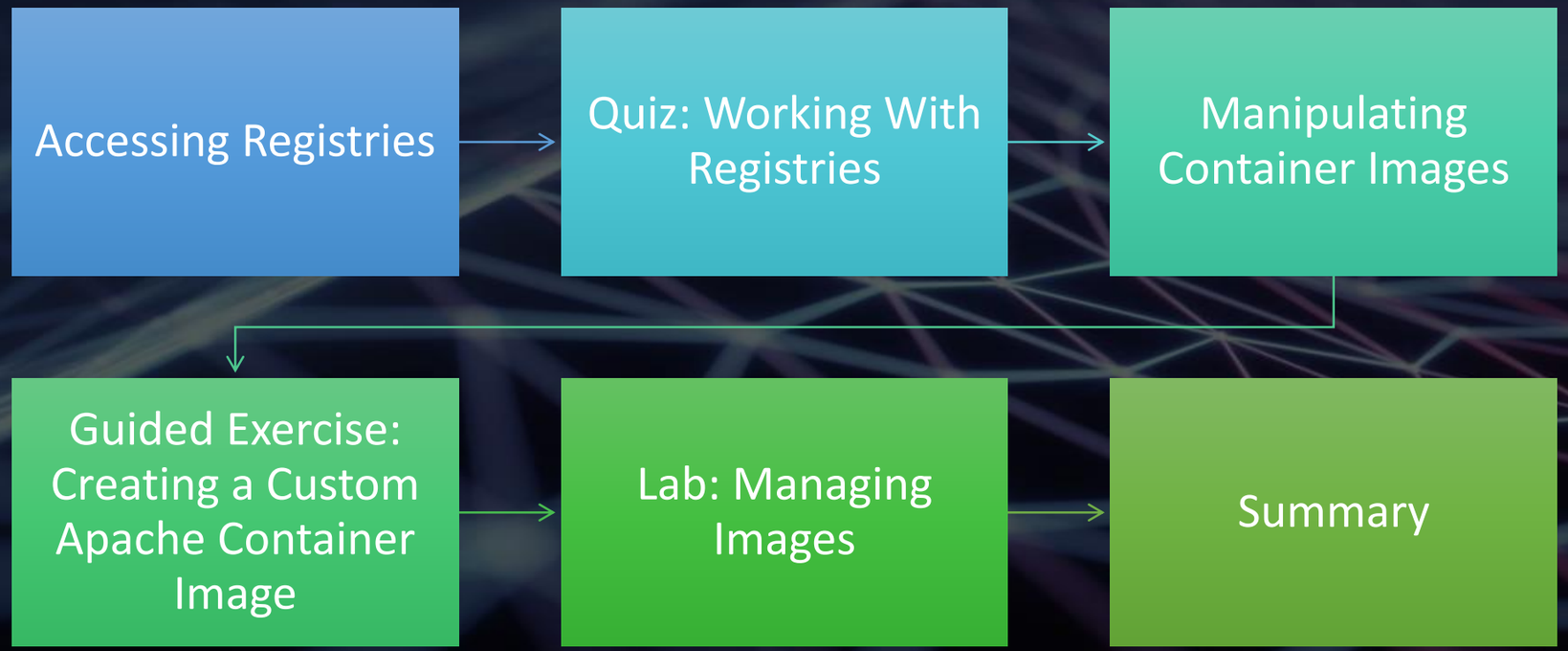
Container Images

Chapter objectives



Accessing Registries

After completing this section, you will be able to:

Search and pull
- images from remote registries

List advantages of
- using certified public registry

Customize the configuration to
- access alternative container image registries.

List downloaded images
- from a registry to the local file system.

Manage
- tags to pull tagged images.

Public Registries

- Containing container images to download
- Store and distribute images
- Podman can search images from public or private registries
- Example of public registries:
  - Red Hat Container Catalog
  - Quay.io
  - Docker Hub
  - Google Container Registry
  - and etc

Red Hat Container Catalog

- Trusted Source: All container images comprise sources known and trusted by Red Hat.
- Original dependencies: None of the container packages have been tampered with, and only include known libraries.
- Vulnerability-free: Container images are free of known vulnerabilities in the platform components or layers.
- Runtime protection: All applications in container images run as non-root users, minimizing 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 complete stack.

