# Contents

# Exercise 1 : Creating a GitHub Account

Creating a GitHub Account You need a GitHub account to create one or more public Git repositories for the labs in this course. If you already have a GitHub account, you can skip these steps

**Important** If you already have a GitHub account, ensure that you only create public Git repositories for the labs in this course. The lab grading scripts and instructions require unauthenticated access to clone the repository. The repositories must be accessible without providing passwords, SSH keys, or GPG keys.

**Instructions**

To create a new GitHub account, perform the following steps:

1. Navigate to https://github.com using a web browser.

2. Enter the required details and then click Sign up for GitHub.

3. You will receive an email with instructions on how to activate your GitHub account. Verify your email address and then sign in to the GitHub website using the username and password you provided during account creation.

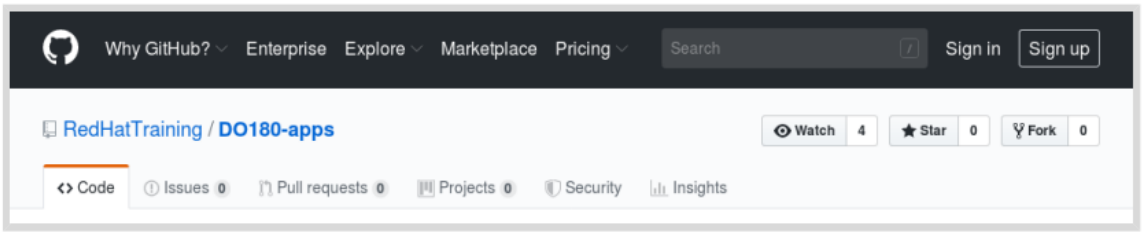
4. After you have logged in to GitHub, you can create new Git repositories by clicking New in the Repositories pane on the left of the GitHub home page.

# Exercise 2 : Fork sample applications into your personal GitHub account

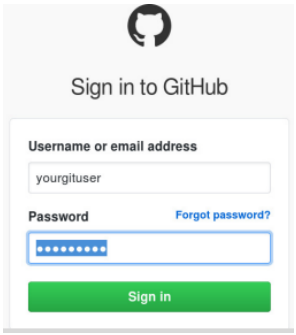
**Instructions**

1. Sign into your GitHub account

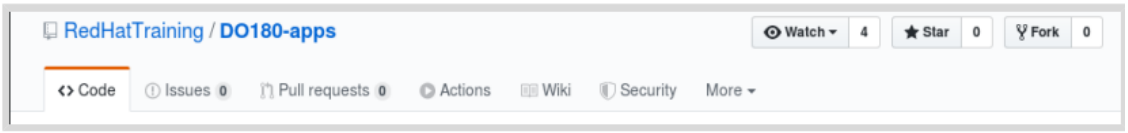
1.1. Open a web browser and navigate to https://github.com/RedHatTraining/DO180-apps. If you are not logged in to GitHub, click Sign in in the upper-right corner



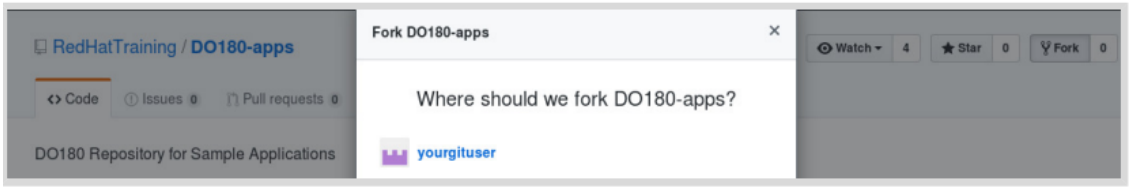

1.2. Log in to GitHub using your personal user name and password



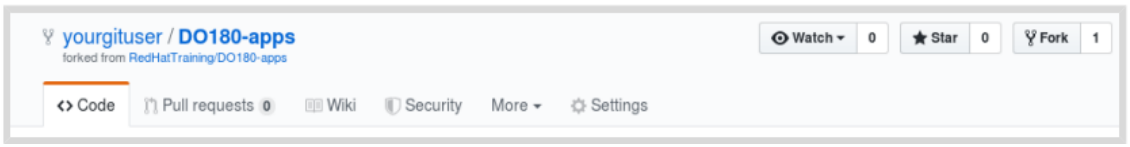

1.3. Navigate to the RedHatTraining/DO180-apps repository and click **Fork** in the top right corner.




1.4. In the Fork DO180-apps window, click yourgituser to select your personal GitHub project.




1.5. After a few minutes, the GitHub web interface displays your new repository yourgituser/DO180-apps.




2. Clone this course's sample applications from your personal GitHub account to your workstation machine. Perform the following steps:

2.1. Run the following command to clone this course's sample applications repository. Replace yourgituser with the name of your personal GitHub account.

```
[student@workstation ~]$ git clone https://github.com/<yourgituser>/DO180-apps
Cloning into 'DO180-apps'...
...output omitted...
```

2.2. Verify that /home/user/DO180-apps is a Git repository.

```
[student@workstation ~]$ cd DO180-apps
[student@workstation DO180-apps]$ git status
# On branch master
nothing to commit, working directory clean
```

2.3. Create a new branch to test your new personal access token.

```
[student@workstation DO180-apps]$ git checkout -b testbranch
Switched to a new branch `testbranch`
```

2.3 Configure Git

```
[student@workstation DO180-apps]$ git config --global user.email "<your@example.com>"
[student@workstation DO180-apps]$ git config --global user.name "<Your name>"
```

2.4. Make a change to the TEST file and then commit it to Git.

```
[student@workstation DO180-apps]$ echo "I love container" > TEST
[student@workstation DO180-apps]$ git add .
[student@workstation DO180-apps]$ git commit -am "My first commit"
...output omitted...
```

2.5. Push the changes to your recently created testing branch.

```
[student@workstation DO180-apps]$ git push --set-upstream origin testbranch
Username for `https://github.com`:  < Enter your GitHub username >
Password for `https://_yourgituser_@github.com`: < Enter your personal access token >
...output omitted...
```

