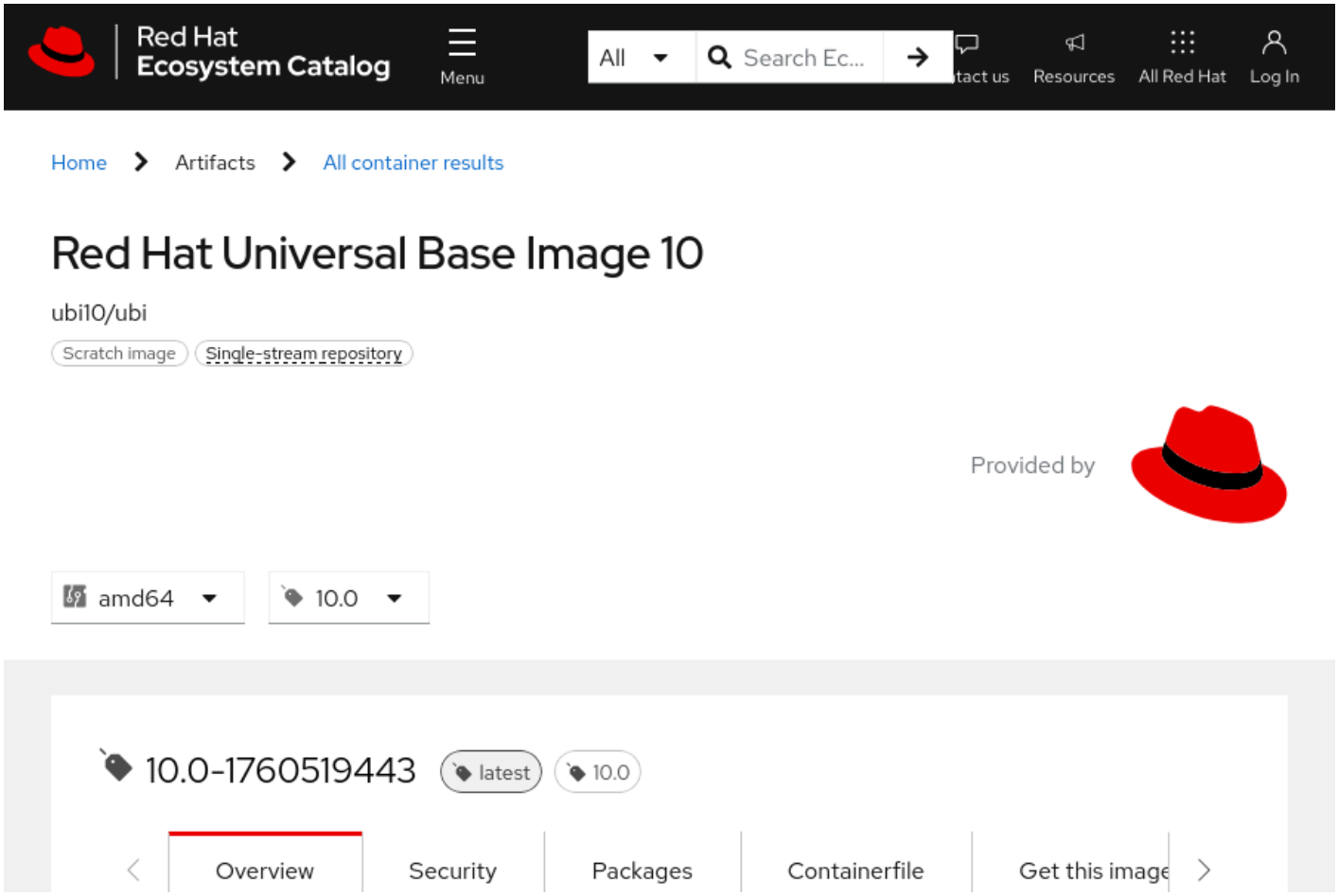

Figure 3.5: The Red Hat Universal Base Image 10 details page

The Red Hat internal security team validates all images in the container catalog. Red Hat rebuilds all components to avoid known security vulnerabilities.