QUAY.io

- Public registry managed by Red Hat team
- Server-side image building
- Automatic scanning for known vulnerabilities
- Offers live upload by Image creator
- Fine-grained access controls with namespace

Private Registries

- Company privacy and secret protection.
- Legal restrictions and laws.
- Avoidance of publishing images in development.
- Total control about their image's placement, distribution and usage.

Configuring Registries in Podman

- Update /etc/containers/registries.conf

```
$vi /etc/containers/registries.conf

…output omitted…

[registries.search]

registries = [“registry.access.redhat.com”, “quay.io”, “docker.io”]


[registries.insecure]

registries = [“localhost:5000”]
```

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

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

It defines build-time variables, typically to make a customizable container build. Developers commonly configure the ENV instructions by using the ARG instruction. It 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 ID (UIDs)* and *group IDs (GIDs)*.

ENTRYPOINT

It defines the executable to run when the container is started.

CMD

It 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.

Container Image Instructions and Metadata

following instructions affect the state of a running container:

WORKDIR

It sets 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

It indicates 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

It 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

Adds a key-value pair to the metadata of the image for organization and image selection.

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 Container Images

- Known as universal base images (UBI)
- Engineered to be base operating system layer
- Open Container Initiative (OCI) compliant images
- Include subset of RHEL content, prebuilt runtime languages, associated DNF repositories
- No cost , no restriction nor subscription required to use and distribute it
- Comes in 4 variants:
  - Micro: Only minimum packages. Do not have package manager
  - Minimal: Bigger than micro. Provides nice-to-have features. Uses microdnf minimal package manager
  - Init: Bigger than minimal + include systemd
  - Standard: primary UBI, include many utilizies such as DNF, systemd, gzip, tar and etc

Inspecting and Managing Container Images

Skopeo

- open source utility for manipulating, inspecting, signing, transferring container images between container registries and repositories
- Copy image from remote repository to local disk
- Delete images
- List available tags
- Can execute without privileged root account
- Do not require running daemon to execute various operations
- Install skopeo utilities

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

- Publicly available as image - quay.io/skopeo/stable
- More information or guides - github.com/containers/skopeo

Skopeo inspect

- View low-level information for an image

```
[user@host ~]$ skopeo inspect --config
 docker://registry.access.redhat.com/ubi9/httpd-24
...output omitted...
    "config": {
        "User": "1001",
        "ExposedPorts":
            "8080/tcp":
            "8443/tcp":
        },
        "Env": [
...output omitted...
            "HTTPD_MAIN_
            "HTTPD_MAIN_
                "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 list-tags

- List all available tags of a container image

```
[user@host ~]$ skopeo list-tags docker://registry.access.redhat.com/ubi9/httpd-24
{
    "Repository": "registry.access.redhat.com/ubi9/httpd-24",
    "Tags": [
        "1-229",
        "1-217.1666632462",
        "1-201",
        "1-194.1655192191-source",
        "1-229-source",
        "1-194-source",
        "1-201-source",
        "1-217.1664813224",
        "1-217.1664813224-source",
        "1-210-source",
        "1-233.1669634588",
        "1",
```

Skopeo copy

- The syntax

skopeo copy **SRC_IMAGE DEST_IMAGE**

[--src-creds username[:password]]

[-dest-creds username[:password]]

[--src-tls-verify true|false]

[--dest-tls-verify true|false]

SRC_IMAGE | DEST_IMAGE

docker://quay.io/jason_wong76/webserver:latest

containers-storage:localhost/myimage:1.0

dir:myfolder

oci:myfolder:1.0

- Copy image from repository to another

$ skopeo copy docker://quay.io/skopeo/stable:latest docker://registry.example.com/skopeo:

Skopeo copy

- Copy image from local disk to private repository

```
$ read -s passwd
< enter password to the private registry>

$ skope copy --dest-creds jason.wong76:$passwd \
    containers-storage:localhost/jason/local-ubuntu:1.0 \
    docker://quay.io/jason.wong76/ubuntu:1.0
```

- Copy image from specific folder in local disk to private repository

```
$ podman login -u jason.wong76 -p $passwd quay.io

$ skopeo oci:/home/student/DO288/labs/external-registry/ubi-sleep \
   docker://quay.io/jason.wong76/ubi-sleep:1.0
```

Skopeo delete

skopeo delete

Delete a container image from a repository. You must 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

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. The command uses the `skopeo sync` *`[command options]`* `--src` *`transport`* `--dest` *`transport SOURCE DESTINATION`* format. The following command synchronizes the `registry.access.redhat.com/ubi8/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/ubi8/httpd-24 registry.example.com/httpd-24
```