2.6. Make other change to a text file, commit it and push it. You will notice you are no longer asked to put your user and password. This is because the git config command you ran in step 1.7.

```
[student@workstation DO180-apps]$ echo "Containers are the best" > TEST
[student@workstation DO180-apps]$ git add .
[student@workstation DO180-apps]$ git commit -am "My second commit"
[student@workstation DO180-apps]$ git push
...output omitted...
```

2.7. Verify that /home/user/DO180-apps contains this course's sample applications, and change back to the user's home folder.

```
[student@workstation DO180-apps]$ head README.md
# DO180-apps
...output omitted...
```

```
[student@workstation DO180-apps]$ cd ~
[student@workstation ~]$
```

3. Repeat all above steps for following repository

```
https://github.com/RedHatTraining/DO180
```

**Note** Now that you have a local clone of the DO180-apps repository on the workstation machine, you are ready to start this course's exercises. During this course, all exercises that build applications from source start from the master branch of the DO180-apps Git repository. Exercises that make changes to source code require you to create new branches to host your changes so that the master branch always contains a known good starting point. If for some reason you need to pause or restart an exercise and need to either save or discard changes you make into your Git branches,

**Finish**

This concludes the guided exercise.

# Exercise 3 : Creating a Red Hat Account

You need a Red Hat account to access the Red Hat Container Catalog images. If you already have a Red Hat account, you can skip this steps.

**Instructions**

To create a new Red Hat account, perform the following steps:

1. Navigate to https://redhat.com using a web browser.

2. Click Log in in the upper-right corner.

3. Click Register now on the right side pane.

4. Set Account type as Personal.

5. Fill in the remainder of the form with your personal information. In this form you also select the username and password for this account.

6. Click CREATE MY ACCOUNT.

**References**

Red Hat Customer Portal
https://access.redhat.com/

**Finish**

This concludes the exercise

# Exercise 4 : Creating a Quay Account

You need a Quay account to create one or more public container image repositories for the labs in this course. If you already have a Quay account, you can skip these steps to create a new account

**Important** If you already have a Quay account, ensure that you only create *public* container image repositories for the labs in this course.

**Instructions**

To create a new Quay account, perform the following steps:

1. Navigate to https://quay.io using a web browser.

2. Click Sign in in the upper-right corner (next to the search bar).

3. On the Sign in page, you can log in using your Red Hat ID, Google or GitHub credentials

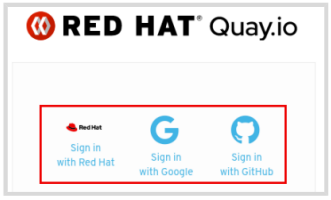



Figure C.1: Sign in using Google or GitHub credentials.

Alternatively, click Create Account to create a new account.

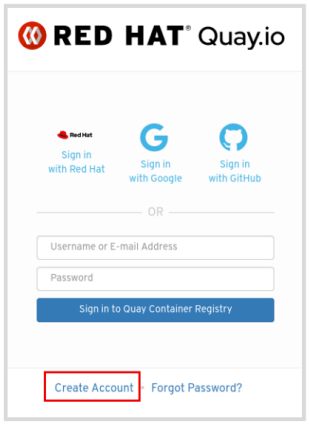



Figure C.2: Creating a new account

1. If you chose to skip the Red Hat, Google or GitHub login method and instead opted to create a new account, you will receive an email with instructions on how to activate your Quay account. Verify your email address and then sign in to the Quay website with the username and password you provided during account creation.

2. After you have logged in to Quay you can create new image repositories by clicking Create New Repository on the Repositories page.

**Working with CLI tools**

If you created your account with Red Hat, Google or Github, you need to set an account password to use CLI tools like Podman or Docker.

1. Click in the upper-right corner.

2. Click Account Settings.

3. Click the Change password link

**References**

Getting Started with Quay.io
https://docs.quay.io/solution/getting-started.html

**Repository Visibility**

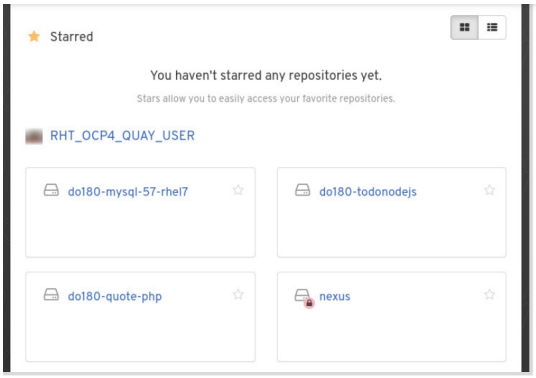
Quay.io Repository Visibility Quay.io offers the possibility of creating public and private repositories. Public repositories can be read by anyone without restrictions, despite write permissions must be explicitly granted. Private repositories have both read and write permissions restricted. Nevertheless, the number of private repositories in quay.io is limited, depending on the namespace plan.

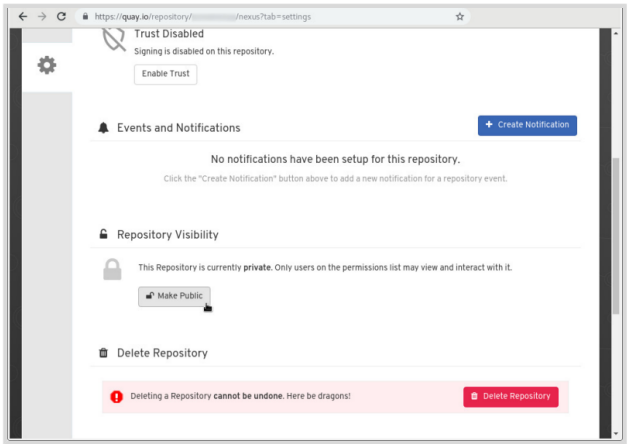
**Default Repository Visibility**

Repositories created by pushing images to quay.io are private by default. In order OpenShift (or any other tool) to fetch those images you can either configure a private key in both OpenShift and Quay, or make the repository public, so no authentication is required. Setting up private keys is out of scope of this document.