Red Hat container images provide the following benefits:

- Trusted source: All container images use sources that Red Hat knows and trusts.
- Original dependencies: None of the container packages are altered, and they include only known libraries.
- Vulnerability-free: Container images are without known critical vulnerabilities in the platform components or layers.
- Runtime protection: All applications in container images run as non-root users to minimize the exposure surface to malicious or faulty applications.
- Red Hat Enterprise Linux (RHEL) compatible: Container images are compatible with all RHEL platforms, from bare metal to cloud.
- Red Hat support: Red Hat commercially supports the container images in the catalog.

> **NOTE**
>
> You must log in to the catalog website with a customer portal account or a Red Hat Developer account to use the stored container images in the registry.

Quay.io

Although the Red Hat Registry stores only images from Red Hat and certified providers, you can store your own images with Quay.io, which is another public image registry that Red Hat sponsors. Although storing public images in Quay is without cost, some options are available only for paying customers. Quay also offers an on-premise version of the product, to set up a localized image registry in your own servers.

Quay.io introduces features such as server-side image building, fine-grained access controls, and automatic scanning of images for known vulnerabilities.

Quay.io offers live images that creators regularly update. Quay.io users can create their namespaces, with fine-grained access control, and publish their created images to that namespace. Container Catalog users seldom push new images, and mostly consume trusted images from the Red Hat team.

Figure 3.6: The Quay.io welcome page

Private Registries

Image creators or maintainers can choose to make their images publicly available. However, other image creators might prefer to keep their images private, for the following reasons:

- Company privacy and protection of secrets
- Legal restrictions and laws
- Avoidance of publishing images in development

In some cases, private images are preferred. Private images are more secure than images in public registries. Private registries give image creators control over image placement, distribution, and usage.

Public Registries

Other public registries, such as Docker Hub and Amazon ECR, are also available for storing, sharing, and consuming container images. These registries can include official images that the registry owners or the registry community users create and maintain. For example, Docker Hub hosts a Docker Official Image of a WordPress container image. Although the `docker.io/library/wordpress` container image is a Docker Official Image, the container image is not supported by WordPress, Docker, or Red Hat. Instead, the Docker Community, which is a global group of Docker Hub users, supports and maintains the container image. Support for this container image depends on the availability and skills of the Docker Community users.

Consuming container images from public registries brings risks. For example, a container image might include malicious code or vulnerabilities, which can compromise the host system that executes the container image. A host system can also be compromised by public container images, because the images often execute by using the privileged `root` user. Additionally, the software in a container image might not be correctly licensed, or violate licensing terms.

Before you use a container image from a public registry, review and test the container image for your environment. Also ensure that you have the correct permissions to use the software in the container image.

Container Image Identifiers

Several objects provide identifying information about a container image.

`Registry`
A registry is a content server, such as `registry.access.redhat.com`, to store and share container images. A registry consists of one or more repositories that contain tagged container images.

`Name`

The name identifies the container image repository, and contains a string of letters, numbers, and special characters. The name refers to the directory or container repository within the container registry where the container image is stored.

For example, consider the fully qualified domain name (FQDN) of the `registry.access.redhat.com/ubi10/httpd-24:1-233` container image. The container image is in the `ubi10/httpd-24` repository in the `registry.access.redhat.com` container registry.

`ID or Hash`
The ID or hash is the SHA (Secure Hash Algorithm) code to uniquely locate, pull, or verify an image within the registry. The SHA image ID cannot change, and always references the same container image content. The ID or hash is the unique identifier for a container image. For example, the `sha256:4186a1ead13fc30796f951694c494e7630b82c320b81e20c020b3b07c888985b` image ID always refers to the `registry.access.redhat.com/ubi10/httpd-24:1` container image.

