Module 3

Purwadhika
Digital Technology School

Full Stack Web Development

Advanced Topic

Job Connector Program

Outline

- Docker Overview
- Install Docker on Windows, Linux, Mac
- Image
- Container
- Container vs Virtual machine
- Docker compose

Intro to Docker

What is Docker ?

Docker is a software platform that allows you to quickly build, test, and deploy applications. Docker packages software into standard units called containers that have everything the software needs to function including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale your app to any environment and have confidence that your code will run.

Intro to Docker - Docker vs VM

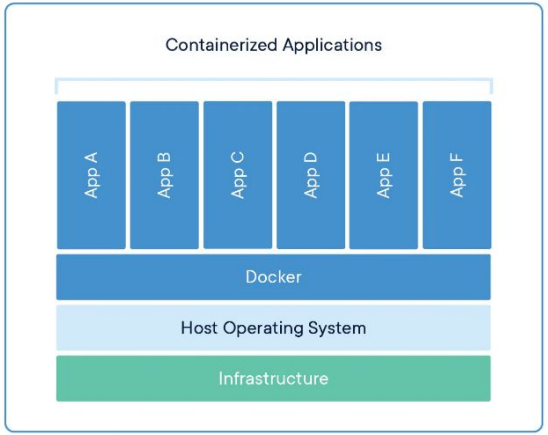
Docker

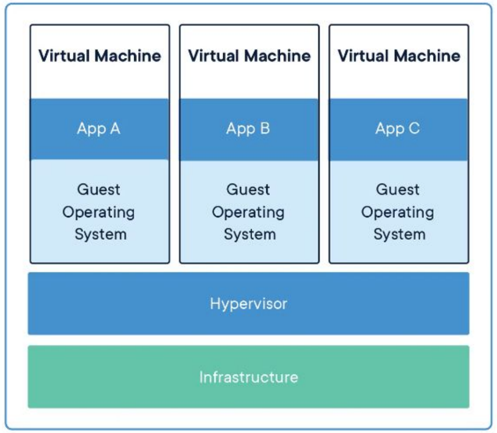
Allows developers to run applications that are owned in the form of containers on the operating system that is already installed on the server. So that on the server we no longer need to set up a VM, this is what causes Docker to be faster during the deployment process than a VM.

Virtual Machine

Every time we want to run an application on the server we have to prepare a virtual operating system first, and each application will usually be on a different virtual operating system on one computer. This is what causes it to take more time if we deploy the application to the VM.

Intro to Docker - Docker vs VM





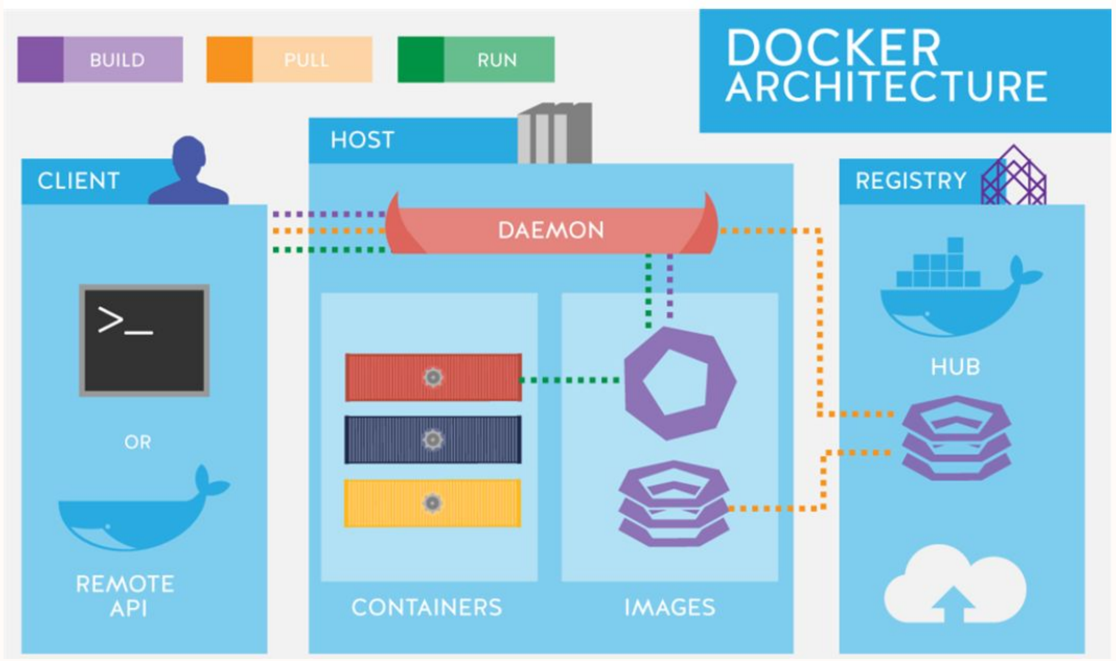
CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.

Intro to Docker - Docker environment



Usually docker contains Docker Client, and Docker Server / Host.

Docker Client - contain terminal or command to execute Docker command

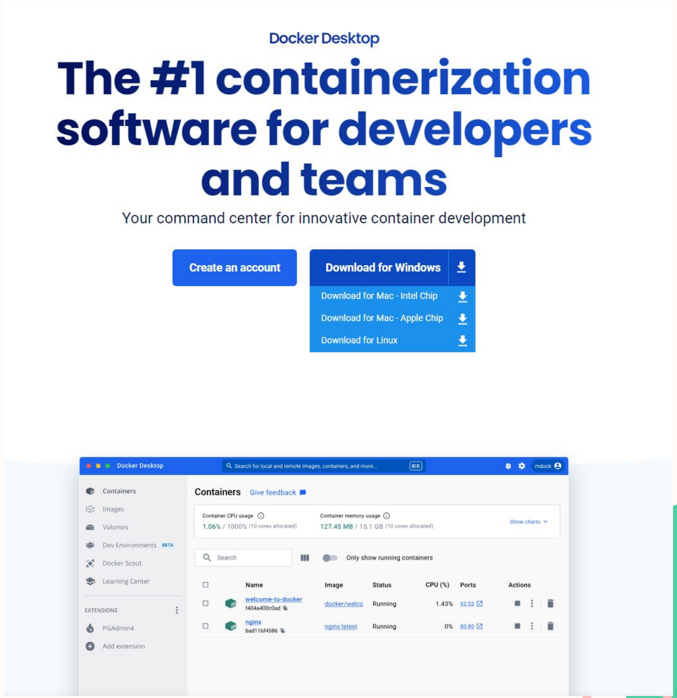
Docker Server - every request made by Docker Client, that would manage and executed using Docker Daemon.

Install Docker on Windows, Linux, Mac

Download from here :
https://www.docker.com/products/docker-desktop/

To run Docker in Windows we need to install **WSL** (windows subsystem linux) .

Why Docker needs Linux?
Docker is written in the Go programming languageopen_in_new and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called namespaces to provide the isolated workspace called the container.


Docker Desktop

The #1 containerization software for developers and teams

Your command center for innovative container development

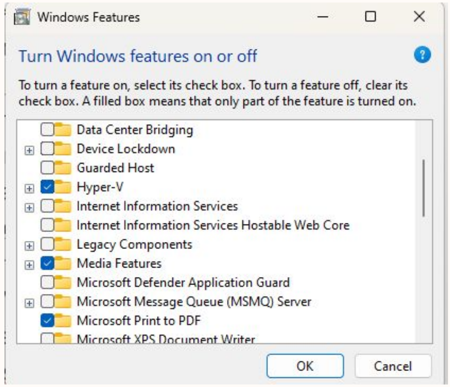
Containers

Container CPU usage 1.06% / 1000% (10 cores allocated)
Container memory usage 127.45 MB / 15.1 GB (10 cores allocated)

Name	Image	Status	CPU (%)	Ports	Actions
welcome-to-docker f404a400c0ad	docker/welco	Running	1.43%	52:52	
nginx bad116f4586	nginx:latest	Running	0%	80:80	