**Updating Repository Visibility**

In order to set repository visibility to public select the appropriate repository in https://quay.io/ repository/ (log in to your account if needed) and open the Settings page by clicking on the gear icon on the bottom left. Scroll down to the Repository Visibility section and click the *Make Public button*.



Get back the the list of repositories. The lock icon besides the repository name have disappeared, indicating the repository became public.

**Finish**

This concludes the exercise

# Exercise 5 : Generate GitHub Personal Access Token

Personal access tokens (PATs) are an alternative to using passwords for authentication to GitHub when using the GitHub API or the command line.

**Instructions**

Before starting any exercise, ensure you have:

1. Prepare your Github access token.

1.1. Navigate to https://github.com using a web browser and authenticate.

1.2. On the top of the page, click your profile icon, select the Settings menu, and then select Developer settings in the left pane of the page.

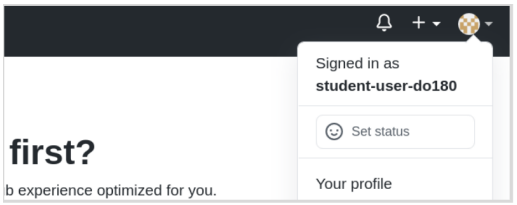



Figure 1.2: User menu

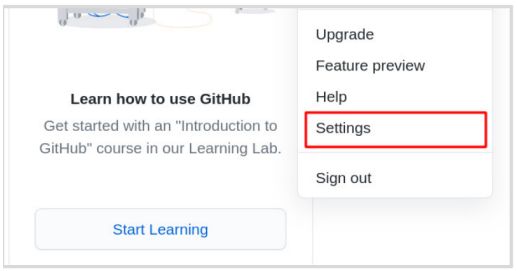



Figure 1.3: Settings menu

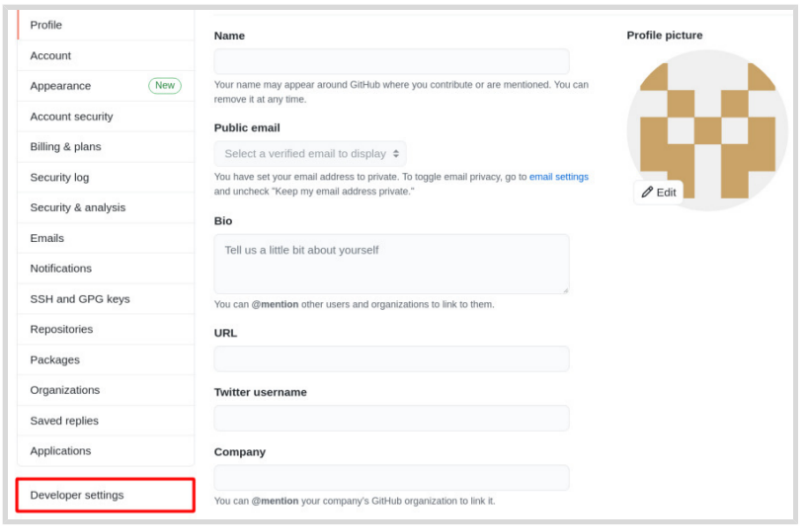



Figure 1.4: Developer settings

1.3. Select the Personal access token section on the left pane. On the next page, create your new token by clicking Generate new token, you are then prompted to enter your password.

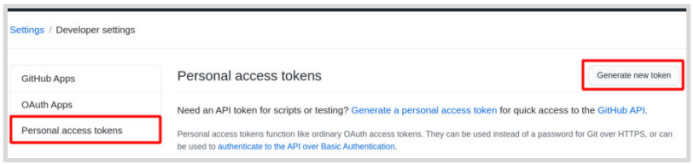



Figure 1.5: Personal access token pane

1.4. Write a short description about your new access token on the Note field.

1.5. Select the public_repo option and leave the other options unchecked. Create your new access token by clicking Generate token.

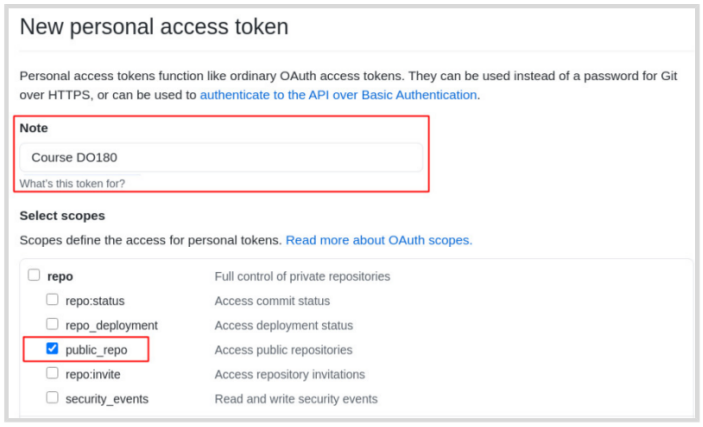



Figure 1.6: Personal access token configuration

1.6. Your new personal access token is displayed in the output. Using your preferred text editor, create a new file in student's home directory named token and ensure you paste in your generated personal access token. The personal access token can not be displayed again in GitHub.

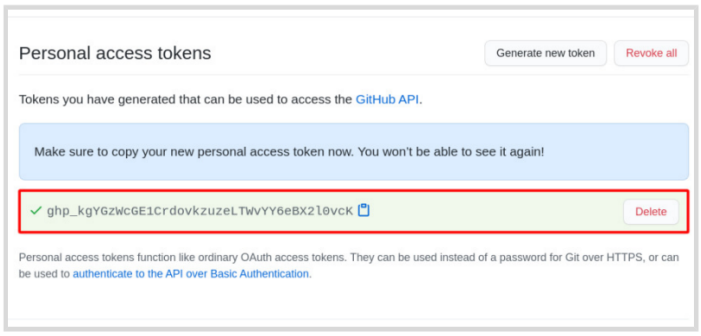
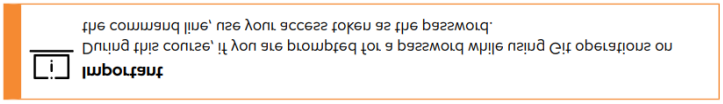



Figure 1.7: Generated access token




## Finish

This concludes the exercise.