`Tag`
The tag is a label for a container image in a repository to distinguish the specific image from others, for version control. The `tag` comes after the image repository name and is delimited by a colon (:).

When an image tag is omitted, the floating tag, `latest`, is used as the default tag. A floating tag is an alias to another tag. In contrast, a fixed tag points to a specific container build. For example, the `registry.access.redhat.com/ubi10/httpd-24:1` container image, `1` is the fixed tag for the image, and corresponds to the floating `latest` flag until a later image is registered.

Container Image Components

A container image is composed of multiple components.

`Layers`
Container images are created from instructions. Each instruction adds a layer to the container image. Each layer consists of the differences between it and the following layer. The layers are then stacked to create a read-only container image.

`Metadata`
Metadata includes the instructions and documentation for a container image.

Container Image Instructions and Metadata

Container image layers consist of instructions, or steps, and metadata for building the image. You can override instructions during container creation to adjust the container image according to your needs. Some instructions can affect the running container, and other instructions are for informational purposes only.

The following instructions affect the state of a running container:

`ENV`

Defines the available environment variables in the container. A container image might include multiple `ENV` instructions. Any container can recognize additional environment variables that are not listed in its metadata.

`ARG`
Defines build-time variables, typically to make a customizable container build. Developers commonly configure the `ENV` instructions by using the `ARG` instruction. `ARG` is useful for preserving the build-time variables for run time.

`USER`
Defines the active user in the container. Later instructions run as this user. It is a good practice to define a user other than `root` for security purposes. OpenShift does not honor the user in a container image, for regular cluster users. Only cluster administrators can run containers (pods) with their chosen user IDs (UIDs) and group IDs (GIDs).

`ENTRYPOINT`
Defines the executable to run when the container is started.

`CMD`
Defines the command to execute when the container is started. This command is passed to the executable that the `ENTRYPOINT` instruction defines. Base images define a default `ENTRYPOINT` executable, which is usually a shell executable, such as Bash.

`WORKDIR`
Defines the current working directory within the container. Later instructions execute within this directory.
Metadata is used for documentation purposes, and does not affect the state of a running container. You can also override the metadata values during container creation.

The following metadata is for information only, and does not affect the state of the running container:

`EXPOSE`
Specifies the network port that the application binds to within the container. This metadata does not automatically bind the port on the host, and is used only for documentation purposes.

`VOLUME`
Defines where to store data outside the container. The value shows the path where your container runtime mounts the directory inside the container. More than one path can be defined to create multiple volumes.

`LABEL`
Defines a key-value pair in the metadata of the image for organization and image selection.

Container engines are not required to honor metadata in a container image, such as `USER` or `EXPOSE`. A container engine can also recognize additional environment variables that are not listed in the container image metadata.

Base Images

A *base image* is the image that your resulting container image is built on. Your chosen base image determines the Linux distribution and any of the following components:

- Package manager
- `Init` system
- File-system layout
- Preinstalled dependencies and runtimes

The base image can also influence factors such as image size, vendor support, and processor compatibility.

Red Hat provides enterprise-grade container images that are engineered to be the base operating system layer for your containerized applications. These container images are intended as a common starting point for containers, and are known as *universal base images* (UBI). Red Hat UBI container images are *Open Container Initiative* (OCI) compliant images that contain portions of Red Hat Enterprise Linux. UBI container images include a subset of RHEL content. They provide a set of prebuilt runtime languages, such as Python and Node.js, and associated DNF repositories to add application dependencies. UBI-based images can be distributed without cost or restriction. These images can be deployed to both Red Hat and non-Red Hat platforms, and be pushed to your chosen container registry.

A Red Hat subscription is not required to use or distribute UBI-based images. However, Red Hat provides full support for containers that are built on UBI only if the containers are deployed to a Red Hat platform, such as an RHOCP cluster or RHEL.

