Lab 4

Chapter 4 and Chapter 5

Chapter 4: Managing Container Images

What is Image registries?

Public registries vs private registries?

Give examples for public registries?

Which commands used to search in registries?

Image registries

Image registries are services offering container images to download. They allow image creators and maintainers to store and distribute container images to public or private audiences.

Public registries: images that are publicly available to be downloaded. Examples: Quay.io , Red Hat Container Catalog and docker hub

Private registries: Private registries give image creators the control about their images placement, distribution and usage.

```
podman search {image name}
```

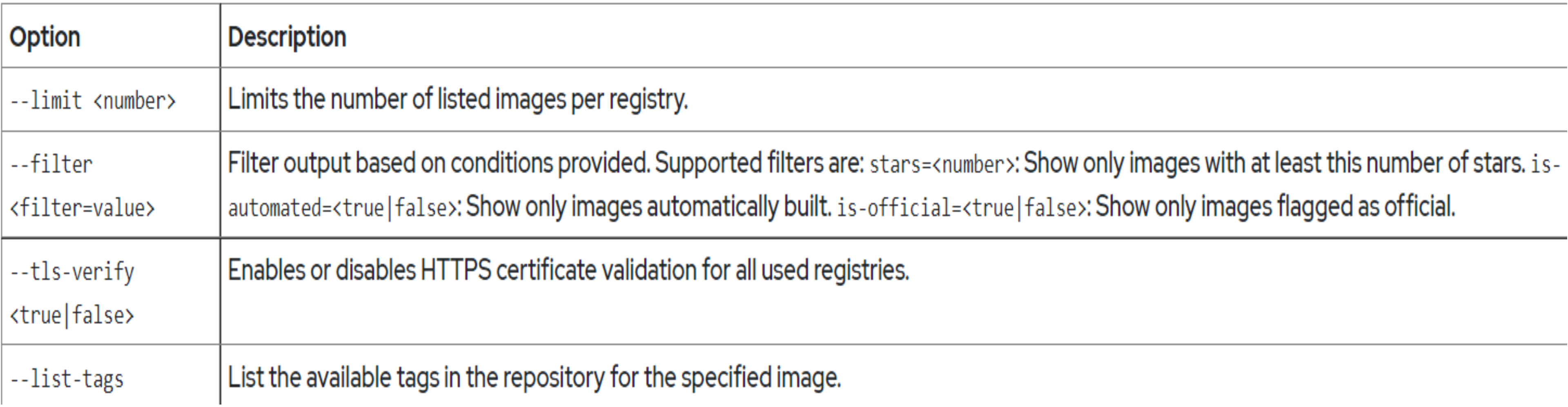
Configuring Registries in Podman

To configure registries for the podman command, you need to update the /etc/containers/registries.conf file

```
[registries.search]
registries = ["registry.access.redhat.com", "quay.io"]
```

```
[user@host ~]$ podman search [OPTIONS] <term>
```

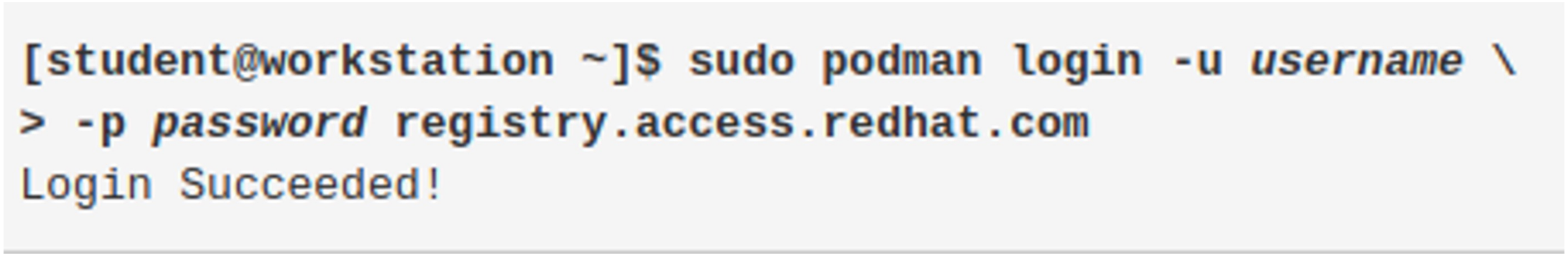
Extra options when searching commands



Option	Description
`--limit <number>`	Limits the number of listed images per registry.
`--filter <filter=value>`	Filter output based on conditions provided. Supported filters are: `stars=<number>`: Show only images with at least this number of stars. `is-automated=<true|false>`: Show only images automatically built. `is-official=<true|false>`: Show only images flagged as official.
`--tls-verify <true|false>`	Enables or disables HTTPS certificate validation for all used registries.
`--list-tags`	List the available tags in the repository for the specified image.