# Exercise 6 : Creating a MySQL Database Instance

In this exercise, you will start a MySQL database inside a container and then create and populate a database.

**Instructions**

1. Create a MySQL container instance.

   1.1. Log in to the Red Hat Container Catalog with your Red Hat account. If you need to register with Red Hat, see the instructions above, Creating a Red Hat Account.

```
[student workstation ~]$ cd; podman login registry.redhat.io
Username: < your_username >
Password: < your_password  >
Login Succeeded!
```

   1.2. Start a container from the Red Hat Container Catalog MySQL image.

```
[student@workstation ~]$ podman run --name mysql-basic \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> -d registry.redhat.io/rhel8/mysql-80:1
Trying to pull ...output omitted...
Copying blob ...output omitted...
Writing manifest to image destination
Storing signatures
`2d37682eb33a`70330259d6798bdfdc37921367f56b9c2a97339d84faa3446a03
```

   This command downloads the MySQL 8.0 container image with the 1 tag, and then starts a container based on that image. It creates a database named items, owned by a user named user1 with mypa55 as the password. The database administrator password is set to r00tpa55 and the container runs in the background.

   1.3. Verify that the container started without errors.

```
[student@workstation ~]$ podman ps --format "{{.ID}} {{.Image}} {{.Names}}"
2d37682eb33a registry.redhat.io/rhel8/mysql-80:1 mysql-basic
```

2. Access the container sandbox by running the following command:

```
[student@workstation ~]$ podman exec -it mysql-basic /bin/bash
bash-4.2$
```

   This command starts a Bash shell, running as the mysql user inside the MySQL container.

3. Add data to the database.

   3.1. Connect to MySQL as the database administrator user (root). Run the following command from the container terminal to connect to the database:

```
bash-4.2$ mysql -uroot
Welcome to the MySQL monitor. Commands end with ; or \g.
...output omitted...
mysql>
```

The mysql command opens the MySQL database interactive prompt. Run the following command to determine the database availability:

```
mysql> show databases;

 ------------------------------------
| Database                           |
 ---------------------- ----------------
| information_schema          |
| items                              |
| mysql                              |
| performance_schema         |
| sys                                |
 ------------------------------------
 5 rows in set (0.01 sec)
```

3.2. Create a new table in the items database. Run the following command to access the database.

```
mysql> use items;
Database changed
```

3.3. Create a table called Projects in the items database.

```
mysql> CREATE TABLE Projects (id int NOT NULL,
 -> name varchar(255) DEFAULT NULL,
 -> code varchar(255) DEFAULT NULL,
 -> PRIMARY KEY (id));
 Query OK, 0 rows affected (0.01 sec)
```

3.4. Use the show tables command to verify that the table was created.

```
mysql> show tables;
 ----------------------------
| Tables_in_items     |
----------------------------
| Projects                 |
----------------------------
1 row in set (0.00 sec)
```

3.5. Use the insert command to insert a row into the table.

```
mysql> insert into Projects (id, name, code) values (1,'DevOps','ABC123');
Query OK, 1 row affected (0.02 sec)
```

3.6. Use the select command to verify that the project information was added to the table.

```
mysql> select * from Projects;

--------------------------------
| id | name   | code       |
 -------------------------------+
| 1 | DevOps | ABC123  |
--------------------------------+
1 row in set (0.00 sec)
```

3.7. Exit from the MySQL prompt and the MySQL container.

```
mysql> exit
Bye
bash-4.2$ exit
exit
```

**Finish**

This concludes the exercise.

# Exercise 7 : Exploring Root And Rootless Containers

Instructions

1. Run a container as the root user and check the UID of a running process inside it.

   1.1. Start a container with sudo from the Red Hat UBI 8 image.

```
[student@workstation ~]$ cd; sudo podman run --rm --name asroot -ti \
> registry.access.redhat.com/ubi8:latest /bin/bash
Trying to pull registry.access.redhat.com/ubi8:latest...
Getting image source signatures
...output omitted...
[root@f95d16108991 /]# whoami
root [root@f95d16108991 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

   1.2. Start a sleep process inside the container.

```
[root@f95d16108991 /]# sleep 1000
```

   1.3. On a new terminal, run ps to search for the process.

```
[student@workstation ~]$ sudo ps -ef | grep "sleep 1000"
root 3137 3117 0 10:18 pts/0 00:00:00 /usr/bin/coreutils -- coreutils-prog-shebang=sleep
/usr/bin/sleep 1000
```

   **Note** The process is owned by root account

   1.4. Exit the container

```
[root@f95d16108991 /]# sleep 1000
^C [root@f95d16108991 /]# exit
[student@workstation ~]$
```

2. Run a container as a regular user and check the UID of a running process inside it.

   2.1. Start another container from the Red Hat UBI 8 as a regular user.

```
[student@workstation ~]$ podman run --rm --name asuser -ti \ >
registry.access.redhat.com/ubi8:latest /bin/bash
Trying to pull registry.access.redhat.com/ubi8:latest...
Getting image source signatures
...output omitted...
[root@d289dccd5285 /]# whoami
root
[root@d289dccd5285 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

   2.2. Start a sleep process inside the container.

```
[root@d289dccd5285 /]# sleep 2000
```

2.3. On a new terminal, run ps to search for the process.

```
[student@workstation ~]$ sudo ps -ef | grep "sleep 2000" | grep -v grep
student 3345 3325 0 10:24 pts/0 00:00:00 /usr/bin/coreutils -- coreutils-prog-shebang=sleep
/usr/bin/sleep 2000
```

**Note** The process is owned by student account

2.4. Exit the container.

```
[root@d289dccd5285 /]# sleep 2000
^C
[root@d289dccd5285 /]# exit
[student@workstation ~]$
```

3. Remove all container processes created in this lab

3.1. List all running container processes

```
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                              COMMAND       CREATED        STATUS …
a97b664acfe1  registry.redhat.io/rhel8/mysql..  “container-entryp”…
```

3.2. Removes all running container processes

```
[student@workstation ~]$ podman stop  -a
[student@workstation ~]$ podman rm  -af
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                              COMMAND       CREATED        STATUS …
```

**Note U**ses of **-a** will perform the task on all running container processes. Use **-f** will force it.

**Finish**

This concludes the section.