Install WSL2 on Windows

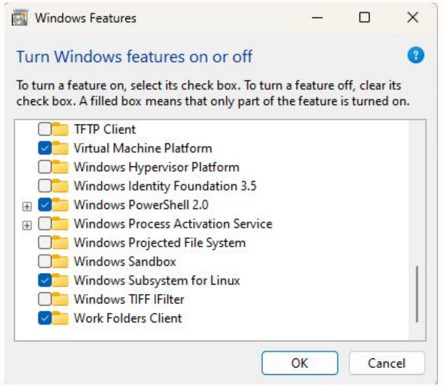
- Open PowerShell as Administrator (Start menu > PowerShell > right-click > Run as Administrator) and enter this command: **dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart**
- Before installing WSL 2, you must enable the Virtual Machine Platform optional feature. Your machine will require virtualization capabilities to use this feature. Run : **dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart**
- Download the Linux kernel update package. here : *download here*
- Open PowerShell and run this command to set WSL 2 as the default version when installing a new Linux distribution: **wsl --set-default-version 2**
- Install linux distribution on powershell. Here : **wsl --install -d Ubuntu-22.04**
- Setup username and password to start using ubuntu on windows
- Enable **virtual machine platform**, **windows subsystem for linux**, **hyper v** on windows features


Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ Data Center Bridging
- ☐ Device Lockdown
- ☐ Guarded Host
- ✓ Hyper-V
- ☐ Internet Information Services
- ☐ Internet Information Services Hostable Web Core
- ☐ Legacy Components
- ✓ Media Features
- ☐ Microsoft Defender Application Guard
- ☐ Microsoft Message Queue (MSMQ) Server
- ✓ Microsoft Print to PDF
- ☐ Microsoft XPS Document Writer


Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ TFTP Client
- ✓ Virtual Machine Platform
- ☐ Windows Hypervisor Platform
- ☐ Windows Identity Foundation 3.5
- ✓ Windows PowerShell 2.0
- ☐ Windows Process Activation Service
- ☐ Windows Projected File System
- ☐ Windows Sandbox
- ✓ Windows Subsystem for Linux
- ☐ Windows TIFF IFilter
- ✓ Work Folders Client

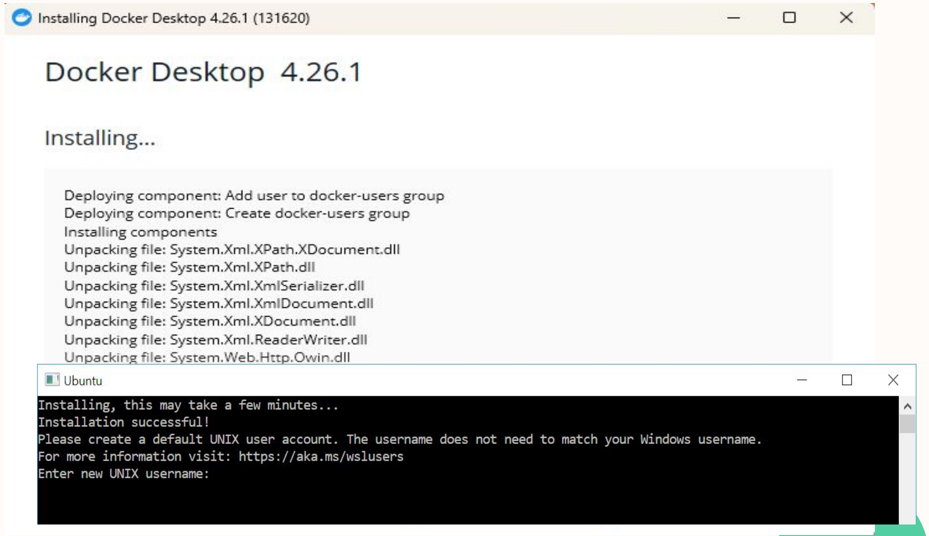
Ref :
https://learn.microsoft.com/en-us/windows/wsl/install-manual#step-3---enable-virtual-machine-feature

https://learn.microsoft.com/en-us/windows/wsl/install-manual#step-1---enable-the-windows-subsystem-for-linux

Install Docker Desktop on Windows

- Double-click Docker Desktop Installer.exe to run the installer.
- When prompted, ensure the Use WSL 2 instead of Hyper-V option on the Configuration page is selected or not depending on your choice of backend.
- If your system only supports one of the two options, you will not be able to select which backend to use.
- Follow the instructions on the installation wizard to authorize the installer and proceed with the install.
- When the installation is successful, select Close to complete the installation process.
- If your admin account is different to your user account, you must add the user to the docker-users group. Run Computer Management as an administrator and navigate to Local Users and Groups > Groups > docker-users. Right-click to add the user to the group. Sign out and sign back in for the changes to take effect.

Ref : https://docs.docker.com/desktop/install/windows-install/#install-docker-desktop-on-windows



```
Deploying component: Add user to docker-users group
Deploying component: Create docker-users group
Installing components
Unpacking file: System.Xml.XPath.XDocument.dll
Unpacking file: System.Xml.XPath.dll
Unpacking file: System.Xml.XmlSerializer.dll
Unpacking file: System.Xml.XmlDocument.dll
Unpacking file: System.Xml.XDocument.dll
Unpacking file: System.Xml.ReaderWriter.dll
Unpacking file: System.Web.Http.Owin.dll
```

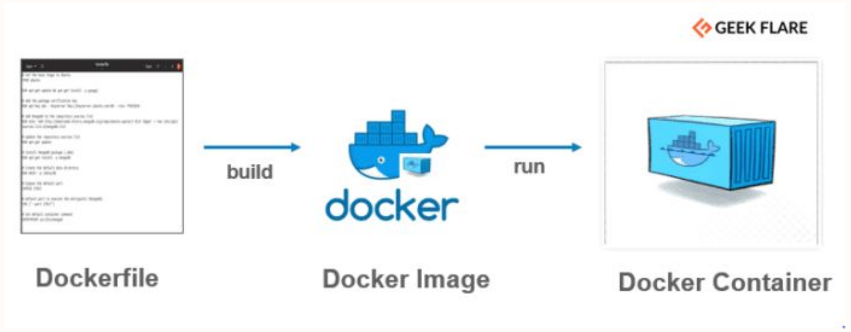
```
Installing, this may take a few minutes...
Installation successful!
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username:
```

Things you need to know in Docker !

- **Containers :** A container is an isolated environment for your code. (**your apps**)

- **Images :** A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. (**instances of a container**)

- **Volumes :** In the context of Docker, a volume is a persistent storage location that exists outside of the container (**storage for your container**)



Docker Client

The Docker client provides a **command line interface (CLI)** that allows you to issue build, run, and stop application commands to a Docker daemon.

The main purpose of the Docker Client is to provide a means to direct the pull of images from a registry and to have it run on a Docker host.

Docker CLI - Common Commands

docker build

- Build an image from a Dockerfile

docker images

- List all images on a Docker host

docker run —name container_name images_name

- Run an image to create container

docker start container container_name

- Start existing container

Docker CLI - Common Commands

docker restart container container_name

- Restart existing container

docker rmi images_name

- Delete a local images

docker ps

- List all running

docker ps -a

- List all containers

Docker Server

Docker Daemon - In charge of managing images, either making changes or deleting. And also manage all the containers that are in docker.

Docker Host - The primary server that provides and runs the Docker Daemon.

Docker Registry - A place to store docker images, can be stored in the docker daemon for personal storage or stored in the Docker Hub so that it can be used by others.

Docker Objects

- Images Project templates that have been built into docker images.
- Networks Docker network that is used as a medium of communication between docker containers
- Containers Portable application that runs from docker images.