Red Hat provides four UBI variants: `standard`, `init`, `minimal`, and `micro`. All UBI variants and UBI-based images use RHEL at their core and are available from the Red Hat Container Catalog. The main differences are as follows:

Standard
This image is the primary UBI, which includes DNF, systemd, and utilities such as `gzip` and `tar`.

Init
This image simplifies running multiple applications within a single container by managing them with systemd.

Minimal
This image is smaller than the `init` image and provides nice-to-have features. This image uses the `microdnf` minimal package manager instead of the full-sized version of DNF.

Micro
This image is the smallest available UBI, and includes only the minimum packages. For example, this image does not include a package manager.

Additionally, Red Hat provides language-specific runtime images, such as for python and node.js.

Inspecting and Managing Container Images

Various tools can inspect and manage container images, including the `oc image` command and Skopeo.

Skopeo

Skopeo is a tool to inspect and manage remote container images. With Skopeo, you can copy and synchronize container images from different container registries and repositories. You can copy an image from a remote repository and save it to a local disk. If you have the appropriate repository permissions, then you can delete an image from container registry. You can use Skopeo to inspect the configuration and contents of a container image, and to list the available tags for a container image. Unlike other container image tools, Skopeo can execute without a privileged account such as `root`. Skopeo does not require a running daemon to execute various operations.

Skopeo is executed with the `skopeo` command-line utility, which you can install with various package managers, such as DNF, Brew, or APT. The `skopeo` utility might already be installed on some Linux-based distributions. You can install the `skopeo` utility on Fedora, CentOS Stream 8 and later, and Red Hat Enterprise Linux 8 and later systems, by using the DNF package manager.

```
[user@host ~]$ sudo dnf -y install skopeo
```

The `skopeo` utility is currently not available as a packaged binary for Windows-based systems. However, the `skopeo` utility is available as a container image from the `quay.io/skopeo/stable` container repository. For more information about the Skopeo container image, refer to the `skopeoimage` overview guide in the Skopeo repository at https://github.com/containers/image_build/tree/main/skopeo

You can alternatively build `skopeo` from source code in a container, or build it locally without using a container. Refer to the installation guide in the Skopeo repository for more information about installing or building Skopeo from source code at https://github.com/containers/skopeo/blob/main/install.md#container-images

The `skopeo` utility provides commands for managing and inspecting container images and container image registries. For container registries that require authentication, you must first log in to a registry before you can execute additional `skopeo` commands.

```
[user@host ~]$ skopeo login quay.io
```

> **NOTE**
>
> OpenShift clusters are typically configured with registry credentials. When a pod is created from a container image in a remote repository, OpenShift authenticates to the container registry with the configured registry credentials, and then *pulls* the image, to copy from the remote repository. Because OpenShift automatically uses

the registry credentials, you typically do not need to manually authenticate to a container registry when you create a pod. By contrast, the `oc image` command and the `skopeo` utility require you first to log in to a container registry.

After you log in to a container registry (if required), you can execute additional `skopeo` commands against container images in a repository. When you execute a `skopeo` command, you must specify the transport and the repository name. A *transport* is the mechanism to transfer container images between locations. Two common transports are `docker` and `dir`. The `docker` transport is used for container registries, and the `dir` transport is used for local directories.

The `oc image` command and other tools default to the `docker` transport, and so you do not need to specify the transport when executing commands. However, the `skopeo` utility does not define a default transport; you must specify the transport with the container image name. Most `skopeo` commands use the `skopeo` *`command [command options] transport://IMAGE-NAME`* format. For example, the following `skopeo list-tags` command lists all available tags in a `registry.access.redhat.com/ubi10/httpd-24` container repository by using the `docker` transport:

```
[user@host ~]$ skopeo list-tags docker://registry.access.redhat.com/ubi10/httpd-24
{
    "Repository": "registry.access.redhat.com/ubi10/httpd-24",
    "Tags": [
        "1",
...output omitted...
        "latest"
...output omitted...
```

The `skopeo` utility includes other commands for container image management.