# Exercise 8 : Managing a MySQL Container

In this exercise, you will create and manage a MySQL® database container.

**Instructions**

1. Download the MySQL database container image and attempt to start it. The container does not start because several environment variables must be provided to the image.

1.1. Log in to the Red Hat Container Catalog with your Red Hat account. If you need to register with Red Hat, see the instructions in Creating a Red Hat Account.

```
[student@workstation ~]$ cd; podman login registry.redhat.io
Username: < your_username >
Password: < your_password  >
Login Succeeded!
```

1.2. Download the MySQL database container image and attempt to start it.

```
[student@workstation ~]$ podman run --name mysql-db \
> registry.redhat.io/rhel8/mysql-80:1
Trying to pull
...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
You must either specify the following environment variables:
MYSQL_USER (regex: '^$')
MYSQL_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]$')
MYSQL_DATABASE (regex: '^$')
`Or the following environment variable:`
MYSQL_ROOT_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]$')
Or both.
Optional Settings:
...output omitted...
```

For more information, see https://github.com/sclorg/mysql-container

**Note** If you try to run the container as a daemon (-d), then the error message about the required variables is not displayed. However, this message is included as part of the container logs, which can be viewed using the following command:

```
[student@workstation ~]$ podman logs mysql-db
```

2. Create a new container named mysql, and then specify each required variable using the -e parameter.

**Note** Make sure you start the new container with the correct name.

```
[student@workstation ~]$ podman run --name mysql \
> -d -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> registry.redhat.io/rhel8/mysql-80:1
```

The command displays the container ID for the mysql container. Below is an example of the output.

```
a49dba9ff17f2b5876001725b581fdd331c9ab8b9eda21cc2a2899c23f078509
```

3. Verify that the mysql container started correctly. Run the following command:

```
[student@workstation ~]$ podman ps --format="{{.ID}} {{.Names}} {{.Status}}"
a49dba9ff17f mysql Up About a minute ago
```

The command only shows the first 12 characters of the container ID displayed in the previous command.

4. Populate the items database with the Projects table:

4.1. Run podman cp command to copy database file to mysql container.

```
[student@workstation ~]$ podman cp \
> /home/student/DO180/labs/manage-lifecycle/db.sql mysql:/
```

4.2. View the db.sql content

```
[student@workstation ~]$ podman exec mysql cat /db.sql
CREATE TABLE `Project` (id int(11) NOT NULL, `name` varchar(255) DEFAULT NULL, `code`
varchar(255) DEFAULT NULL, PRIMARY KEY (id));
insert into `Projects` (`id`, `name`, `code`) values (1,'DevOps','DO180');
```

4.3. Populate the items database with the Projects table.

```
[student@workstation ~]$ podman exec mysql /bin/bash \
> -c 'mysql -uuser1 -pmypa55 items < /db.sql'
mysql: [Warning] Using a password on the command line interface can be insecure.
```

5. Create another container using the same container image as the previous container. Interactively enter the /bin/bash shell instead of using the default command for the container image.

```
[student@workstation ~]$ podman run --name mysql-2 \
> -it registry.redhat.io/rhel8/mysql-80:1 /bin/bash
bash-4.4$
```

6. Try to connect to the MySQL database in the new container:

```
bash-4.4$ mysql -uroot
```

The following error displays:

```
ERROR 2002 (HY000): Can't connect to local MySQL ...output omitted...
```

**Note** The MySQL database server is not running because the container executed the /bin/ bash command rather than starting the MySQL server.

7. Exit the container.

```
bash-4.4$ exit
```

8. Verify that the mysql-2 container is not running.

```
[student@workstation ~]$ podman ps -a --format="{{.ID}} {{.Names}} {{.Status}}"
2871e392af02 mysql-2 Exited (1) 19 seconds ago
```

```
a49dba9ff17f mysql Up 10 minutes ago
c053c7e09c21 mysql-db Exited (1) 44 minutes ago
```

**Notice** mysql-2 container exited right after it starts

9. Query the mysql container to list all rows in the Projects table. The command instructs the bash shell to to query the items database using a mysql command.

```
[student@workstation ~]$ podman exec mysql /bin/bash \
> -c 'mysql -uuser1 -pmypa55 -e "select * from items.Projects;"'
mysql: [Warning] Using a password on the command-line interface can be insecure.
id name      code
1  DevOps   DO180
```

**Finish**

This concludes the exercise.

# Exercise 9 : Create MySQL Container with Persistent Database

In this exercise, you will create a container that stores the MySQL database data into a host directory.

**Instructions**

1. Create the /home/student/local/mysql directory with the correct SELinux context and permissions.

   1.1. Create the /home/student/local/mysql directory.

```
[student@workstation ~]$ cd; mkdir -pv /home/student/local/mysql
mkdir: created directory /home/student/local
mkdir: created directory /home/student/local/mysql
```

   1.2. Add the appropriate SELinux context for the /home/student/local/mysql directory and its contents.

```
[student@workstation ~]$ sudo semanage fcontext -a \
>   -t container_file_t '/home/student/local/mysql(/.*)?'
```

   1.3. Apply the SELinux policy to the newly created directory.

```
[student@workstation ~]$ sudo restorecon -R /home/student/local/mysql
```

   1.4. Verify that the SELinux context type for the /home/student/local/mysql directory is container_file_t.

```
[student@workstation ~]$ ls -ldZ /home/student/local/mysql
drwxrwxr-x. 2 student student unconfined_u:object_r:container_file_t:s0 6 May 26 14:33
/home/student/local/mysql
```

   1.5. Change the owner of the /home/student/local/mysql directory to the mysql user and mysql group:

```
[student@workstation ~]$ podman unshare chown 27:27 /home/student/local/mysql
```

   **Note** The user running processes in the container must be capable of writing files to the directory.

   The permission should be defined with the numeric user ID (UID) from the container. For the MySQL service provided by Red Hat, the UID is 27. The podman unshare command provides a session to execute commands within the same user namespace as the process running inside the container.

