Agenda:

* Run a hello-world container
* List the running and exited containers
* View the logs
* Pull Image from Dockerhub
* Start few containers
* Run a container in the interactive mode
* To delete the container
* To run a container with a web application
* Work with nginx web application
* Inspect the container for its network
* Send the request to the application running on 80 (Default port of the app)
* Create custom images
* Attaching volumes to containers

# Create and Start container 'c1' from Image hello-world

$ docker container run --name c1 hello-world

$ docker images

$ docker image ls

# List of all containers ('-a' for stopped containers)

$ docker ps -a

$ docker container ls

$ docker container ps -a

# List to see only see container IDS. ('q' for quiet mode)

# docker ps -aq

# See logs generated by the application running within the container

$ docker logs c1

To format the output of image ls command:

$ docker image ls --format "table {{.Repository}}\t{{.Tag}}\t{{.ID}}\t{{.Size}}"

To inspect any component in docker

1. docker inspect <container-name>/<image-name>/<network-name>/<volume-name>

# Remove Single container

$ docker container rm c1

# Remove all containers ($() command)

$ docker rm $(docker ps -aq)

$docker rm $(docker ps -a status=Exited)

# Pull the busybox image from docker hub

$ docker pull busybox

# List the image

$ docker image ls’

# Create and Start the container (Runs default sh command)

$ docker run --name box1 busybox

# Change the start execution command to ls

$ docker run --name box2 busybox ls

# Change the start execution command to echo

$ docker run --name box3 busybox echo how are you

# List all the created containers

$ docker ps -a

# Get the interactive terminal in the container

$ docker run -it --name box4 busybox

# Then, you enter in the shell within the container

/ #

# List all files in the container

/ # ls

# Create a new file using echo

/ # echo "my first file" > file1

# List all files

/ # ls

# Display content of file1

/ # cat file

# exit out of shell

/ # exit

# List the containers

$ docker ps -a

# Remove a single container

$ docker rm box1

==========================================================================

# To remove a container automatically after its execution is over add --rm to it.

$ docker run -it --rm busybox hostname -i

# 172.17.0.2 . This IP address is assigned under bridge network

# Create and start the container

$ docker run --name web1 -d nginx

# List the container

$ docker ps

# Inspect the container

$ docker inspect web1

# Get the IP address of the container to send request

IP Address = 172.17.0.2 (Donot type)

# Run the curl command

$ curl http://172.17.0.2

# To check the process running in nginx

$ docker top web1

# You will see two process

# PID USER . .... COMMAND

# 464 root ....

# 522 101 .... Worker-process ...

# list all processes running in Linux machine

$ ps

**Containers do not have their own operating system, so all process are run on the host machine.**

To start MYSQL container

docker run -d \

  --name mysql-container \

  -e MYSQL\_ROOT\_PASSWORD=my-secret-pw \

  -p 3306:3306 \

  mysql:latest

* -d: Detached mode
* --name: Assigns a name to the container
* -e MYSQL\_ROOT\_PASSWORD: Sets the root password
* -p 3306:3306: Maps MySQL port to host
* mysql:latest: Uses the latest MySQL image
* **Enter the Running Container**

docker exec -it mysql-container mysql -u root -p

* exec -it: Runs an interactive terminal
* mysql-container: Name of the container
* mysql -u root -p: Starts MySQL client as root (you’ll be prompted for the password)
* **Create a Database Inside MySQL**
* CREATE DATABASE my\_database;
* SHOW DATABASES;

To connect to a **MySQL container running in Docker Desktop** from **WSL (Ubuntu on Windows)**, follow these steps:

**✅ 1. Ensure MySQL Container is Running in Docker Desktop**

Run this in **PowerShell** or **Command Prompt** (not WSL):

This exposes MySQL on port 3306 of your **Windows host**.

**✅ 2. Find the Host IP from WSL**

In WSL, Docker Desktop is **not running natively**, so you need to connect to the **Windows host IP**.

Run this in WSL to get the Windows host IP:

You’ll get something like:

Use that IP (172.22.224.1 in this example) as the **host** when connecting to MySQL.

**✅ 3. Connect Using MySQL Client in WSL**

If MySQL client is installed in WSL, run:

* Replace 172.22.224.1 with the IP from step 2
* Enter the password (my-secret-pw) when prompted

**✅ Alternative: Use host.docker.internal (WSL 2)**

If you're using **WSL 2**, Docker Desktop supports host.docker.internal:

This is simpler and works in most recent Docker Desktop versions.

Network

Linux Container supports three types of network:

- 1. Bridge (172.17.0.1)

- 2. host (address of host machine is assigned to the container)

$ hostname

$ hostname -i

3. none

# By default every container is running in default network.

# Assign a container to a specific network

# Application running in the container, publish their port for the host machine

Syntax:

docker run or docker container run

Working of --publish option

Assign an host port number port

Host machine -------------->docker cli -----------> Container (nginx: 80)

8081 : 80

-p or --publish <host-machine address>: <port>

-P // ?