Registry Authentication

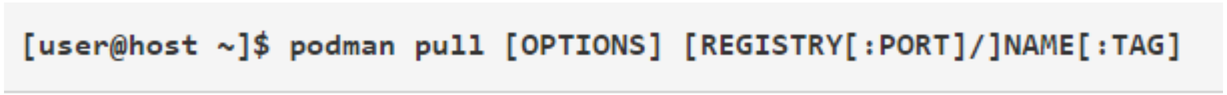
Some container image registries require access authorization. The podman login command allows username and password authentication to a registry:



```
[student@workstation ~]$ sudo podman login -u username \
> -p password registry.access.redhat.com
Login Succeeded!
```

Pulling Images

To pull container images from a registry, use the podman pull command:

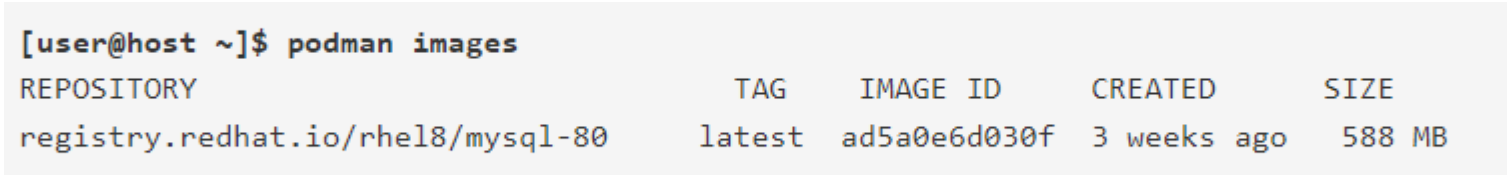


```
[user@host ~]$ podman pull [OPTIONS] [REGISTRY[:PORT]/]NAME[:TAG]
```

```
podman pull quay.io/bitnami/nginx
```

Listing Local Copies of Images

Any container image downloaded from a registry is stored locally on the same host where the podman command is executed. This behavior avoids repeating image downloads and minimizes the deployment time for a container. Podman also stores any custom container images you build in the same local storage.



```
[user@host ~]$ podman images
REPOSITORY                                  TAG     IMAGE ID      CREATED       SIZE
registry.redhat.io/rhel8/mysql-80         latest  ad5a0e6d030f  3 weeks ago    588 MB
```

Image Tags

An image tag is a mechanism to support multiple releases of the same image. This feature is useful when multiple versions of the same software are provided

registry_name/user_name/image_name:tag

Pull image: podman pull rhscl/mysql-57-rhel7:5.7

Start a container: podman run rhscl/mysql-57-rhel7:5.7

Q: which command is used to get an image?

Q: which command is used to list local images?

Q: which command is used to create a container from an image?

Q: what is the image tag?

Saving and Loading Images

2 ways to manage image containers:

1- Save the container image to a .tar file.

2- Publish (push) the container image to an image registry (not the best way …Why?).

Save and Load images

The following example saves the previously downloaded MySQL container image from the Red Hat Container Catalog to the mysql.tar file:

Save image:

```
podman pull registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7
```

```
podman save  -o mysql.tar registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7
```

Load image:

```
podman load -i mysql.tar
```

Q: Which command is used to delete images?

Deleting Images

- An image can be referenced using its name or its ID for removal purposes.
- Podman cannot delete images while containers are using that image.
- You must stop and remove all containers using that image before deleting it.
- Podman keeps any image downloaded in its local storage, even the ones currently unused by any container. However, images can become outdated, and should be subsequently replaced.

Deleting Images

- The **rmi** subcommand has the **--force** option. This option forces the removal of an image even if that the image is used by several containers or these containers are running.
- Podman stops and removes all containers using the forcefully removed image before removing it.