2. Create a MySQL container instance with persistent storage.

   2.1. Log in to the Red  Hat Container Catalog with your Red Hat account. If you need to register with Red Hat, see the instructions in Creating a Red Hat Account.

```
[student@workstation ~]$ podman login registry.redhat.io
Username: < your_username >
Password: < your_password  >
Login Succeeded!
```

2.2. Pull the MySQL container image.

```
[student@workstation ~]$ podman pull registry.redhat.io/rhel8/mysql-80:1
Trying to pull
...output omitted...
registry.redhat.io/rhel8/mysql-80:1
...output omitted...
...output omitted...
Writing manifest to image destination Storing signatures
4ae3a3f4f409a8912cab9fbf71d3564d011ed2e68f926d50f88f2a3a72c809c5
or if it is already been downloaded in previous exercises, it shows this instead
1: Pulling from rhel8/mysql-80
Digest: sha256:500f6 …output omitted…
Status: Image is up to date for …
registry.redhat.io/rhel8/mysql-80:1
```

2.3. Create a new container specifying the mount point to store the MySQL database data:

```
[student@workstation ~]$ podman run --name persist-db \
> -d -v /home/student/local/mysql:/var/lib/mysql/data \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> registry.redhat.io/rhel8/mysql-80:1
6e0ef134315b510042ca757faf869f2ba19df27790c601f95ec2fd9d3c44b95d
```

This command mounts the /home/student/local/mysql directory from the host to the /var/lib/mysql/data directory in the container. By default, the MySQL database stores data in the /var/lib/mysql/data directory.

2.4. Verify that the container started correctly.

```
[student@workstation ~]$ podman ps --format="{{.ID}} {{.Names}} {{.Status}}"
6e0ef134315b persist-db Up 3 minutes ago
```

2.5. Verify that the /var/lib/mysql is mounted in the container

```
[student@workstation ~]$ podman exec persist-db df -k | grep  mysql
/dev/mapper/rhel-home 133067284 1814028 131253256   2% /var/lib/mysql/data
```

3. Verify that the /home/student/local/mysql directory contains the items directory.

```
[student@workstation ~]$ ls -ld /home/student/local/mysql/items
drwxr-x---. 2 100026 100026 6 Apr 8 07:31 /home/student/local/mysql/items
```

**Note** The items directory stores data related to the items database that was created by this container. If the items directory does not exist, then the mount point was not defined correctly during container creation.

**Note** Alternatively you can run the same command with podman unshare to check numeric user ID (UID) from the container.

```
[student@workstation ~]$ podman unshare ls -ld /home/student/local/mysql/items
drwxr-x---. 2 27 27 6 Apr 8 07:31 /home/student/local/mysql/items
```

4. Remove all container processes created in this lab

4.1. List all running container processes

```
[student@workstation ~]$ podman ps
```

4.2. Removes all running container processes

```
[student@workstation ~]$ podman stop  -a
[student@workstation ~]$ podman rm  -af
[student@workstation ~]$ podman ps
CONTAINER ID        IMAGE                           COMMAND       CREATED         STATUS …
```

**Note U**ses of **-a** will perform the task on all running container processes. Use **-f** will force it.

4.3. Removes all files in /home/student/local/mysql for next exercise

```
[student@workstation ~]$ sudo \rm -rf /home/student/local/mysql/*
```

**Note** *rm* is to ensure we not using aliases or functions that the root account may have created. Use of *-rf* flag is to forcibly delete all files including subdirectories

**Finish**

This concludes the exercise

# Exercise 10 : Loading the Database

In this exercise, you will create a MySQL database container with port forwarding enabled. After populating a database with a SQL script, you verify the database content using three different methods.

**Instructions**

1. Create a MySQL container instance with persistent storage and port forwarding.

   1.1. Log in to the Red Hat Container Catalog with your Red Hat account. If you need to register with Red Hat, see the instructions in Creating a Red Hat Account.

```
[student@workstation ~]$ cd; podman login registry.redhat.io
Username: < your_username >
Password: < your_password  >
Login Succeeded!
```

   1.2. Download the MySQL database container image and then create a MySQL container instance with persistent storage and port forwarding.

```
[student@workstation ~]$ podman run --name mysqldb-port \
> -d -v /home/student/local/mysql:/var/lib/mysql/data -p 13306:3306 \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> registry.redhat.io/rhel8/mysql-80:1
Trying to pull
...output omitted...
Copying blob sha256:1c9f515...output omitted...
 72.70 MB / ? [=-----------------------------------------------------------] 5s
Copying blob sha256:1d2c4ce...output omitted...
 1.54 KB / ? [-------------------------------------=---------------------] 0s
Copying blob sha256:f1e961f
...output omitted...
 6.85 MB / ? [-----------------------=-----------------------------------] 0s
Copying blob sha256:9f1840c
...output omitted...
 62.31 MB / ? [-------------------=---------------------------------------] 7s
Copying config sha256:60726
...output omitted...
 6.85 KB / 6.85 KB [=======================================================] 0s
Writing manifest to image destination
Storing signatures
066630d45cb902ab533d503c83b834aa6a9f9cf88755cb68eedb8a3e8edbc5aa
```

   The last line of your output and the time needed to download each image layer will differ. The -p option configures port forwarding so that port 13306 on the local host forwards to container port 3306.

2. Verify that the mysqldb-port container started successfully and enables port forwarding.

```
[student@workstation ~]$ podman ps --format="{{.ID}} {{.Names}} {{.Ports}}"
9941da2936a5        mysqldb-port    0.0.0.0:13306->3306/tcp
```

3. Populate the database using the provided file. If there are no errors, then the command does not return any output.

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
> -P 13306 items < /home/student/DO180/labs/manage-networking/db.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
```

There are multiple ways to verify that the database loaded successfully. The next steps show three different methods. You only need to use one of the methods.

4. Verify that the database loaded successfully by executing a non-interactive command inside the container.