Docker Registry

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

- **Docker Hub :** https://hub.docker.com/
- **Google Container Registry**: https://cloud.google.com/container-registry

Docker Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Docker Container

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

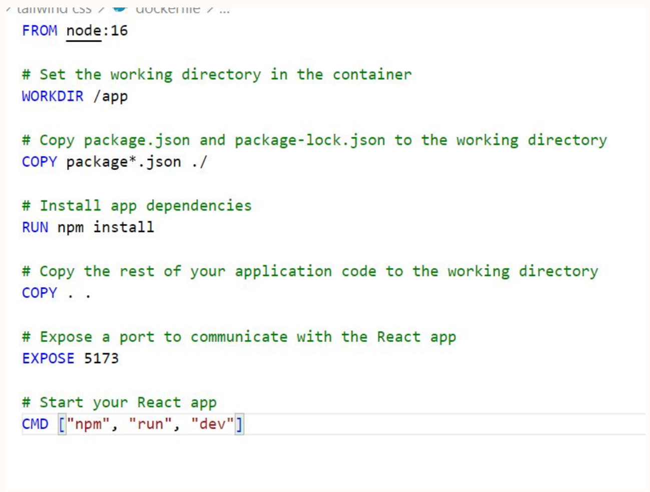
By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Dockerize your Project - Dockerfile

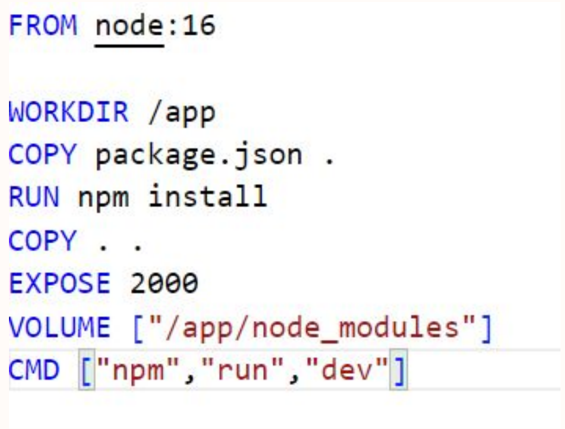
- Create a ‘dockerfile’ for your frontend/backend to stimulate the image of your apps

vite project



```
FROM node:16

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install app dependencies
RUN npm install

# Copy the rest of your application code to the working directory
COPY . .

# Expose a port to communicate with the React app
EXPOSE 5173

# Start your React app
CMD ["npm", "run", "dev"]
```

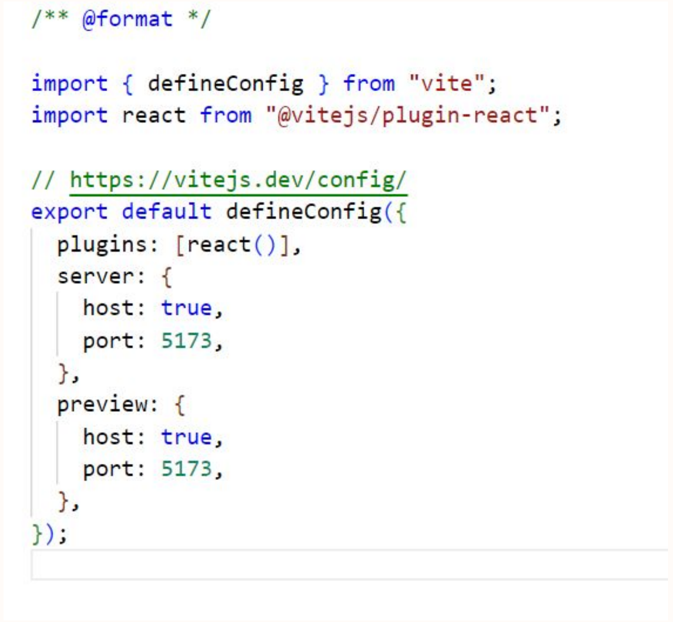
node js



```
FROM node:16

WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 2000
VOLUME ["/app/node_modules"]
CMD ["npm","run","dev"]
```

Dockerize your Project - Vite Dockerfile

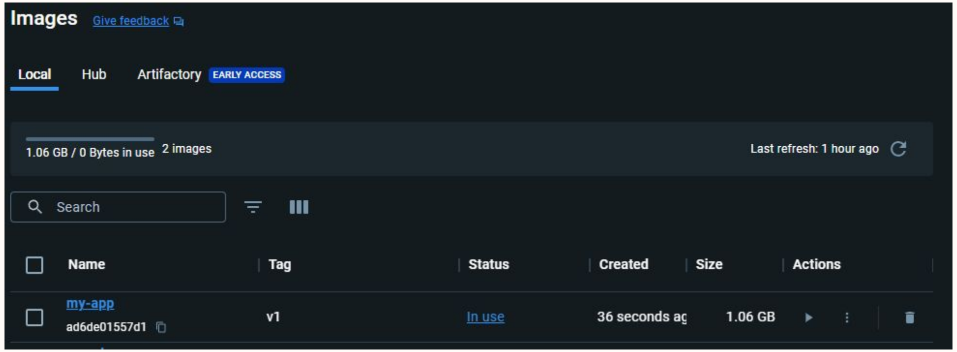
You have to setup server and preview on your vite-config.js to let the host access the project once it started as container.



```
/** @format */

import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    host: true,
    port: 5173,
  },
  preview: {
    host: true,
    port: 5173,
  },
});
```

Dockerize your Project - Build Docker Image

- After you create your dockerfile. Create your app image using this command :
- **docker build -t <Your App Image Name>:<version of the image app>** .
- Example : **docker build -t my-app:v1 .**

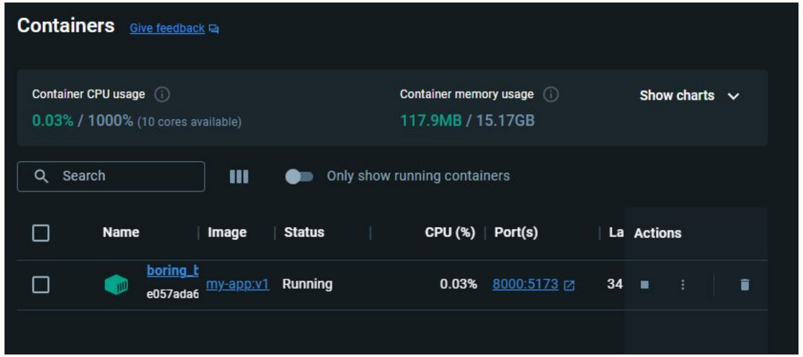


Name	Tag	Status	Created	Size	Actions
my-app ad6de01557d1	v1	In use	36 seconds ag	1.06 GB	

- Once it done, you can find your image app on your docker desktop

Crete and Run Docker Container

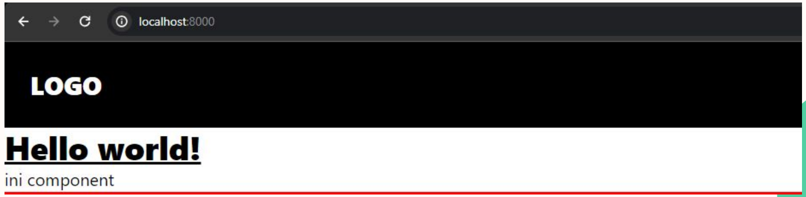
- Create your container from the images of your docker
- Run this command : **docker run -p <host port>:<docker port> <image_name>:<image_version> --name=container_name**
- Example : **docker run -p 8000:5173 my-app:v1**
- Now the container is running, so you can access your project through the host port.
- Check the logs to see if it works properly, by running this command : **docker logs <container_name>**


Containers Give feedback

Container CPU usage 0.03% / 1000% (10 cores available)
Container memory usage 117.9MB / 15.17GB
Show charts

Search
Only show running containers

Name	Image	Status	CPU (%)	Port(s)	La	Actions
boring_t e057ada6	my-app:v1	Running	0.03%	8000:5173	34	


localhost:8000

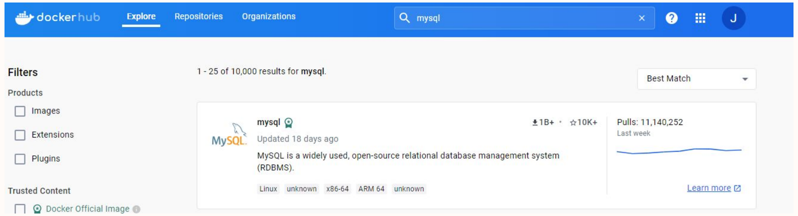
LOGO

Hello world!

ini component

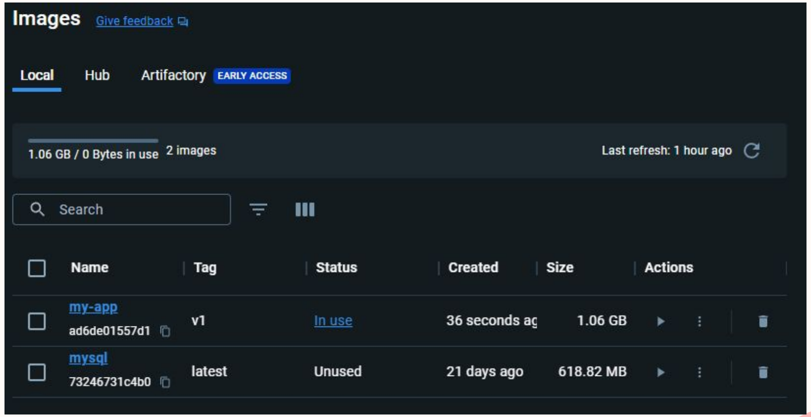
Find Docker Image in Docker Hub

- Find your images, for example search : mysql
- Find your image version
- Use **docker pull <image_name>:<version>**
- Example : **docker pull mysql:8.0**



