Lab- containers in RHEL8

**Exercise1 : RUNNING CONTAINERS WITH PODMAN**

1. Use sudo yum module install container-tools to install the container software.
2. Type podman ps -a to get an overview of currently existing containers. Observe the Status field, where you can see whether the container currently is active.
3. Type podman run -d nginx. This command starts an nginx container in detached mode.
4. Type podman ps and observe the output. In the Container ID field, you’ll see the unique ID that has been generated. Also observe the Name field, where you’ll see a name that has automatically been generated.
5. Type podman run -it busybox. This command runs the busybox cloud image, a minimized Linux distribution that is often used as the foundation for building custom containers.
6. Because the busybox container image was configured to run a shell as the default command, you’ll get access to the shell that it is running. Type ps aux to see the processes running in this container namespace.
7. Type exit to close the busybox shell.
8. Type podman ps. You won’t see the busybox container anymore because in the previous step you exited it.
9. Type podman run -it busybox once more, and when you have access to its interactive shell, type Ctrl-p, Ctrl-q to detach.
10. Use podman ps. You’ll notice the busybox container is still running. Look at the Name column to find the name for the container that was automatically generated.
11. Type podman stop <name>, where <name> should be replaced with the name you found in the preceding step.

**Exercice 2 : MANAGING CONTAINER IMAGES**

1. Type podman info to check which registries are currently used.
2. Use podman login registry.redhat.io and enter your Red Hat account credentials to ensure full access to the Red Hat registries.
3. Use podman search registry.redhat.io/ubi8 to search only in registry.redhat.io for all the UBI images.
4. Use skopeo inspect docker://registry.redhat.io/ubi8 to show information about the container image. Do you see which command is started by default?
5. Now use podman pull registry.redhat.io/ubi8 to pull the image.
6. Type podman images to verify the image is now locally available.
7. Type podman inspect registry.redhat.io/ubi8 and look for the command that is started by default by this image.

**Exercice 3 : RUNNING COMMANDS IN A CONTAINER**

1. Use podman run -d --rm --name=web2 nginx.
2. Type podman ps to verify that the web2 container is available.
3. Use podman exec -it web2 /bin/bash to open a Bash shell in the container.
4. Within the container shell, type ps aux. You will see that there is no ps command in the nginx container; the reason is that many containers come without even fundamental standard tools.
5. Type ls /proc, and notice a few directories have a numeric name. These are the PID directories, and if you don’t have access to the ps command, this is how you can find process information.
6. Each /proc/<PID> directory has a file with the name cdmline. Type cat /proc/1/cmdline to find that the nginx process has been started as PID 1 within the container.
7. Type exit to close the Bash shell you just opened on the container.
8. Type podman ps to confirm that the web2 container is still running. It should be running because the exit command you used in the preceding step only exited the Bash command, not the primary command running inside the container.
9. On the container host, type uname -r to confirm the current kernel version. The el8 part in the name of the kernel indicates this is an Enterprise Linux kernel, which you’ll see only on RHEL or CentOS 8.
10. Type podman run -it ubuntu. This will run the latest Ubuntu image from the Docker registry and give access to a shell. Because the image has the shell set as the entrypoint command (the default command it should start), you don’t need to specify the name of the shell as well.
11. Type cat /etc/os-release to confirm this really is an Ubuntu container.
12. Type uname -r to see the Enterprise Linux kernel that you saw previously in step 6. The reason is that containers really are all running on the same kernel, no matter which Linux distribution container you’re running on top.
13. Type exit to close the Interactive TTY. What do you think? Does that command shut down the container?
14. Use podman ps to verify the Ubuntu container is no longer active. While using exit in step 13, you exited the entrypoint command running in the container, so there is now nothing else to be done anymore.

**Exercice 4 : MANAGING CONTAINER PORT MAPPINGS**

1. Type podman run --name nginxport -d -p 8080:80 nginx to run an nginx container and expose it on host port 8080.
2. Type podman ps to verify that the container has been started successfully with port forwarding enabled.
3. Use sudo firewall-cmd --add-port 8080/tcp –permanent

sudo firewall-cmd --reload to open this port in the firewall on the host operating system.

