**//--------------------------Docker installation on VM-Ware(Ubuntu) -------------------**

**sudo apt-get update** // to make sure system is up to date

**sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual** //Before installing docker need to install recommended packages

**sudo apt-get install -y linux-image-extra-virtual**  //previous command is not working use this as replacement

**docker install** // to get command to install docker

**sudo apt install docker.io** // this command can be used for installation of docker result of previous command

**//--------------------------Docker Basic Commands ------------------------------------------**

**docker - - version**  // to check if docker is installed successfully

**docker - - help**

**docker container** // to container management commands

**docker build -t my-image-name-to-create .** // use for building image from docker file ***-t my-node-app*** *tags the image with the name* ***my-node-app* Dot(.)**specifies the **current directory** as the build context.

**docker run hello-world** // to run hello-world image. Perform pull if cant be found locally

**docker run -p 4200:4200 -d my-node-app** // **-p** defines port mapping of docker and local machine **-d** run container in detached mode means in background not blocking terminal

**docker images** // allow you to check docker images you have in your system

**docker pull ubuntu** // to pull ubuntu image from docker hub

**docker run -it -d ubuntu** // create container from an image and run it. Q: What is difference between simple run and -it -d run command??

**docker ps -a** // to show all containers

**docker exec -it container\_Id\_Here bash** // accessing a running container by id. To get id use

**winpty docker exec -it container\_Id\_Here bash** //use if above not working will be needed if you are using git bash terminal

**docker ps -a** Container details

**docker attach container\_name** //accessing running container simple version but auto stop container on exit

**docker stop container\_Id\_Here** // to stop a container

**docker commit container\_Id\_Here Your\_DockerHub\_Id/ImageNme(ubuntu)**  // Use container Id to create new image from a container and give it name. Use **docker images** command to check new created images

**docker login** // to login to docker hub

**docker push name\_of\_Image\_Prepared\_With\_docker\_commit\_command** // to push image to docker hub

**docker rm container\_id\_here** // to delete a docker container by id

**docker rmi image\_id\_here** //to delete an image

**//------------------------------------------------------------------------**

You can inspect where a named volume is stored on your host by running the following command:

**docker volume inspect volume\_name**

This command will provide information about the volume, including its mount point on the host machine.

**//------------------------------------------------------------------------**

Use Docker Network Inspect: You can inspect the Docker network to ensure that the containers are correctly attached:

**docker network ls**

**docker network inspect <network-name>**

**//--------------Creating Simple python application using docker Compose---------**

You need 🡪 codeFile, DockerFile, RequirementsFile, Docker\_Compose.yml

Then navigate to your files thorugh CLI🡪 cmd/Powershell and run command

**docker-compose up** //building image from files

**//--------------------------Docker file -------------------------------------------------------------**

**FROM ubuntu** // it means having a base image/machine on which your other functionalities are built

**RUN apt-get install -y riak** // used to run commands during image build

**CMD “echo” “Welcome to Edureka!”** //similar to **RUN** command but used in image execution after build is completed

**ENTRYPONT echo** //This will be first command that will be executed when you run docker container of that image or used for replacing/overriding functionality defined by CMD command

**ADD /path\_source /path\_destination** //ADD or COPY can be used interchangeable because ADD copy flies from place to other for example copying files from host to container

**ENV SERVER\_WORKS 4** //Environment command used to specify variables needed by application

**WORKDIR path** // specify path for executing commands

**EXPOSE path** // to expose an application front end on specific port for access inside a container if you want to specify host port you need to perform port mapping

**MAINTAINER authors\_name** // if you want to specify your name along with the image you are building before you upload it to docker hub to show who is the person who is maintaining the image. Only use after FROM command

**USER 751** // if you want to specify a user to run or execute a container specify that user using user id

**VOLUME [MY FILES]** // used to specify path to store files related to your docker contaner

**//--------------------------Building an image from DockerFile ----------------------------**

Use following code to create a docker file

*FROM ubuntu*

*MAINTAINER muhammadtabish*

*RUN apt update*

*RUN apt-get install -y apache2*

*RUN service apache2 start*

*ENV APACHE\_RUN\_USER www-data*

*ENV APACHE\_RUN\_GROUP www-data*

*ENV APACHE\_LOG\_DIR /var/log/apache*

*EXPOSE 80*

*CMD ["usr/sbin/apache2","-D","FOREGROUND"]*

Navigate to folder where docker file is using cli and run following command

**docker build -t image\_name .** // This command is used for building image out of docker file.dot(.) means the DockerFile is in this folder

**docker run -p 80:80 - -name=containerName imageName** //This command is building container from specified image.—name is used for specifying container which is to be built.-p is mapping port 80 of container to port 80 of host first is host than container.