`skopeo inspect`
View detailed information for an image name, such as environment variables and available tags. Use the `skopeo inspect` *`[command options] transport://IMAGE-NAME`* command format. You can include the `--config` flag to view the configuration, metadata, and history of a container repository. The following example retrieves the configuration information for the `registry.access.redhat.com/ubi10/httpd-24` container repository:

```
[user@host ~]$ skopeo inspect --config docker://registry.access.redhat.com/ubi10/httpd-24
...output omitted...
    "config": {
        "User": "1001",
        "ExposedPorts": {
            "8080/tcp": {},
            "8443/tcp": {}
        },
        "Env": [
...output omitted...
            "HTTPD_MAIN_CONF_PATH=/etc/httpd/conf",
            "HTTPD_MAIN_CONF_MODULES_D_PATH=/etc/httpd/conf.modules.d",
            "HTTPD_MAIN_CONF_D_PATH=/etc/httpd/conf.d",
            "HTTPD_TLS_CERT_PATH=/etc/httpd/tls",
            "HTTPD_VAR_RUN=/var/run/httpd",
            "HTTPD_DATA_PATH=/var/www",
            "HTTPD_DATA_ORIG_PATH=/var/www",
            "HTTPD_LOG_PATH=/var/log/httpd"
        ],
        "Entrypoint": [
            "container-entrypoint"
        ],
        "Cmd": [
            "/usr/bin/run-httpd"
        ],
        "WorkingDir": "/opt/app-root/src",
...output omitted...
    }
...output omitted...
```

skopeo copy

Copy an image from one location or repository to another. Use the `skopeo copy` *`transport`*`://`*`SOURCE-IMAGE`* *`transport`*`://`*`DESTINATION-IMAGE`* format. For example, the following command copies the `quay.io/skopeo/stable:latest` container image to the `skopeo` repository in the `registry.example.com` container registry:

```
[user@host ~]$ skopeo copy docker://quay.io/skopeo/stable:latest \
docker://registry.example.com/skopeo:latest
```

skopeo delete

Delete a container image from a repository. Use the `skopeo delete` *`[command options]`* *`transport`*`://`*`IMAGE-NAME`* format. The following command deletes the `skopeo:latest` image from the `registry.example.com` container registry:

```
[user@host ~]$ skopeo delete docker://registry.example.com/skopeo:latest
```

skopeo sync

Synchronize one or more images from one location to another. Use this command to copy all container images from a source to a destination. Use the `skopeo sync` *`[command options]`* `--src` *`transport`* `--dest` *`transport`* *`SOURCE DESTINATION`* format. The following command synchronizes the `registry.access.redhat.com/ubi10/httpd-24` container repository to the `registry.example.com/httpd-24` container repository:

```
[user@host ~]$ skopeo sync --src docker --dest docker \
registry.access.redhat.com/ubi10/httpd-24 registry.example.com/httpd-24
```

Registry Credentials

Some registries require users to authenticate before accessing container images and metadata. For example, Red Hat containers that are based on RHEL typically require authenticated access:

```
[user@host ~]$ skopeo inspect docker://registry.redhat.io/rhel10/httpd-24
FATA[0000] Error parsing image name "docker://registry.redhat.io/rhel10/httpd-24":
unable to retrieve auth token: invalid username/password: unauthorized: Please login to
the Red Hat Registry using your Customer Portal credentials. Further instructions can b
e found here: https://access.redhat.com/RegistryAuthentication
```

You might choose a different image that does not require authentication, such as the UBI 10 image:

```
[user@host ~]$ skopeo inspect docker://registry.access.redhat.com/ubi10:latest
{
    "Name": "registry.access.redhat.com/ubi10",
    "Digest": "sha256:b515...",
    "RepoTags": [
        "10.0",
        "10.0-1745487123",
...output omitted....
```