1 - 25 of 10,000 results for **mysql**.

mysql
Updated 18 days ago
1B+ · 10K+
MySQL is a widely used, open-source relational database management system (RDBMS).
Linux unknown x86-64 ARM 64 unknown
Pulls: 11,140,252
Last week
Learn more



1.06 GB / 0 Bytes in use 2 images

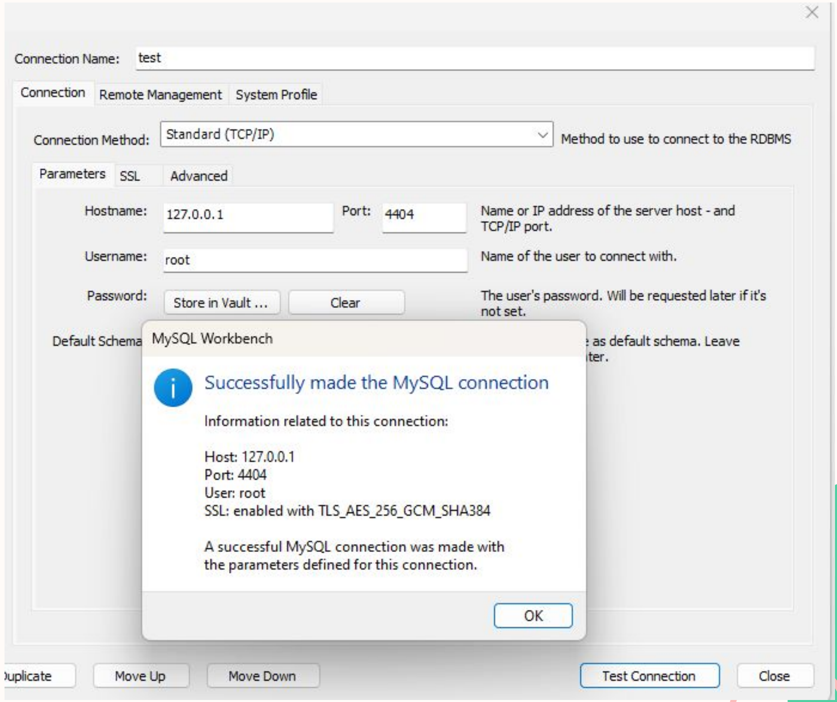
Last refresh: 1 hour ago

Name	Tag	Status	Created	Size	Actions
my-app ad6de01557d1	v1	In use	36 seconds ag	1.06 GB	
mysql 73246731c4b0	latest	Unused	21 days ago	618.82 MB	

Docker Container - MySQL

- **docker run --name <container_name> -e MYSQL_ROOT_PASSWORD=<your_password> -p <host_port>:<docker_port> mysql:<version>**

- Example: **docker run --name MYSQL_1 -e MYSQL_ROOT_PASSWORD=password -p 4404:3306 mysql:8.0.35**


Connection Name: test

Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

Hostname: 127.0.0.1 Port: 4404 Name or IP address of the server host - and TCP/IP port.

Username: root Name of the user to connect with.

Password: The user's password. Will be requested later if it's not set.

Default Schema

MySQL Workbench

Successfully made the MySQL connection

Information related to this connection:

Host: 127.0.0.1
Port: 4404
User: root
SSL: enabled with TLS_AES_256_GCM_SHA384

A successful MySQL connection was made with the parameters defined for this connection.

Access to Database

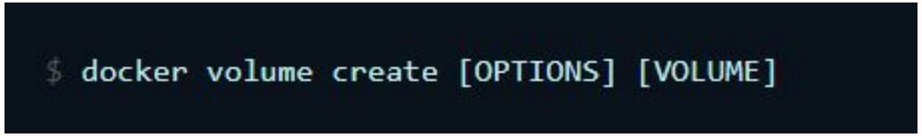
- Accessing database through host. Set your mysql setting **host : “host.docker.internal”.**
- You can also access your mysql through container name. **host: “container_name” .**

Docker Volume

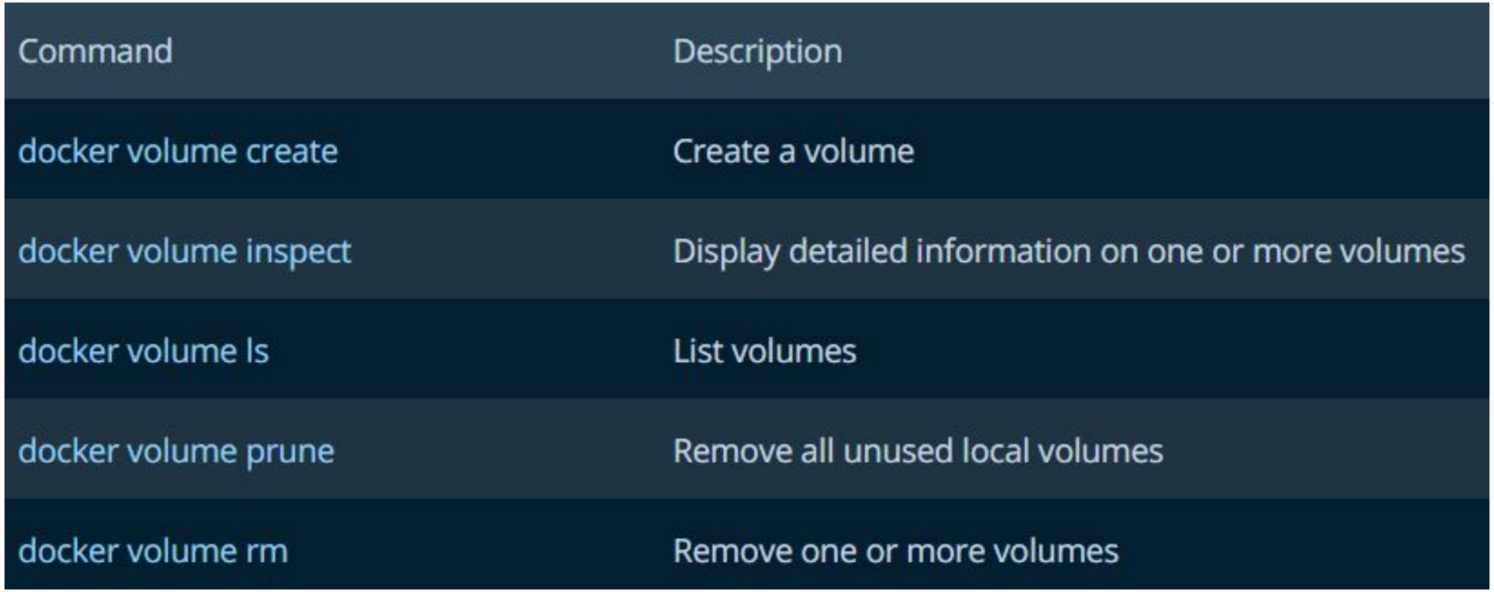
Docker volumes are a widely used and useful tool for ensuring data persistence while working in containers. Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container.

- The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. The data cannot be easily moveable somewhere else.
- Writing into a container's writable layer requires a storage driver to manage the filesystem.