Registry Credentials

- Requires authentication : registry.redhat.io

```
[user@host ~]$ skopeo inspect docker://registry.redhat.io/rhel8/httpd-24
FATA[0000] Error parsing image name "docker://registry.redhat.io/rhel8/
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 be found here: https://access.redhat.com/
RegistryAuthentication
```

- Do not require authentication : registry.access.redhat.com

```
[user@host ~]$ skopeo inspect docker://registry.access.redhat.com/ubi8:latest
{
    "Name": "registry.access.redhat.com/ubi8",
    "Digest": "sha256:70fc...1173",
    "RepoTags": [
        "8.7-1054-source",
        "8.6-990-source",
        "8.6-754",
        "8.4-203.1622660121-source",
...output omitted....
```

Skopeo login

- Log into private registries

```
[user@host ~]$ skopeo login registry.redhat.io
Username: YOUR_USER
Password: YOUR_PASSWORD
Login Succeeded!
[user@host ~]$ skopeo list-tags docker://registry.redhat.io/rhel8/httpd-24
{
    "Repository": "registry.redhat.io/rhel8/httpd-24",
    "Tags": [
        "1-166.1645816922",
        "1-209",
        "1-160-source",
        "1-112",
...output omitted...
```

Skopeo login

- Upon successful logins, skopeo encodes and stores credentials in ${XDG_RUNTIME_DIR}/containers/auth.json
- View the credential

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

oc image info Command

- Inspects and retrieve information about container image.
- List image layers
- Use --filter-by-os to filter list platform specific such as amd64 or arm64 or x86_64

```
~]$ oc image info registry.access.redhat.com/ubi
-by-os amd64
                registry.access.redhat.com/ubi9/httpd-24:1-2
:               sha256:4186...985b
tput omitted...
e Size:         130.8MB in 3 layers
rs:             79.12MB sha256:d74e...1cad
                17.32MB sha256:dac0...a283
                34.39MB sha256:47d8...5550
:               linux
ch:             amd64
ntrypoint:      container-entrypoint
ommand:         /usr/bin/run-httpd
orking Dir:     /opt/app-root/src
ser:            1001
poses Ports:    8080/tcp, 8443/tcp
vironment:      container=oci
output omitted...
                HTTPD_CONTAINER_SCRIPTS_PATH=/usr/share/conta
                HTTPD_APP_ROOT=/opt/app-root
                HTTPD_CONFIGURATION_PATH=/opt/app-root/etc/ht
                HTTPD_MAIN_CONF_PATH=/etc/httpd/conf
                HTTPD_MAIN_CONF_MODULES_D_PATH=/etc/httpd/co
```

More on oc image command

`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.

Using Rootless Containers

After completing this section, you should be able to:

- Explain the differences between running root and rootless containers.
- Describe the advantages and disadvantages of each case.

Evolution of Container Usage

- Privilege user
  - Needed to create resources such network interfaces, or mount file system
- Bad practice to use privilege user
- Security bugs, exposure to all possible attacks and CVE
- Better practice: Do not require privileged user
- Many community images: still require root to run
- Podman and OpenShift: Default start rootless container.
- Docker announced rootless mode

Understanding Rootless Containers

User Namespaces

- Containers isolate from host using namespace
- Rootless containers
- Container runs as root, appear under different ID to host
- Success attack on container, will only have capabilities of unprivileged user outside of container

Networking

- Rootfull container have full access to SDN
- Rootless container have limited access to SDN

Storage

- Rootfull container uses overlay2 (layers)
- Rootless container barred from mounting overlay file systems

Pros and Cons : Rootless Containers

- Do not require root privileges to run
- Pros:
  - ✓ Root privilege within container, non-root privilege on host
  - ✓ New security layer
  - ✓ Multiple unprivileged users
  - ✓ Allows isolation inside nested containers
- Cons:
  - × Dropped capabilities
  - × Binding to port less than 1024
  - × Volume mounting

Guided Exercise: Creating a Custom Apache Container Image

You should be able to:

- Create a custom Apache container image
- Tag an image
- Push the tagged image to Quay.io registry