# Docker Fundamentals Workshop

In this workshop, we will be working on 4 different projects. We need

* Docker Desktop
* Dbeaver
* Postman

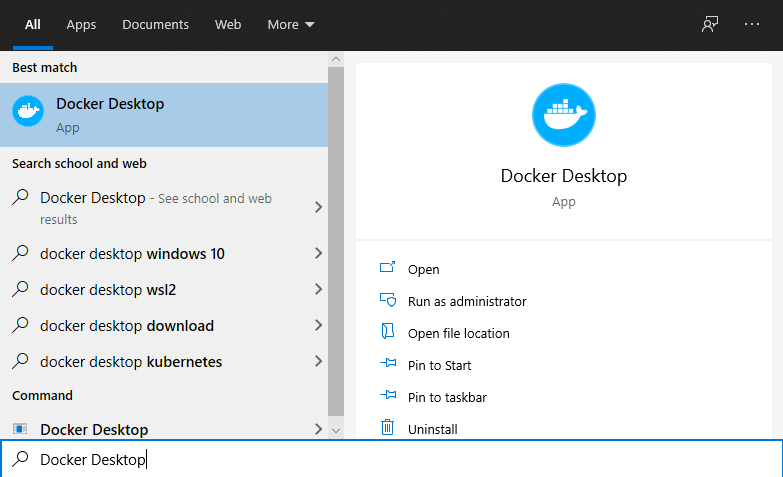
# Install Docker in windows

1. Visit the official Docker website (<https://www.docker.com/products/docker-desktop>) and download Docker Desktop for your operating system (Windows or macOS) install it.
2. Verify Installation

open a terminal or command prompt and run the command docker --version

## start docker in windows to work in the sample projects

* In the Search type docker desktop



* Docker desktop will open as below

A screenshot of a computer

Description automatically generated

# Project 1: Build and run a docker file to build basic python application

* This project is a simple python project. The project folder is ‘python\_basic’

### contents of the project

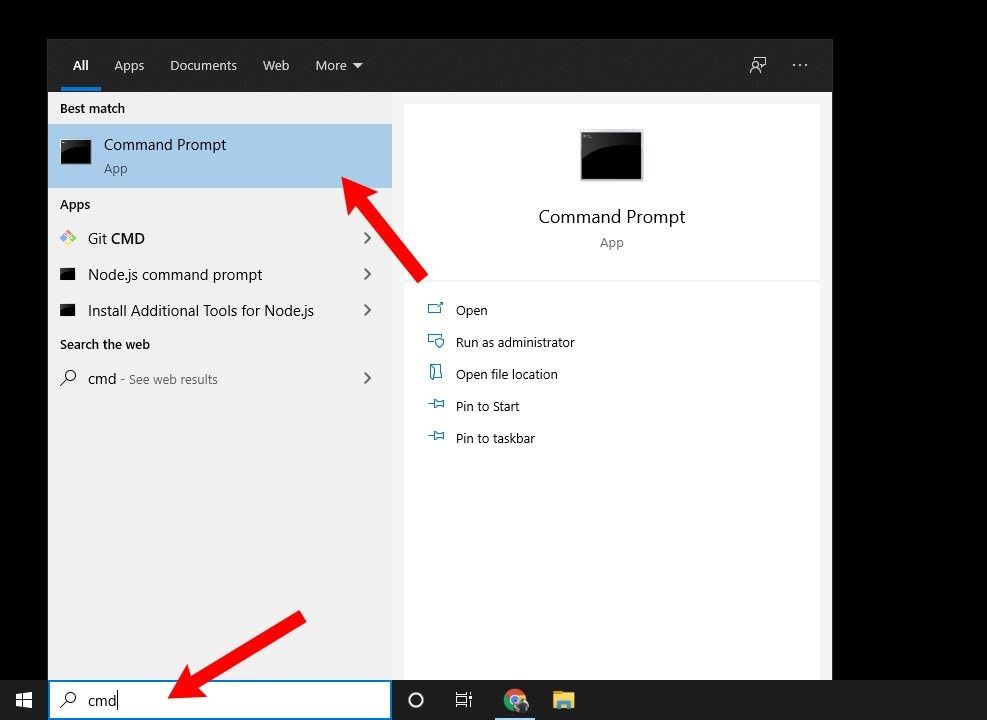
A screenshot of a computer

Description automatically generated

* DockerFile 🡪 Dockerfile is a text files which contains instructions on how to build a Docker image for our python applcation.
* hello.py 🡪 python file
* requirements.txt 🡪 dependency libraries required by the project

### create docker image

1. Open command prompt in windows



1. From command prompt, navigate to the ‘python\_basic’ folder

cd C:\Users \Desktop\workshop\python\_basic

A black and white screen with white text

Description automatically generated

1. Type the below command in the command prompt to create a docker image for this project

**docker build -t first\_app .**

A computer screen with blue text

Description automatically generated

This step will create a docker image for this project, verify it in the docker desktop.