Docker Volume



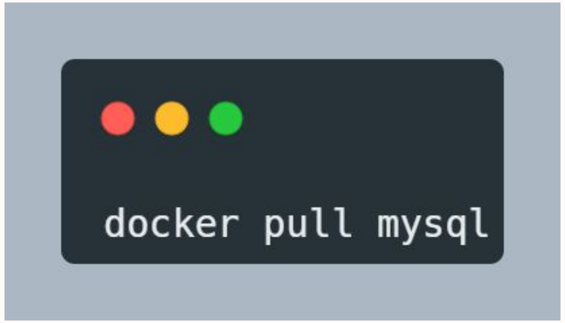
```
$ docker volume create [OPTIONS] [VOLUME]
```



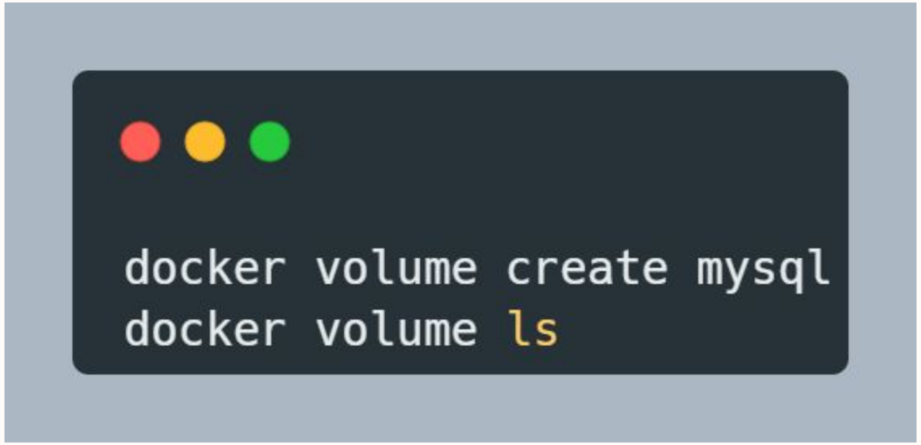
Command	Description
docker volume create	Create a volume
docker volume inspect	Display detailed information on one or more volumes
docker volume ls	List volumes
docker volume prune	Remove all unused local volumes
docker volume rm	Remove one or more volumes

Docker Volume

Lets pull mysql image from docker hub: https://hub.docker.com/_/mysql and create a new volume and name it as mysql and check if that volume is created successfully.



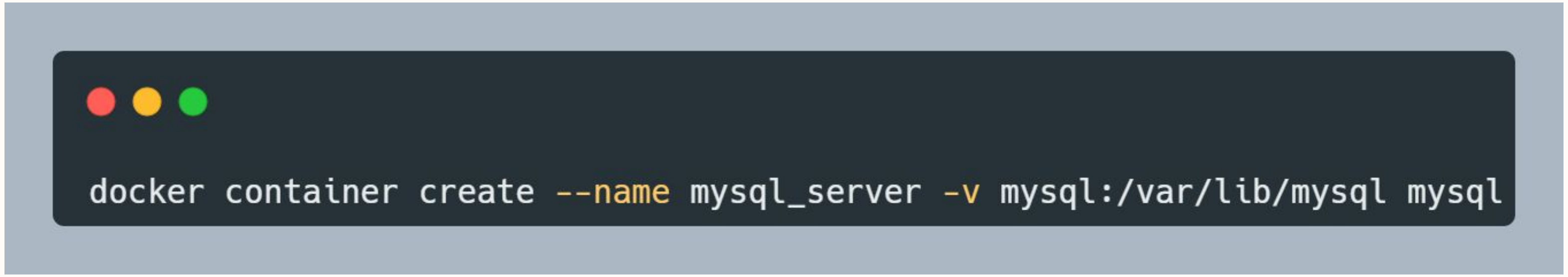
```
docker pull mysql
```



```
docker volume create mysql
docker volume ls
```

Docker Volume

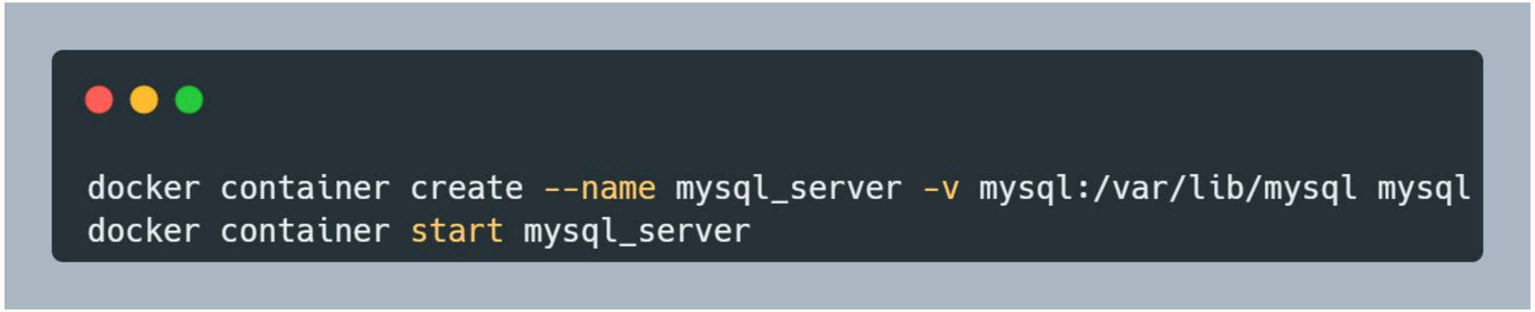
Create a new container based on mysql image, in this code we named it as mysql_server. There is -v argument, this used to define that our container would pointing into volume that has a name mysql.



```
docker container create --name mysql_server -v mysql:/var/lib/mysql mysql
```

Docker Volume

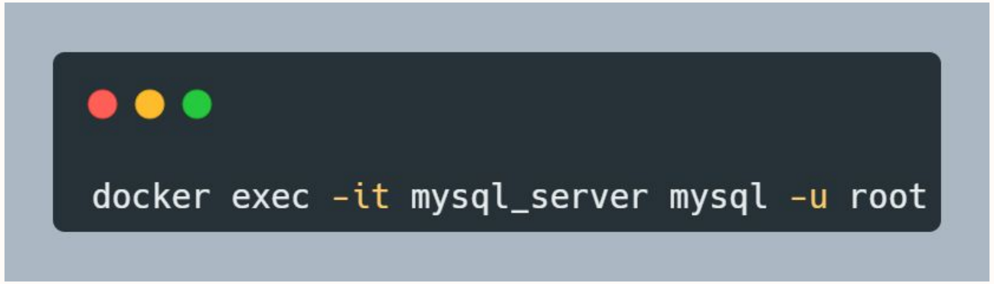
Create a new container based on mysql image, in this code we named it as mysql_server. There is -v argument, this used to define that our container would pointing into volume that has a name mysql. After that run the mysql_server container.



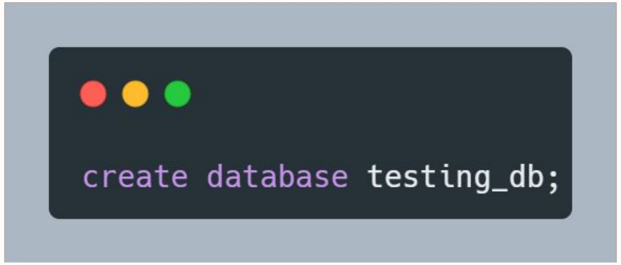
```
docker container create --name mysql_server -v mysql:/var/lib/mysql mysql
docker container start mysql_server
```

Docker Volume

Try accessing mysql from running container. By default we already have a root user in that mysql with no password. Lets create a new database.