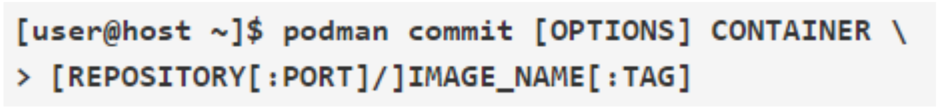
Delete all images: podman rmi -a -f

Delete one image: podman rmi IMAGE

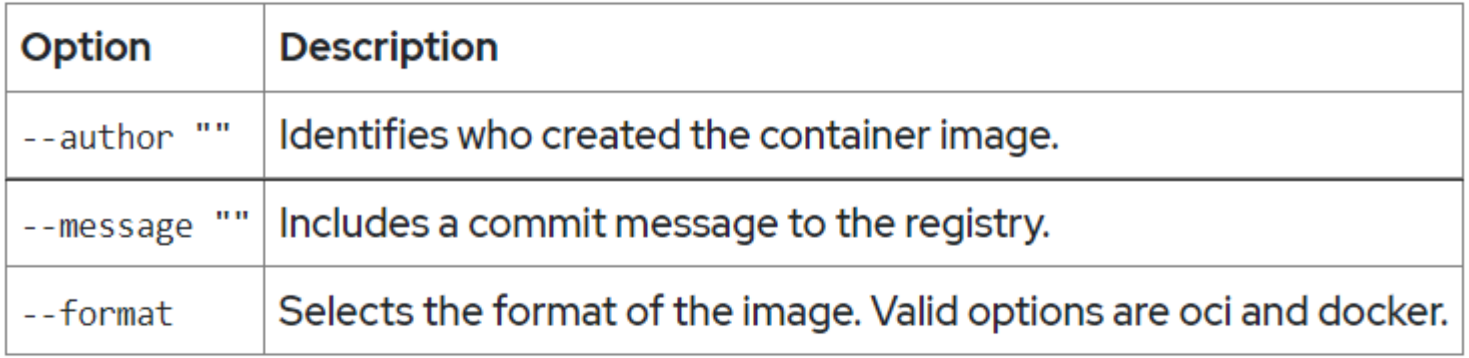
Modifying Images

- all container images should be built using a **Dockerfile**, in order to create a clean, lightweight set of image layers without log files, temporary files, or other artifacts created by the **container customization.**
- As an alternative approach to creating new images ---> **change a running container in place and save its layers to create a new container image.**
- The podman **commit** command provides this feature.

Podman commit



```
[user@host ~]$ podman commit [OPTIONS] CONTAINER \
> [REPOSITORY[:PORT]/]IMAGE_NAME[:TAG]
```

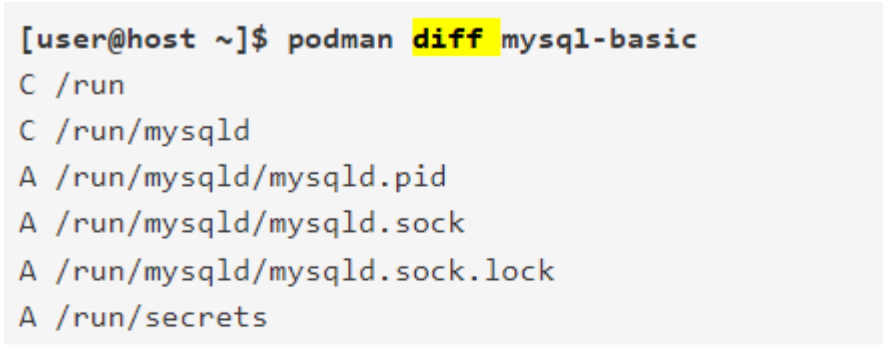


Option	Description
`--author ""`	Identifies who created the container image.
`--message ""`	Includes a commit message to the registry.
`--format`	Selects the format of the image. Valid options are oci and docker.

sudo podman commit -a 'Your Name' official-nginx-dev do180/mynginx:v1.0

Diff subcommand

- To identify which files were changed, created, or deleted since the container was started, use the diff subcommand.
- This subcommand only requires the container name or container ID.



```
[user@host ~]$ podman diff mysql-basic
C /run
C /run/mysqld
A /run/mysqld/mysqld.pid
A /run/mysqld/mysqld.sock
A /run/mysqld/mysqld.sock.lock
A /run/secrets
```

Guided Exercise: Creating a Custom Apache Container Image

- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch04s04

```
[student@workstation ~]$ lab image-operations start
```

Lab: Managing Images

- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch04s05

```
[student@workstation ~]$ lab image-review start
```

Chapter 5: Creating Custom Container Images

- Designing Custom Container Images
- Building Custom Container Images with Containerfiles

Dockerfile / Containerfile

Dockerfile is used to create a custom Docker image, in other words to define your custom environment to be used in a Docker container.