1. Type curl localhost:8080 to verify that you get access to the default nginx welcome page.

**Exercice 5 : manage containers environment variables**

1. Use podman run mariadb. It will fail (and you will see an error message on the STDOUT).
2. Use podman ps -a to see the automatically generated name for the failing mariadb container.
3. Use podman logs container\_name to see the Entrypoint application error log. Make sure to replace container\_name with the name you found in step 2.
4. Use podman inspect mariadb and look for a usage line. You won’t see any.
5. Use podman search registry.redhat.io/rhel8/mariadb to find the exact version number of the mariadb image in the RHEL registry.
6. Use podman run registry.redhat.io/rhel8/mariadb-nnn (make sure to replace nnn with the version number you found in step 5). It will also fail but will show much more usage details on the STDOUT. The reason is that the Red Hat mariadb image is not the same as the image that was fetched from the Docker registry in the first step of this procedure.
7. Use podman inspect registry.redhat.io/rhel8/mariadb-nnn and in the command output search for the usage line. It will tell you exactly how to run the mariadb image.
8. According to the instructions that you found here, type podman run -d -e MYSQL\_USER=bob -e MYSQL\_PASSWORD=password -e MYSQL\_DATABASE=mydb -e MYSQL\_ROOT\_PASSWORD=password -p 3306:3306 registry.redhat.io/rhel8/mariadb-103. (By the time you read this, the version number may be different, so make sure to check the version number of the image if you’re experiencing a failure in running this command.)
9. Use podman ps. You will see the mariadb container has now been started successfully.

**Exercice 5 : ATTACHING STORAGE TO CONTAINERS**

1. Use sudo mkdir /opt/dbfiles; sudo chmod o+w /opt/dbfiles to create a directory on the host operating system.
2. Use podman run -d --name mydbase -v /opt/dbfiles:/var/lib/mysql:Z -e MYSQL\_USER=user -e MYSQL\_PASSWORD=password -e MYSQL\_DATABASE=mydbase registry.redhat.io/rhel8/mariadb-103.
3. Type podman ps -a. You’ll see starting the container has failed.
4. Use podman logs mydbase to investigate why it has failed. Because the error was not related to the container application, the logs don’t show you anything.
5. Remove the failed container by using podman rm mydbase.
6. Type sudo chown $(id -un) /opt/dbfiles.
7. Run the command shown in step 2 again. It will now be successful.

**Exercice 6 :**

Create a container with the name, "logserver" from rhel8/rsyslog image from registry.lab.example.com registry.  
configure the container with systemd services as the wallah user using the service name, "container-logserver" so that it can be persistent across reboot.  
Configure your host journal to store all journal across reboot copy all journal form /var/log/journal and all subdirectories to /home/wallah/container-logserver. create and mount /home/wallah/container-logserver as a persistent storage to the container as /var/log/journal when container start  
Use "administrator" as the username and "admin123" as the credentials for the image registry. use wallah as wallah's password.

**Exercice 6 : FAIRE FONCTIONNER LES CONTENEURS COMME SERVICES SYSTEMD**

1. Use sudo useradd linda to create a user linda.
2. Use sudo passwd linda to set the password for user linda.
3. Type sudo loginctl enable-linger linda to enable the linger feature for user linda.
4. Log in as user linda (no su or sudo).
5. Type mkdir -p ~/.config/systemd/user; cd ~/.config/systemd/user to create and activate the directory where the systemd user files will be created.
6. Use podman run -d --name mynginx -p 8081:80 nginx to start an nginx pod.
7. Type podman ps to verify the nginx pod has been started.
8. Create the systemd user files using podman generate systemd --name mynginx --files

A systemd unit file with the name container-mynginx.service is created.

1. Use vim container-mynginx.service and change the WantedBy line such that it reads WantedBy=default.target.
2. Type systemctl --user daemon-reload to ensure that systemd picks up the changes.
3. Use systemctl --user enable container-mynginx.service to enable the systemd user service. (Do not try to start it because it has already been started!)
4. Type systemctl --user status container-mynginx.service to verify the service has the state of enabled.
5. Reboot your server, and after rebooting, verify that the container is automatically started.

**Partie 2 : exercice récap**

1. Assurez-vous d'être connecté pour accéder aux registres de conteneurs Red Hat.
2. Téléchargez l'image du conteneur mariadb sur l'ordinateur local.
3. Démarrez le conteneur mariadb en respectant les conditions suivantes :

* Le conteneur doit être accessible au port 3206.
* Le MYSQL\_ROOT\_PASSWORD doit être défini sur "mot de passe".
* Une base de données nommée mydb est créée.
* Un répertoire monté en liaison est accessible : le répertoire /opt/mariadb sur l'hôte doit être mappé sur /var/lib/mysql dans le conteneur.

1. Configurez systemd pour démarrer automatiquement le conteneur en tant « user systemd unit » au (re)démarrage de l'ordinateur.