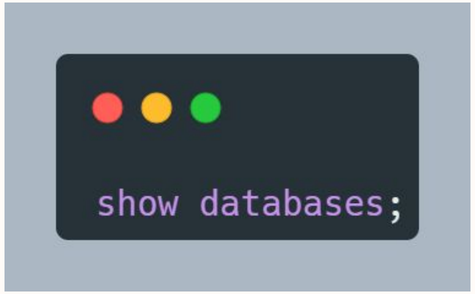
```
docker exec -it mysql_server mysql -u root
```



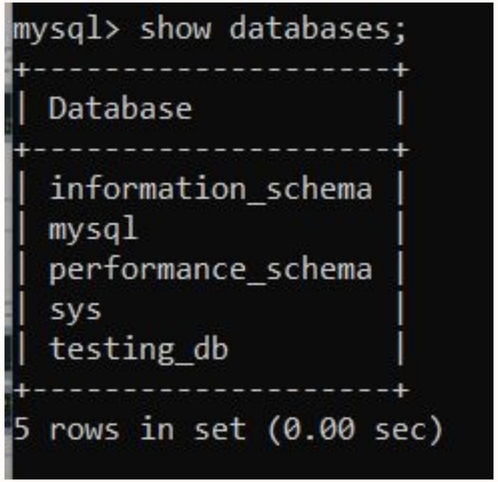
```
create database testing_db;
```

Docker Volume

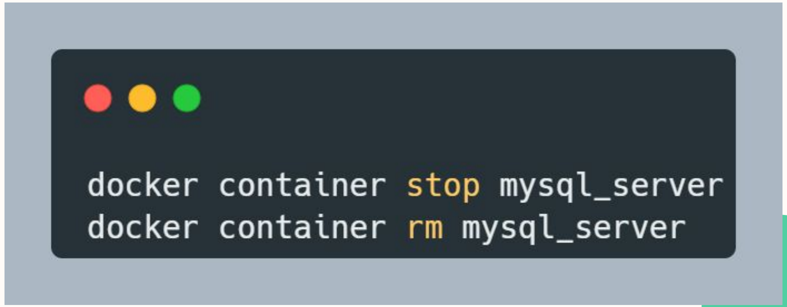
Let's check database that we just created before. After that, try to exit from mysql terminal. At this point, we will try to stop and remove our container with name mysql_server.



```
show databases;
```



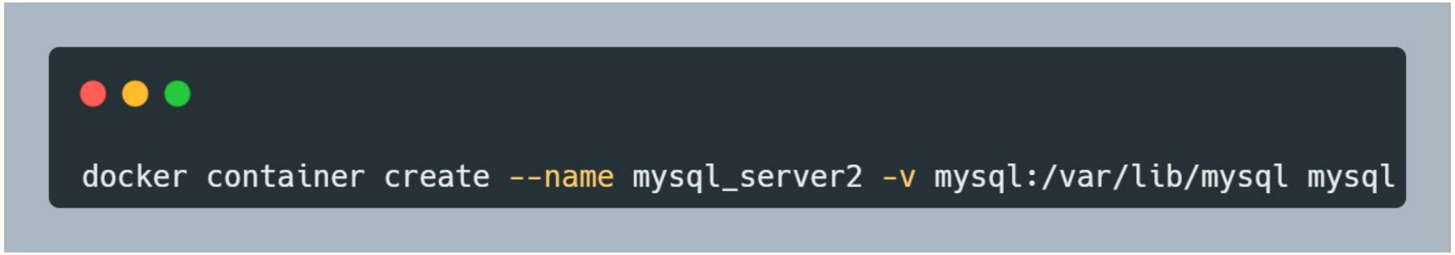
```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testing_db         |
+--------------------+
5 rows in set (0.00 sec)
```



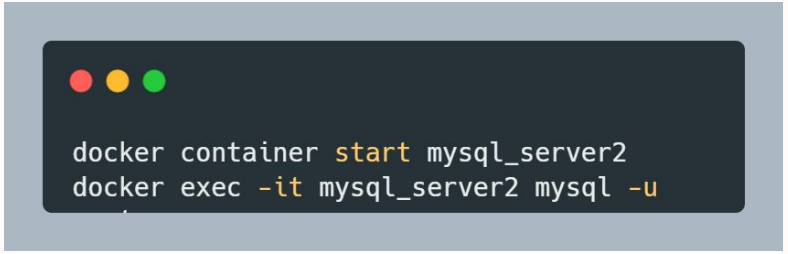
```
docker container stop mysql_server
docker container rm mysql_server
```

Docker Volume

After you remove the container try to create a new one container again but define the volume same as the last one. Start the container and check the list of database in this new container.



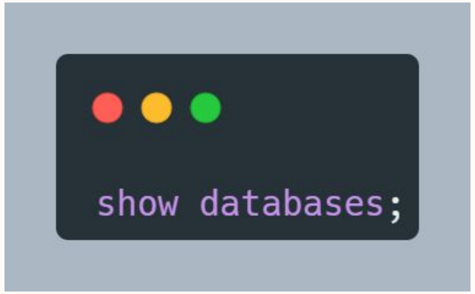
```
docker container create --name mysql_server2 -v mysql:/var/lib/mysql mysql
```



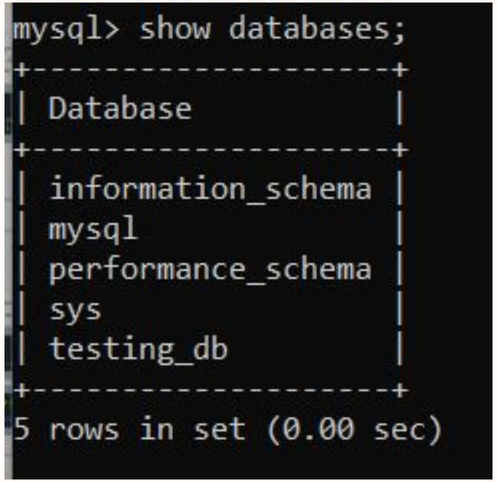
```
docker container start mysql_server2
docker exec -it mysql_server2 mysql -u
```

Docker Volume

As you can see, it would keep the database same as mysql_server container before. Its caused by detach volume into this new container.



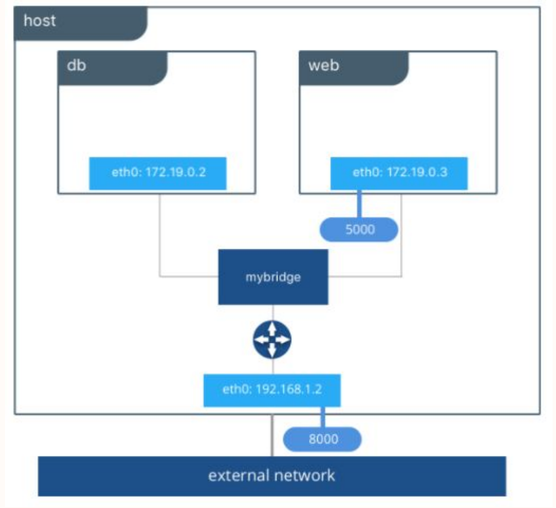
```
show databases;
```



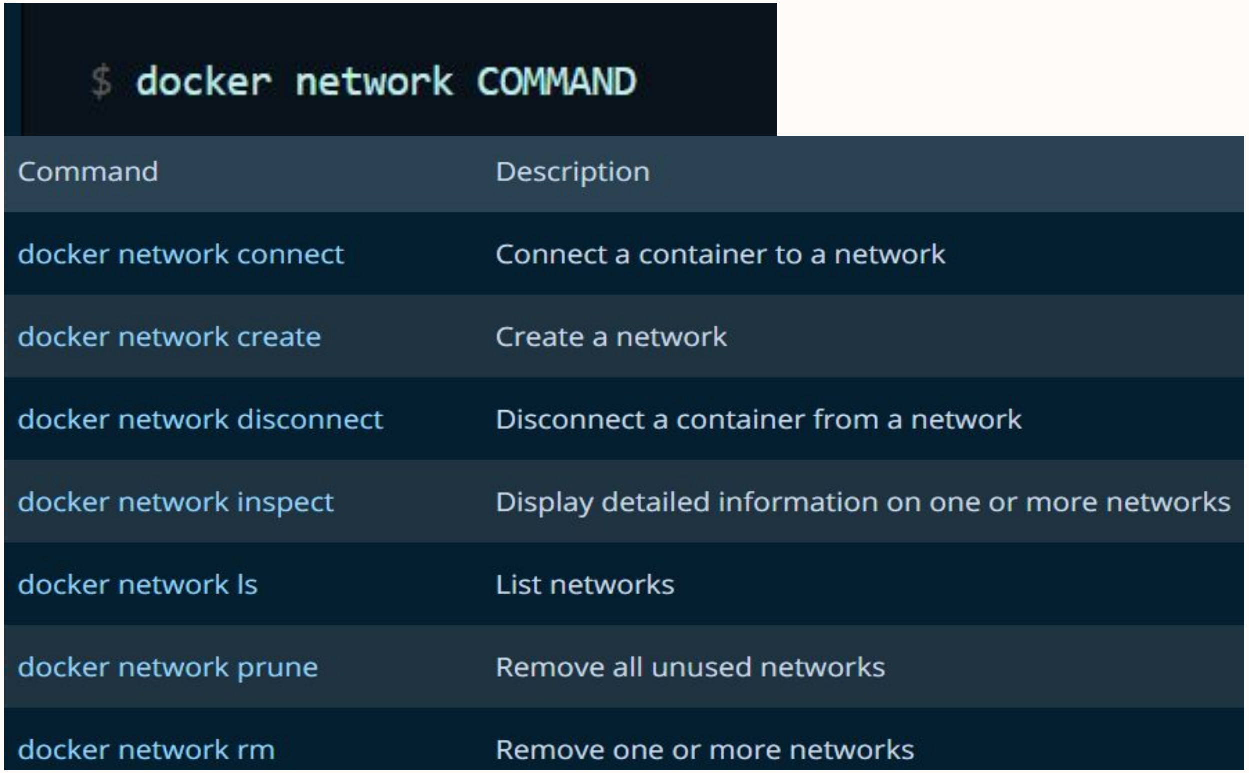
```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testing_db         |
+--------------------+
5 rows in set (0.00 sec)
```

Docker Network

Networking is about communication among processes, and Docker's networking is no different. Docker networking is primarily used to establish communication between Docker containers and the outside world via the host machine where the Docker daemon is running.