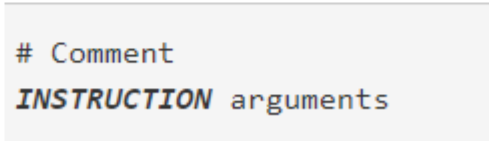
The Dockerfile contains a list of instructions that Docker will execute when you issue the docker build command

To create a custom image

1. Create a working directory
2. Write the Dockerfile
3. Build the image with Podman

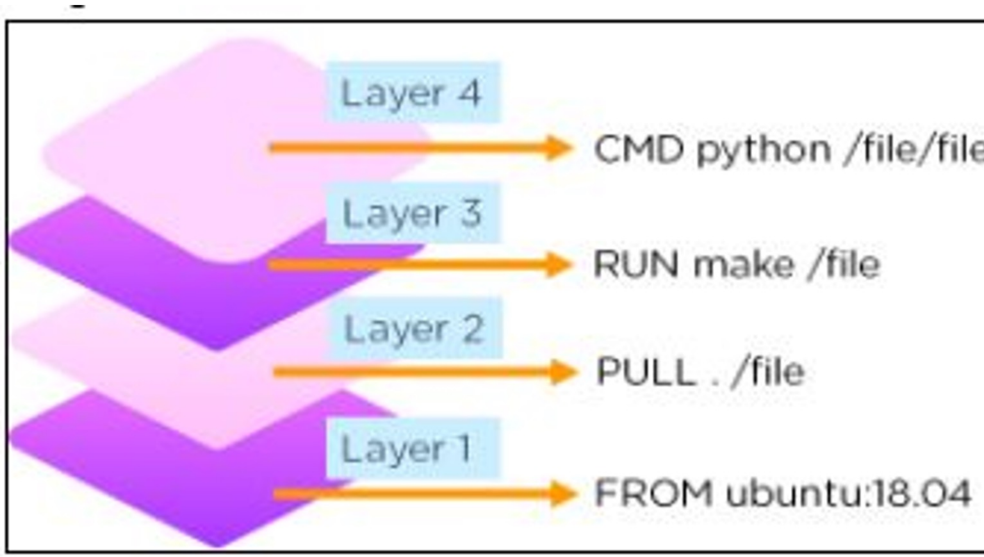
Create a text file with name **Dockerfile** or **Containerfile** with no extension

Write the commands that are required to build the image, follow the syntax



```
# Comment
INSTRUCTION arguments
```

- Each Dockerfile instruction runs in an independent container using an intermediate image built from every previous command. This means each instruction is independent from other instructions in the Dockerfile.



Podman build

Sample file

```
# This is a comment line ❶
FROM ubi7/ubi:7.7 ❷
LABEL description="This is a custom httpd container image" ❸
MAINTAINER John Doe <jdoe@xyz.com> ❹
RUN yum install -y httpd ❺
EXPOSE 80 ❻
ENV LogLevel "info" ❼
ADD http://someserver.com/filename.pdf /var/www/html ❽
COPY ./src/ /var/www/html/ ❾
USER apache ❿
ENTRYPOINT ["/usr/sbin/httpd"] ⓫
CMD ["-D", "FOREGROUND"] ⓬
```

Contents of a containerfile

- Lines that begin with a hash, or pound, sign (**#)** are comments.
- The **FROM** instruction declares that the new container image extends ubi7/ubi:7.7 container base image.
- **LABEL** is responsible for adding generic metadata to an image. A LABEL is a simple key-value pair.
- **MAINTAINER** indicates the Author field of the generated container image's metadata.
- **RUN** executes commands in a new layer on top of the current image. The shell that is used to execute commands is /bin/sh.
- **EXPOSE** indicates that the container listens on the specified network port at runtime.

Contents of a containerfile

- **ENV** is responsible for defining environment variables that are available in the container.
- **ADD** instruction copies files or folders from a **local or remote** source and adds them to the container's file system. If used to copy local files, those must be in the working directory.
- **COPY** copies files from the working directory and adds them to the container's file system. It is not possible to copy a remote file using its URL with this Containerfile instruction.
- **USER** specifies the username or the UID to use when running the container image for the RUN, CMD, and ENTRYPOINT instructions.
- **ENTRYPOINT** specifies the default command to execute when the image runs in a container. If omitted, the default ENTRYPOINT is /bin/sh -c.
- **CMD** provides the default arguments for the ENTRYPOINT instruction. If the default ENTRYPOINT applies (/bin/sh -c), then CMD forms an executable command and parameters that run at container start.

Guided Exercise: Creating a Basic Apache Container Image

- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch05s04

```
[student@workstation ~]$ lab dockerfile-create start
```

Lab: Creating Custom Container Images

- https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch05s05

```
[student@workstation ~]$ lab dockerfile-review start
```

Google Classroom

- https://classroom.google.com/c/NDkxODU1NjEwNDI2?cjc=hth6isq
- Invitation code: hth6isq