This step will download python and all the library mentioned in the docker file

A screenshot of a computer

Description automatically generated

### run docker container

1. Run the Docker container using the image, we created above, using the below command

**docker run first\_app**

A screen shot of a computer

Description automatically generated

This step will run a docker container for this project, verify it in the docker desktop

A screenshot of a computer

Description automatically generated

1. We have successfully run our first project in docker

**Description for the docker build command**

|  |
| --- |
| Syntax: docker build -t <image\_name > .  The command ‘docker build -t <image\_name> . ‘ builds a Docker image using the Dockerfile in the current directory and tags the image with the specified name |

**Description for the docker run command**

|  |
| --- |
| Syntax: docker run <image\_name >  Description: This command runs a Docker container using the image. When you run this command, Docker starts a new container based on the specified image. |

# Project 2: Start SQL in docker

* This project is a simple mySQL database project. The project folder is ‘sql’

### contents of the project

A screenshot of a computer

Description automatically generated

|  |
| --- |
| * docker-compose.yml 🡪 This is a YAML file that defines the services, networks, and volumes for Docker application. In our case, it will define a MySQL service, specifying details like the image to use, environment variables, ports to expose, etc. * init.sql 🡪 This is an SQL script file used to initialize or set up your MySQL database. * query\_to\_test.txt 🡪 SQL script to check on our database. |

### start docker compose container

1. From command prompt, navigate to the ‘sql’ folder

cd C:\Users \Desktop\workshop\sql

1. start the docker container for our docker compose using the below command

docker compose up -d

A screen shot of a computer

Description automatically generated

1. Check if our sql container is started using the command

docker ps

A black screen with white text

Description automatically generated

From the UI also we can see the container is running

A screenshot of a computer

Description automatically generated

### Test our SQL server using Dbeaver

1. Open Dbeaver

A screenshot of a computer

Description automatically generated

1. From dbeaver we create a new connection and database connection

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Select mysql in the database

A screenshot of a computer

Description automatically generated

1. Now open the docker-compose.yml file to check the db configuration for this project
2. Enter the connection details as given in the ‘docker-compose.yml’ file.

Server Host : localhost

Port : 3307

Database: employeedb

UserName : root

Password: root\_pass

A screenshot of a computer

Description automatically generated

Note: In the ports, given in the docker\_compose file we are specified “3307:3306”

The Syntax for ports is , **"host\_port:container\_port"**.

* The port mentioned first ‘3307’ is the port, db is deployed on host machine (our computer), that is the reason we are connecting to 3307 via dbeaver
* The port ‘3306’ is where the mysql is deployed within the docker container

1. In the Driver properties set the below and click finish

allowPublicKeyRetrieval 🡪 TRUE

A screenshot of a computer

Description automatically generated

1. Open SQL Script to execute the queries given in the file ‘query\_to\_test.txt’ file one by one and test our sql db

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Stop the docker compose container

Stop the docker container we created with docker compose

docker compose down

docker ps

A black screen with white text

Description automatically generated

# Project 3: MySQL\_Flask Application

* This project is a FlaskAPI project designed to create Flask API web services. It facilitates performing CRUD (Create, Read, Update, Delete) operations on a MySQL database using Postman.
* The project folder is ‘sql\_flaskapi’
* ***Note: In this project, using docker-compose.yml, we are starting both the containers with a single command ‘docker compose up’ first starting the DB, then FlaskAPI . Refer the keyword depends\_on, (our flask API depends on DB, so we start DB first)***

### contents of the project

A screenshot of a computer

Description automatically generated

|  |
| --- |
| * **docker-compose.yml** 🡪 it contains services for the Flask API (app) and MySQL database (db). * init.sql 🡪 This is an SQL script file used to initialize or set up your MySQL database. * postman\_json\_input\_to\_test.txt 🡪 JSON data input to test our FlaskAPI. * ‘app’ directory, that contains FlaskAPI related files.   + app.py-> Python file containing the Flask application code.   + Dockerfile-> Instructions for building the Docker image for the Flask application.   + requirements.txt-> Dependency libraries required by the Flask application. |