Docker Network



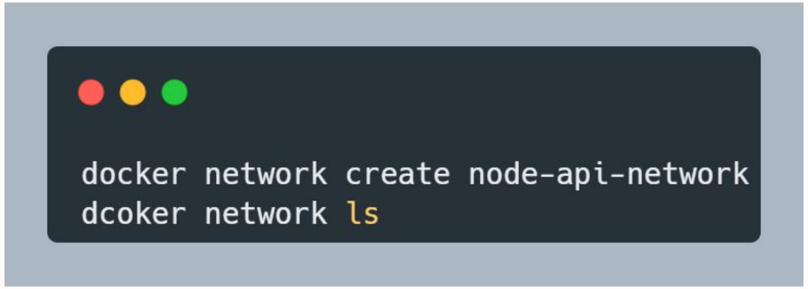
```
$ docker network COMMAND
```

Command	Description
docker network connect	Connect a container to a network
docker network create	Create a network
docker network disconnect	Disconnect a container from a network
docker network inspect	Display detailed information on one or more networks
docker network ls	List networks
docker network prune	Remove all unused networks
docker network rm	Remove one or more networks

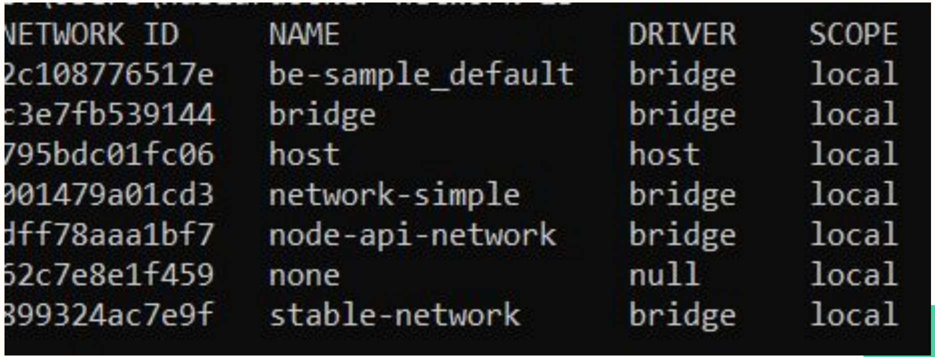
In order to manage networks, you can use subcommands to create, inspect, list, remove, prune, connect, and disconnect networks.

Docker Network

Lets create a new network in our docker. Write down this command to create and check if that network created successfully.



```
docker network create node-api-network
dcoker network ls
```

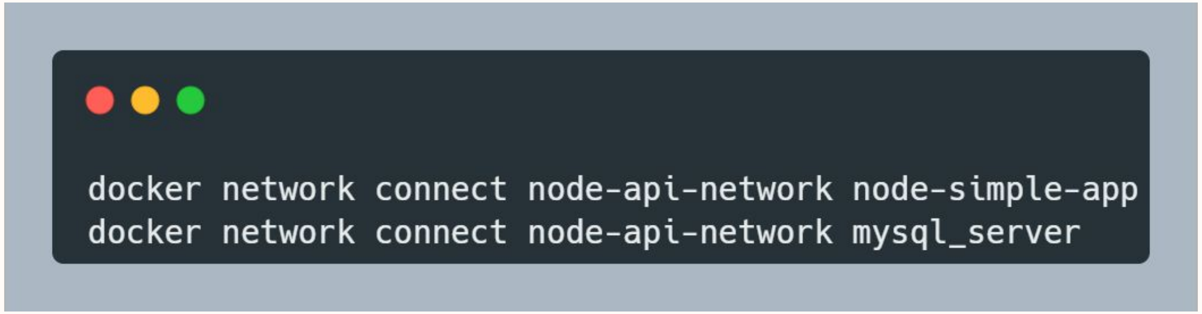


```
NETWORK ID     NAME                 DRIVER    SCOPE
2c108776517e   be-sample_default    bridge    local
c3e7fb539144   bridge               bridge    local
795bdc01fc06   host                 host      local
001479a01cd3   network-simple       bridge    local
dff78aaa1bf7   node-api-network     bridge    local
62c7e8e1f459   none                 null      local
399324ac7e9f   stable-network       bridge    local
```

Docker Network

Add container name that you would like to connect into network you just made before.

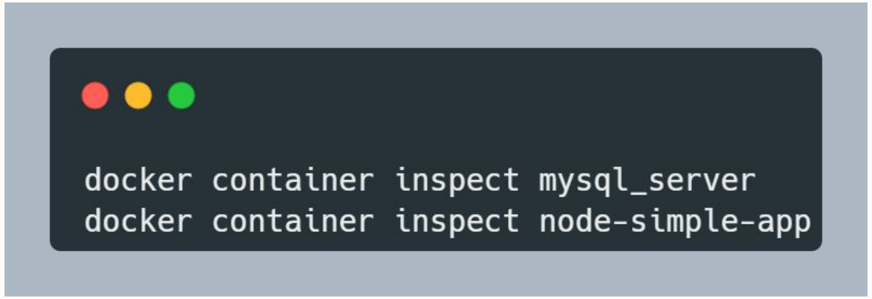
Lets connect node-simple-app and mysql_server into the same network.



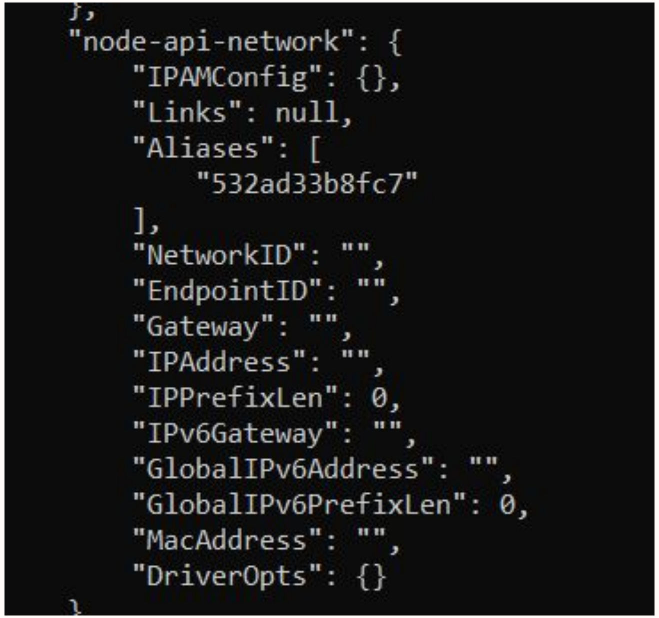
```
docker network connect node-api-network node-simple-app
docker network connect node-api-network mysql_server
```

Docker Network

Let's check if the container we just add is successfully connect with the network. Check each container using inspect argument.



```
docker container inspect mysql_server
docker container inspect node-simple-app
```



```
    },
    "node-api-network": {
        "IPAMConfig": {},
        "Links": null,
        "Aliases": [
            "532ad33b8fc7"
        ],
        "NetworkID": "",
        "EndpointID": "",
        "Gateway": "",
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "",
        "DriverOpts": {}
    }
```

Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Compose works in all environments: production, staging, development, testing, as well as CI workflows.

Docker Compose

Using Compose is basically a three-step process:

- Define your app’s environment with a Dockerfile so it can be reproduced anywhere.
- Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
- Run docker compose up and the Docker compose command starts and runs your entire app. You can alternatively run docker-compose up using the docker-compose binary.