# Linking the two container (using docker network link)

# Create a custom network

$ docker network create net1 -d bridge --subnet 182.18.0.0/16

# inpsect the network

$ docker network inspect net1

# Create a container in the network

$ docker run -d --net net1 --name c1 nginx

$ docker network inspect net1

# Similarly, create another container

$ docker run -d --net net1 --name c2 nginx

docker run -d --net net1 --name c3 nginx

# Ping one container from another container

$ docker exec -it c1 /bin/sh

$ ls

$ curl http://182.18.0.3

$ ls -l /usr/share/nginx/html

# ping or send http request to a container with its name

# resolving names dynamically

$ curl c2

**To work with host network**

**to assign host network**

$ docker run -it --net host --rm busybox hostname -i

# 192.168.0.13 - Container is assigned with the IP address of Linux (host) machine

# to see this is host ip address

$ hostname -i

# 192.168.0.13

**Working with nginx container**

# To access container on local machine

$ docker run --name web1 -p 8080:80 nginx

# Get inside running container for trouble shooting

# Win: docker exec -it {CONTAINERNAME} powershell

# Linux: docker exec -it {CONTAINERNAME} sh

# Enter into running container using exec command

$ docker exec -it web1 sh

\_# ls -l /usr/share/nginx/html

\_# cd /usr/share/nginx/html

\_# rm index.html

\_# echo "<h2> Hello Priyanka </h2>" > index.html

# Refresh the web page from local machine browser to view the updated index.html page

$ docker stop c1

$ docker rm c1

**# Working with volumes**

============================================================

# To remove exited containers

$ docker rm $(docker ps -aq -f status=exited)

# Display list of volumes

$ docker volume ls

# Create a new volume

$ docker volume create webdata

# Inspect volume

$ docker volume inspect webdata

# Attach a volume to the container (:ro - readonly and rw: readwrite) - named volumes

$ docker run --name c1 -v my-data-vol:/usr/share/nginx/html:rw -p 8080:80 nginx

-v <<volume-name>> : folder within the container

# Add some data to webdata, by executing some commands in nginx

$ docker exec -it c1 sh

\_# cd /usr/share/nginx/html

\_# ls

\_# rm index.html

\_# echo "<h2> Hello Priyanka </h2>" > index.html

\_# exit

$ docker stop c1

$ docker rm c1

# Recreate a new container with same volume

$ docker run --name c1 -v webdata:/usr/share/nginx/html -p 8080:80 nginx

# Try accessing URL http://...:8080 from web browser

# Refresh the page to see updated content in the browser

# Clean up

$ docker rm -f c1

$ docker volume rm webdata

# Verify volumes

$ cd /var/lib/docker/volumes

**Creating Images**

A Dockerfile is a file that contain steps to create the container

Two main components of Dockerfile are:

# INSTRUCTION

# ARGUMENT

# INSTRCUTIONS - FROM, RUN, COPY, ENTRYPOINT, CMD, and so on

You pass arguments to the instructions

# Start image building from a base image

$ touch Dockerfile

# Build Docker image

Syntax:

**# docker build -t img\_from .**

# Scenario 1: We want to build an image based on ubuntu os and run the bash command on startup.

mkdir ubuntu

cd ubuntu

sudo vim Dockerfile

**ARG CODE\_VERSION=20.04**

**FROM ubuntu:${CODE\_VERSION}**

**RUN apt-get update -y**

**CMD ["bash"]**

Scenario 2: We want to build an image based on ubuntu and install curl command in it. Then remove temp files from the /var/lib/apt/lists/\* directory. Then, we need to set the environment variables

**FROM ubuntu:16.04**

**RUN apt-get update && apt-get install -y curl \**

**&& apt-get clean \**

**&& rm -rf /var/lib/apt/lists/\***

**RUN mkdir /home/Codes**

**ENV USER Joker-Canvas**

**ENV SHELL /bin/bash**

**ENV LOGNAME Joker-Canvas**

**CMD ["bash"]**

**# Build the image**

**$ docker build -t img\_run\_env .**

**# Run container**

**$ docker run -itd --name c2 img\_run\_env**

**$ docker exec -it c2 bash**

**$ # ls**

**$ # echo $USER**

**$ # echo $SHELL**

**$ # echo $LOGNAME**

**$ # cd /home**

**$ /home# ls**

**$ /home# exit**

**# Searching of Images**

**=====================================**

**1. docker search ubuntu**

**2. docker search --filter "is-official=true" registry**

**3. docker search --format "table {{.Name}}\t{{.Description}}\t{{.IsOfficial}}" ubuntu**

**# Listing Images**

**=======================================**

**1. docker images**

**2. docker image ls**

**3. docker images ubuntu:16.04**

**# You can create repository on Dockerhub**

**============================================**

**$ docker login**

**$ docker tag nginx:latest priya12345/repo-nginx:cc-nginx**

**$ docker image push priya12345/repo-nginx:cc-nginx**

**To push an image on docker repository**

**docker tag local-image:tagname new-repo:tagname**

**docker push new-repo:tagname**

**docker push priya123456/myrepo:tagname**

**# myubuntu is a tag to the image**

**$ docker tag img\_ubuntu:mywork priya123456/myrepo:myubuntu**

**$ docker login**

**# username - dockerid**

**# password -**

**$ docker push priya123456/myrepo:myubuntu**

**Login into Dockerhub**

**$ docker pull priya123456/myrepo:myubuntu**

**$ docker run -it --name c1 priya123456/myrepo:myubuntu**

# Scenario 3: Build an ubuntu image and install nginx in it

**FROM ubuntu:16.04**

**RUN apt-get update && apt-get install nginx -y \**

**&& apt-get clean \**

**&& rm -rf /var/lib/apt/lists/\***

**EXPOSE 80**

**CMD ["nginx", "-g", "daemon off;"]**

**$ docker build -t img\_expose .**

**$ docker run -itd --name web1 -p 8080:80 img\_expose**

**$ curl http://localhost:8080**

**Assignment:**

**# To create an python image to run python application**

**=============================================================**

**## Steps**

**1. OS- Ubuntu**

**2. Update apt repo**

**3. Install dependencies using apt**

**4. Install Python dependencies using pip**

**5. Copy source code to /opt folder**

**6. Run the web server using "flask" command**