```
[student@workstation ~]$ podman exec -it mysqldb-port \
> mysql -uroot items -e "SELECT * FROM Item"
--------------------------------------------------
| id | description               | done  |
--------------------------------------------------
| 1 | Pick up newspaper          |      0  |
| 2 | Buy groceries              |      1  |
--------------------------------------------------
```

5. Verify that the database loaded successfully by using port forwarding from the local host. This alternate method is optional.

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
> -P13306 items -e "SELECT * FROM Item"
--------------------------------------------------
| id | description               | done  |
--------------------------------------------------
| 1 | Pick up newspaper          |      0  |
| 2 | Buy groceries              |      1  |
--------------------------------------------------
```

6. Verify that the database loaded successfully by opening an interactive terminal session inside the container. This alternate method is optional.

6.1. Open a Bash shell inside the container.

```
[student@workstation ~]$ podman exec -it mysqldb-port /bin/bash
bash-4.2$
```

6.2. Verify that the database contains data:

```
bash-4.4$ mysql -uroot items -e "SELECT * FROM Item"

--------------------------------------------------
| id | description               | done  |
--------------------------------------------------
| 1 | Pick up newspaper          |      0  |
```

```
| 2 | Buy groceries                |       1  |
-------------------------------------------------
```

6.3. Exit from the container:

```
bash-4.4$ exit
```

**Finish**

This concludes the exercise.

# Exercise 11 : Creating a Custom Apache Container Image

In this guided exercise, you will create a custom Apache container image using the podman commit command.

**Instructions**

1. Log in to your Quay.io account and start a container by using the image available at quay.io/redhattraining/httpd-parent. The -p option allows you to specify a redirect port. In this case, Podman forwards incoming requests on TCP port 8180 of the host to TCP port 80 of the container.

```
[student@workstation ~]$ cd; podman login quay.io
Username: < your_quay_username >
Password: < your_quay_password >
Login Succeeded!

[student@workstation ~]$ podman run -d --name official-httpd \
> -p 8180:80 quay.io/redhattraining/httpd-parent
...output omitted...
Writing manifest to image destination
Storing signatures
`3a6baecaff2b`4e8c53b026e04847dda5976b773ade1a3a712b1431d60ac5915d
```

**Note** Your last line of output is different from the last line shown above. Note the first twelve characters.

2. Create an HTML page on the official-httpd container.

2.1. Access the shell of the container by using the podman exec command and create an HTML page.

```
[student@workstation ~]$ podman exec -it official-httpd /bin/bash
bash-4.4# echo "Hello World" > /var/www/html/page.html
```

2.2. Exit from the container.

```
bash-4.4# exit
```

2.3. Ensure that the HTML file is reachable from the workstation machine by using the curl command.

```
[student@workstation ~]$ curl 127.0.0.1:8180/page.html
```

**Note** You should see the following output:

Hello World

**Note**: Continue next page

3. Use the podman diff command to examine the differences in the container between the image and the new layer created by the container.

```
[student@workstation ~]$ podman diff official-httpd
C /root
 ...output omitted...
C /var
C /var/log
C /var/log/httpd
A /var/log/httpd/access_log
A /var/log/httpd/error_log
C /var/www
C /var/www/html
A /var/www/html/page.html
```

**Note** Often, web server container images label the /var/www/html directory as a volume. In these cases, any files added to this directory are not considered part of the container file system, and would not show in the output of the podman diff command. The quay.io/redhattraining/httpd-parent container image does not label the /var/www/html directory as a volume. As a result, the change to the /var/www/html/do180.html file is considered a change to the underlying container file system.

4. Create a new image with the changes created by the running container.

4.1. Stop the official-httpd container.

```
[student@workstation ~]$ podman stop official-httpd
`3a6baecaff2b`4e8c53b026e04847dda5976b773ade1a3a712b1431d60ac5915d
```

4.2. Commit the changes to a new container image with a new name. Use your name as the author of the changes.

```
[student@workstation ~]$ podman commit \
> -a 'Your Name' official-httpd custom-httpd
Getting image source signatures
Skipping fetch of repeat blob sha256:071d8bd765171080d01682844524be57ac9883e...
...output omitted...
Copying blob sha256:1e19be875ce6f5b9dece378755eb9df96ee205abfb4f165c797f59a9...
 15.00 KB / 15.00 KB [=======================================================] 0s
Copying config sha256:8049dc2e7d0a0b1a70fc0268ad236399d9f5fb686ad4e31c7482cc...
 2.99 KB / 2.99 KB [=========================================================] 0s
Writing manifest to image destination Storing signatures
`31c3ac78e9d4`137c928da23762e7d32b00c428eb1036cab1caeeb399befe2a23
```

4.3. List the available container images.

```
[student@workstation ~]$ podman images
```

**Note** The expected output is similar to the following:

```
REPOSITORY                TAG     IMAGE ID      CREATED SIZE
localhost/custom-httpd    latest  31c3ac78e9d4  ...     ...
…Output omitted…
```

**Note** The image ID matches the first 12 characters of the hash. The most recent images are listed at the top.

5. Publish the saved container image to the container registry.

5.1. To tag the image with the registry host name and tag, run the following command.

```
[student@workstation ~]$ podman tag custom-httpd \
> quay.io/< YOUR_QUAY_USERNAME >/custom-httpd:v1.0
```

5.2. Run the podman images command to ensure that the new name has been added to the cache.

```
[student@workstation ~]$ podman images
REPOSITORY                                TAG     IMAGE ID      ...
localhost/custom-httpd                    latest  31c3ac78e9d4  ...
quay.io/YOUR_QUAY_USERNAME/custom-httpd   v1.0    31c3ac78e9d4  ...
…Output omitted…
```

5.3. Publish the image to your Quay.io registry.

```
[student@workstation ~]$ podman push \
> quay.io/< YOUR_QUAY_USERNAME >/custom-httpd:v1.0
Getting image source signatures
Copying blob sha256:071d8bd765171080d01682844524be57ac9883e53079b6ac66707e19...
  200.44 MB / 200.44 MB [==============================================] 1m38s
...output omitted...
Copying config sha256:31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb3...
  2.99 KB / 2.99 KB [==========================================================] 0s
Writing manifest to image destination
Copying config sha256:31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb3...
  0 B / 2.99 KB [----------------------------------------------------------------------------------------------] 0s
Writing manifest to image destination
Storing signatures
```

**Note** Pushing the custom-httpd image creates a private repository in your Quay.io account. Currently, private repositories are disallowed by Quay.io free plans. You can either create the public repository prior to pushing the image, or change the repository to public afterwards.