Docker Compose

Compose has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Docker Compose - Compose Specification

The Compose file is a YAML file defining services, networks, and volumes for a Docker application. The latest and recommended version of the Compose file format is defined by the Compose Specification.

This document specifies the Compose file format used to define multi-containers applications. Distribution of this document is unlimited.

Docker Compose - Compose File

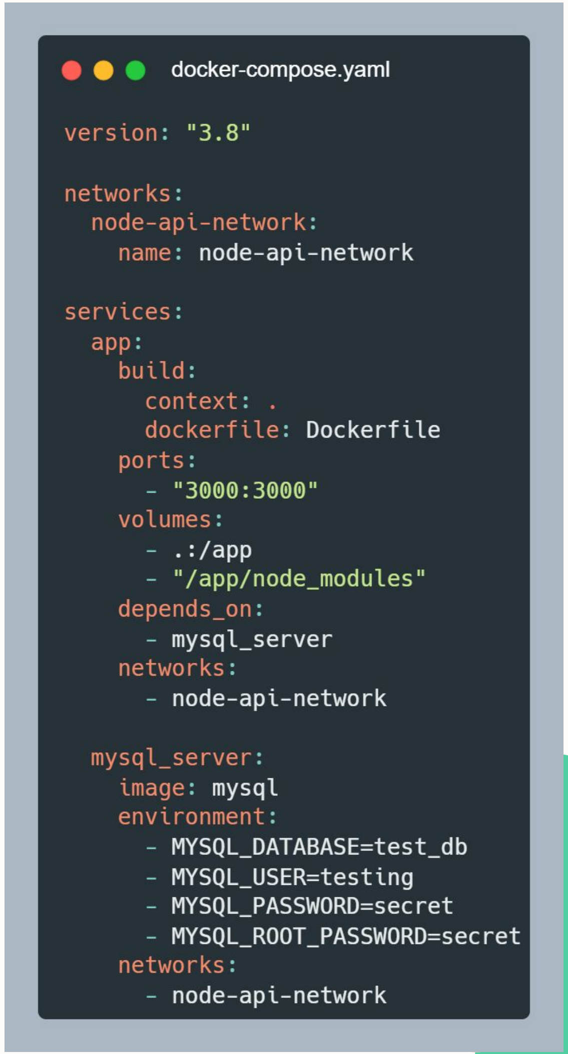
The Compose file is a YAML file defining:

- Version (DEPRECATED)
- Services (REQUIRED)
- Networks, volumes, configs and secrets.

The default path for a Compose file is compose.yaml (preferred) or compose.yml in working directory. Compose implementations SHOULD also support docker-compose.yaml and docker-compose.yml for backward compatibility. If both files exist, Compose implementations MUST prefer canonical compose.yaml one.

Docker Compose - Compose File

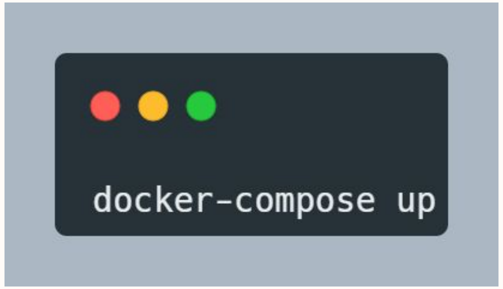
Lets go back to our projects, this time we will implement starting container node app and mysql using docker compose. Lets create file name as docker-compose.yaml and put this code into your file.


docker-compose.yaml

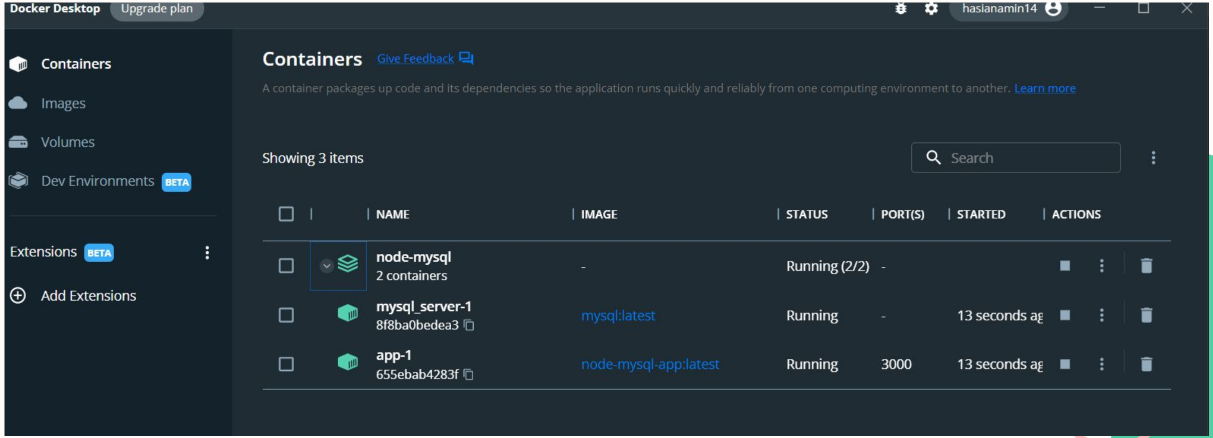
```yaml
version: "3.8"

networks:
  node-api-network:
    name: node-api-network

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - .:/app
      - "/app/node_modules"
    depends_on:
      - mysql_server
    networks:
      - node-api-network

  mysql_server:
    image: mysql
    environment:
      - MYSQL_DATABASE=test_db
      - MYSQL_USER=testing
      - MYSQL_PASSWORD=secret
      - MYSQL_ROOT_PASSWORD=secret
    networks:
      - node-api-network
```

Docker Compose - Compose File

Lets run your container, use docker-compose command to take an action for docker-compose.yaml file. You can check through docker desktop for running container. But this time, there is more than one container running and wrapped by one container



```
docker-compose up
```



Containers

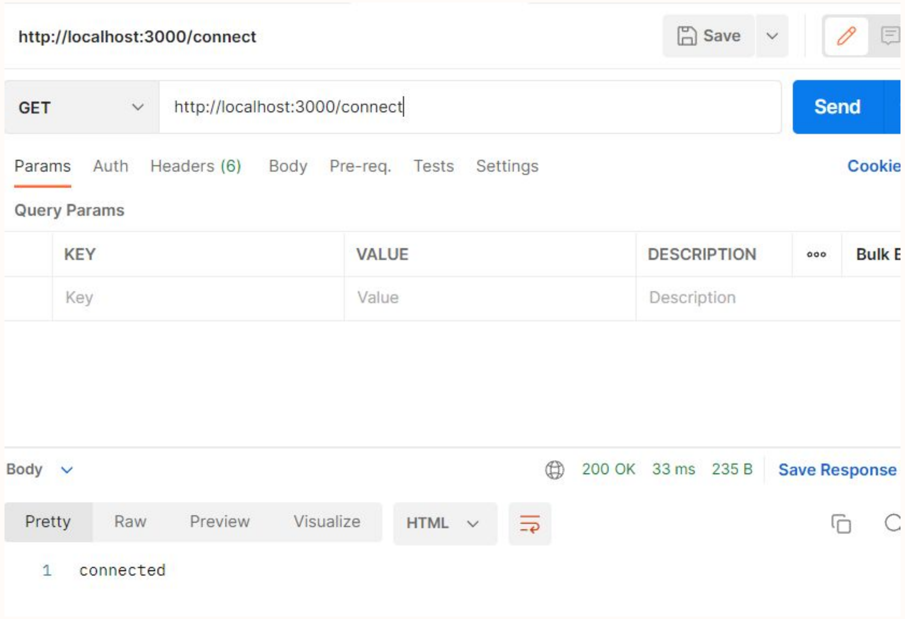
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 3 items

NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
node-mysql 2 containers	-	Running (2/2)	-		
mysql_server-1 8f8ba0bedea3	mysql:latest	Running	-	13 seconds ag	
app-1 655ebab4283f	node-mysql-app:latest	Running	3000	13 seconds ag	

Docker Compose - Compose File

Lets try and test our projects through postman!



http://localhost:3000/connect

GET http://localhost:3000/connect

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body 200 OK 33 ms 235 B

```
connected
```

Thank You!