Alternatively, you must execute the `skopeo login` command for the registry before you can access protected images.

```
[user@host ~]$ skopeo login registry.redhat.io
Username: YOUR_USER
Password: YOUR_PASSWORD
Login Succeeded!
```

Skopeo stores the credentials in the `${XDG_RUNTIME_DIR}/containers/auth.json` file, where `${XDG_RUNTIME_DIR}` refers to a directory that is specific to the current user. The credentials are encoded in Base64 format:

```
[user@host ~]$ cat ${XDG_RUNTIME_DIR}/containers/auth.json
{
        "auths": {
                "registry.redhat.io": {
                        "auth": "dXNlcjpodW50ZXIy"
                }
        }
}
[user@host ~]$ echo -n dXNlcjpodW50ZXIy | base64 -d
user:hunter2
```

> **NOTE**
>
> For security reasons, the `skopeo login` command does not show your password in the interactive session. Although you do not see what you are typing, Skopeo registers every keystroke. After typing your full password in the interactive session, press **Enter** to start the login.

The `oc image` Command

The OpenShift command-line interface provides the `oc image` command. You can use this command to inspect, configure, or retrieve information about container images.

The `oc image info` command inspects and retrieves information about a container image. You can use the `oc image info` command to identify the ID or hash SHA and to list the image layers of a container image. You can also review container image metadata, such as environment variables, network ports, and commands. If a container image repository provides a container image in multiple architectures, such as AMD64 or ARM64, then you must include the `--filter-by-os` tag. For example, you can execute the following command to retrieve information about the `registry.access.redhat.com/ubi10/httpd-24:10.0` container image that is based on the AMD64 architecture:

```
[user@host ~]$ oc image info registry.access.redhat.com/ubi10/httpd-24 \
--filter-by-os amd64
Name:          registry.access.redhat.com/ubi10/httpd-24:10.0
Digest:        sha256:b765...
...output omitted...
```

The `oc image` command provides more options for managing container images.

`oc image append`
Use this command to add layers to container images, and then push the container image to a registry.

`oc image extract`
You can use this command to extract or copy files from a container image to a local disk. Use this command to access the contents of a container image without first running the image as a container. A running container engine is not required.

`oc image mirror`
Copy, or *mirror*, container images from one container registry or repository to another. For example, you can use this command to mirror container images between public and private registries. You can also use this command to copy a container image from a registry to a disk. The command mirrors the HTTP structure of a container registry to a directory on a disk. The directory on the disk can then be served as a container registry.

Running Containers as Root

Running containers as the root user is a security risk, because an attacker could exploit the application, access the container, and exploit vulnerabilities to escape from the containerized environment and access the host system. Attackers might escape the containerized environment by exploiting bugs and vulnerabilities that are typically in the kernel or the container runtime.

Traditionally, when an attacker gains access to the container file system by using an exploit, the `root` user inside the container corresponds to the `root` user on the host system. If an attacker escapes the container isolation, then they have access to elevated privileges on the host system, which provides a vector of attack that could cause damage.

Containers that do not run as the `root` user might prove unsuitable for use in your application because of the following limitations:

Non-trivial Containerization
Some applications might require the `root` user. Depending on the application architecture, some applications might not be suitable for non-privileged containers, or might require additional experience to containerize.

For example, applications such as HTTPd and Nginx start a bootstrap process and then create a process with a non-privileged user, which interacts with external users. Such applications are non-trivial to containerize for non-privileged use.

Red Hat provides containerized versions of HTTPd and Nginx that do not require `root` privileges for production usage. You can peruse these validated containers in the Red Hat container registry (https://catalog.redhat.com/software/containers/explore).

Required Use of Privileged Utilities
Non-privileged containers cannot bind to privileged ports, such as the `80` or `443` ports. Red Hat advises against using privileged ports. Instead, use port forwarding to avoid the use of privileged ports within container definitions.

Similarly, non-privileged containers cannot use the `ping` utility by default, because it requires elevated privileges to establish raw sockets.

REFERENCES

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