5.4. Verify that the image is available from Quay.io. The podman search command requires the image to be indexed by Quay.io. That may take some hours to occur, so use the podman pull command to fetch the image. This proves that the image is available.

```
[student@workstation ~]$ podman pull \
> quay.io/< YOUR_QUAY_USERNAME >/custom-httpd:v1.0
31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb3
```

6. Create a container from the newly published image. Use the podman run command to start a new container. Use your_quay_username/custom-httpd:v1.0 as the base image.

```
[student@workstation ~]$ podman run -d --name test-httpd -p 8280:80 \
>  < YOUR_QUAY_USERNAME >/custom-httpd:v1.0
c0f04e906bb12bd0e514cbd0e581d2746e04e44a468dfbc85bc29ffcc5acd16c
```

7. Use the curl command to access the HTML page. Make sure you use port 8280. This should display the HTML page created in the previous step.

```
[student@workstation ~]$ curl http://localhost:8280/page.html
 Hello World
```

**Finish**

This concludes the guided exercise.

# Exercise 12 : Build an Container Image with Containerfile

In this exercise, you will create a basic Apache container image.

**Instructions**

1. Create the Apache Containerfile.

1.1. Open a terminal on workstation. Using your preferred editor, create a new Containerfile.

```
[student@workstation ~]$ cd; mkdir -p ~/labs; vim ~/labs/Containerfile
```

1.2. Use UBI 8.3 as a base image by adding the following FROM instruction at the top of the new Containerfile:

```
FROM ubi8/ubi:8.3
```

1.3. Beneath the FROM instruction, include the MAINTAINER instruction to set the Author field in the new image. Replace the values to include your name and email address.

```
MAINTAINER Your Name
```

1.4. Below the MAINTAINER instruction, add the following LABEL instruction to add description metadata to the new image:

```
LABEL description="A custom Apache container based on UBI 8"
```

1.5. Add a RUN instruction with a yum install command to install Apache on the new container.

```
RUN yum install -y httpd && \
yum clean all
```

1.6. Add a RUN instruction to replace contents of the default HTTPD home page.

```
RUN echo "Hello from Containerfile" > /var/www/html/index.html
```

1.7. Use the EXPOSE instruction beneath the RUN instruction to document the port that the container listens to at runtime. In this instance, set the port to 80, because it is the default for an Apache server.

```
EXPOSE 80
```

Note The EXPOSE instruction does not actually make the specified port available to the host; rather, the instruction serves as metadata about the ports on which the container is listening.

1.8. At the end of the file, use the following CMD instruction to set httpd as the default entry point:

```
CMD ["httpd", "-D", "FOREGROUND"]
```

1.9. Verify that your Containerfile matches the following before saving and proceeding with the next steps:

```
FROM ubi8/ubi:8.3

MAINTAINER Your Name

LABEL description="A custom Apache container based on UBI 8"

RUN yum install -y httpd && \
      yum clean all

RUN echo "Hello from Containerfile" > /var/www/html/index.html

EXPOSE 80

CMD ["httpd", "-D", "FOREGROUND"]
```

2. Build and verify the Apache container image.

2.1. Use the following commands to create a basic Apache container image using the newly created Containerfile:

```
[student@workstation ~]$ cd ~/labs/
[student@workstation dockerfile]$ podman build --layers=false -t myimage/apache .

STEP 1: FROM ubi8/ubi:8.3
Getting image source signatures
Checking if image destination supports signatures
Copying blob 55eda7743468 done
Copying blob 4b21dcdd136d [>----------------------------------------------]  71.46 MB / 71.46 MB
...output omitted...
Storing signatures
STEP 2: MAINTAINER Your Name
STEP 3: LABEL description="A custom Apache container based on UBI 8"
STEP 4: RUN yum install -y httpd && yum clean all
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
...output omitted...
STEP 5: RUN echo "Hello from Containerfile" > /var/www/html/index.html
STEP 6: EXPOSE 80
STEP 7: CMD ["httpd", "-D", "FOREGROUND"]
STEP 8: COMMIT
...output omitted...
localhost/myimage/apache:latest
Getting image source signatures
...output omitted...
Storing signatures
b49375fa8ee1e549dc1b72742532f01c13e0ad5b4a82bb088e5befbe59377bcf
```

**Note** The container image listed in the FROM instruction is only downloaded if it is not already present in local storage.

**Note** Podman creates many anonymous intermediate images during the build process. They are not be listed unless -a is used. Use the --layers=false option of the build subcommand to instruct Podman to delete intermediate images.

2.2. After the build process has finished, run podman images to see the new image in the image repository.

```
[student@workstation dockerfile-create]$ $ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED         SIZE
localhost/myimage/apache            latest  8ebfe343e08c  15 seconds ago 234 MB
registry.access.redhat.com/ubi8/ubi 8.3     4199acc83c6a  6 weeks ago    213 MB
...output omitted...
```

3. Run the Apache container.

3.1. Use the following command to run a container using the Apache image:

```
[student@workstation dockerfile]$ podman run --name lab-apache \
> -d -p 10080:80 myimage/apache
fa1d1c450e8892ae085dd8bbf763edac92c41e6ffaa7ad6ec6388466809bb391
```

3.2. Run the podman ps command to see the running container.

```
[student@workstation dockerfile-create]$ podman ps
CONTAINER ID  IMAGE                           COMMAND               ...output omitted...
fa1d1c450e88  localhost/myimage/apache:latest httpd -D FOREGROU     ...output omitted...
```

3.3. Use the curl command to verify that the server is running.

```
[student@workstation dockerfile]$ curl 127.0.0.1:10080
Hello from Containerfile
```

**Finish**

This concludes the guided exercise.