A screenshot of a computer

Description automatically generated

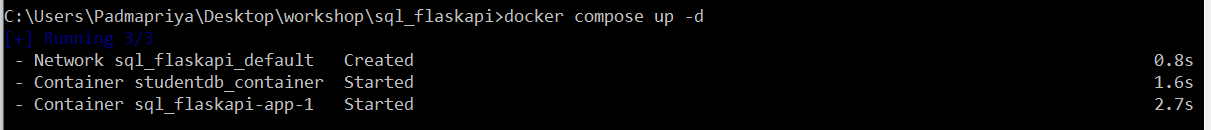
### start docker compose container

1. From command prompt, navigate to the ‘sql\_flaskapi’ folder

cd C:\Users \Desktop\workshop\sql\_flaskapi’

1. start the docker containers, defined in the docker-compose.yml using the below command

docker compose up -d



Note: The above command will start MySQL first and FlaskAPI next, The order to start the application is defined in the docker-compose.yml file

1. Re-Check if our both containers are started using the command

docker ps

A black screen with white text

Description automatically generated

From the UI also we can see both the containers are running

A screenshot of a computer

Description automatically generated

### Test our API using Postman

1. Open Postman

A screenshot of a computer program

Description automatically generated

1. Use the payload given in the file ‘postman\_json\_input\_to\_test.txt’ of our project folder
2. GET Request will fetch the list of students present in our db via API,
   1. Copy the url, select GET from the dropdown menu and click the send button

A screenshot of a computer

Description automatically generated

1. POST Request will add new student to our db via API
   1. Copy the url, select POST from the dropdown menu
   2. Copy the JSON input given in the ‘postman\_json\_input\_to\_test.txt’ in the BODY tab,
   3. select raw radio button and json from dropdown
   4. click the send button , a new student record will be created in our db
   5. In the response we can see success message

A screenshot of a computer

Description automatically generated

1. PUT Request is used to update the details of students in our db via API
   1. In the Url, we have to pass the ID of the student, that we want to update the details.
   2. Pass the updated payload in the BODY and click send
   3. In the response we can see success message

A screenshot of a computer

Description automatically generated

1. DELETE Request is used to delete the details of students in our db via API
   1. In the Url, we have to pass the ID of the student, that we want to delete.
   2. In the response we can see success message

A screenshot of a computer

Description automatically generated

### Stop the docker compose container

1. Stop the docker container (DB and FlaskAPI) we created with docker compose using the below command and verify with docker ps command as below

docker compose down

docker ps

A black screen with white text

Description automatically generated

# Project 4: MySQL\_Flask\_NextJS Application

* This project is a NEXTJS, Flask API, SQL integrated full stack project designed to utilise Flask API web services to connect to db via UI. It facilitates performing CRUD (Create, Read) operations on a MySQL database using UI page.
* The project folder is ‘sql\_flaskapi\_nextjs
* ***Note: In this project, using docker-compose.yml, we are starting 3 containers with a single command ‘docker compose up’ , Refer the keyword depends\_on, (our flask API depends on DB, so we start DB first, then Flask API and finally we start the NextJS)***

|  |
| --- |
| * **docker-compose.yml** 🡪 it contains services for the Flask API (app), NextJS, and MySQL database (db) * init.sql 🡪 This is an SQL script file used to initialize or set up your MySQL database. * ui\_url\_to\_test.txt 🡪 url of our nextJS ui page. * ‘app’ directory, that contains FlaskAPI related files.   + app.py-> Python file containing the Flask application code.   + Dockerfile-> Instructions for building the Docker image for the Flask application.   + requirements.txt-> Dependency libraries required by the Flask application. * ‘next\_app’ directory, that contains UI related items |

### start docker compose container

1. From command prompt, navigate to the ‘sql\_flaskapi\_nextjs’ folder

cd C:\Users \Desktop\workshop\sql\_flaskapi\_nextjs’

1. start the docker containers, defined in the docker-compose.yml using the below command

docker compose up -d

A computer screen with white text

Description automatically generated

From the UI also we can see all 3 containers are running

A screenshot of a computer

Description automatically generated

1. Using our web browser, we can call the url, defined in the file ‘ui\_url\_to\_test.txt’

<http://localhost:3000/>

* 1. Our UI calls our API which internally send data to DB.

A screenshot of a student form

Description automatically generated

### Stop the docker compose container

1. Stop all the docker container (DB, FlaskAPI, NextJS) we created with docker compose using the below command and verify with docker ps command as below

docker compose down

docker ps

A black screen with white text

Description automatically generated