Now you can see output at localhost:80

**Important FOR ACCESSING React or Angular app from docker container:**

While using commands **npm run dev** for react and **ng serve** for angular it will host application on local host of docker which will not be accessed even if you have done port mapping so you need to host/serve at **0.0.0.0:Port**

**Above is not applicable for node app**

**Important for communication between MongoDB and a Node.js:**

You will be needing a separate MongoDB container you can pull latest image for this  
For Node js make changes in your MongoDB connection string **replace localhost:Port with mongo:Port**

**mongo** in above is container name or alias of mongoDB conatainer

**Enable communication between MongoDB and a Node.js application in Docker using Docker Compose**

This is the recommended approach, where you use Docker Compose to manage multiple containers, each responsible for a specific service:

**Create a Dockerfile for Node.js**:

Here’s an example Dockerfile for a Node.js application:

FROM node:latest

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY . /usr/src/app

RUN npm install

RUN npm cache clean --force

EXPOSE 3000

CMD ["./start.sh"]

# *docker build -t mcqs\_backend .*

# *docker run -p 3000:3000 -d mcqs\_backend*

**start.sh File:**

#!*/bin/sh*

*node* ./src/dataBaseSetUp/dbSetup.js

*node* ./src/index.js

**Create a docker-compose.yml file**: This file defines both the Node.js and MongoDB services:

version: "3"

services:

  mongo:

    image: mongo:latest

    # *volumes:*

    # *- mongo\_data:/data/db*

    ports:

      - "27017:27017"

  app:

    image: mcqs\_backend

    # *environment:*

    # *- MONGO\_URI=mongodb://mongo:27017/MCQsShuffle*

    depends\_on:

      - mongo

    ports:

      - "3000:3000"

# *volumes:*

# *mongo\_data:*

**Directory Structure**: Your project directory should look something like this:

├── Dockerfile

├── docker-compose.yml

├── package.json

├── package-lock.json

├── src

└── ...

**Build and Start Containers**: Run the following command in your project directory:

**docker-compose up –build**

Or

**docker-compose up**

**Enable communication between MongoDB and a Node.js application in Docker without using Docker Compose**

Replace localhost with mongo for connecting monogodb to nodejs in connection string in nodejs app

**Run MongoDB Container:**

Start by running a MongoDB container with a specific name, so you can refer to it in your Node.js container.

**docker run -d --name mongodb-container -p 27017:27017 mongo:latest**

 -d: Runs the container in detached mode (in the background).

 --name mongodb-container: Names the container mongodb-container for easy reference.

 -p 27017:27017: Maps port 27017 on the host to 27017 in the container

### ****Run Node.js Container:****

Now, run your Node.js container, linking it to the MongoDB container by using the --link option or by referring to the MongoDB container by its name.

**docker run -d --name nodejs-app -p 3000:3000 --link mongodb-container:mongo mcqs\_backend**

 --link mongodb-container:mongo: This creates a link between your Node.js container and the MongoDB container. Inside the Node.js container, you can refer to the MongoDB instance with the hostname mongo.

 mcqs\_backend: The name of your Node.js image.

### ****Update MongoDB Connection String:****

**Replace localhost with mongo which is mongodb container name or alias**

Ensure that your Node.js application uses the hostname mongo instead of localhost in the MongoDB connection string, like so:

**mongoose.connect('mongodb://mongo:27017/MCQsShuffle', { useNewUrlParser: true, useUnifiedTopology: true });**

### ****Check the Connection:****

After running the Node.js container, check the logs to see if the connection to MongoDB is successful:

**docker logs nodejs-app**

### Notes:

* **Network Mode:** If you don't want to use --link, you can create a Docker network and connect both containers to this network. This is more flexible and modern.

**docker network create mynetwork**

**docker run -d --name mongodb-container --network mynetwork mongo:latest**

**docker run -d --name nodejs-app --network mynetwork mcqs\_backend**

* Then, in your Node.js application, you can still use mongodb://mongo:27017/MCQsShuffle.
* **Port Mapping:** Ensure that the ports (27017 for MongoDB and 3000 for Node.js) are correctly mapped and not being used by other services on your host machine. Use -p to add port mapping in above.

By following these steps, you can enable communication between MongoDB and a Node.js application running in separate Docker containers without needing Docker Compose.

**Hosting frontend without build:**

Specify host 0.0.0.0 for hosting app instead of localhost

**Create a docker file as following (For react vite app):**

FROM node:latest

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY . /usr/src/app

RUN npm install

RUN npm cache clean --force

EXPOSE 5173

CMD ["npm", "run", "dev"]

# *docker build -t mcqs\_frontend .*

# *docker run -p 5173:5173 -d mcqs\_frontend*

**Serving with already created build:**

**Docker file will be following (For react vite app biuld):**

FROM nginx:alpine

# *Copy the build files from the previous stage*

COPY ./dist /usr/share/nginx/html

# *Expose port 80*

EXPOSE 80

# *Start Nginx*

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

# *docker build -t mcqs\_frontend\_already\_build .*

# *docker run -p 5173:5173 -d mcqs\_frontend\_already\_build*

**Create build and serve using docker (Image Optamization):**

**Docker file will be following for multistage build (For react vite app):**

# *Stage 1: Build the React application*

FROM node:18 AS build

# *Set the working directory*

WORKDIR /app

# *Copy package.json and package-lock.json (or yarn.lock)*

COPY package\*.json ./

# *Install dependencies*

RUN npm install

# *Copy the rest of the application code*

COPY . .

# *Build the React application*

RUN npm run build

# *Stage 2: Serve the React application using Nginx*

FROM nginx:alpine

# *Copy the build files from the previous stage*

COPY --from=build /app/dist /usr/share/nginx/html

# *Expose port 80*

EXPOSE 80

# *Start Nginx*

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

# *docker build -t mcqs\_frontend\_buid .*

# *docker run -p 5173:5173 -d mcqs\_frontend\_build*