**# 1 Lightstreamer Container**

docker run -d --name ls-server -p 80:8080 lightstreamer:latest

**# 2 Ghost Container**

docker run -d --name ghost-container -p 3001:2368 -e NODE\_ENV=development ghost:latest

**# 3 Apache HTTP Server Container**

**# cd in new folder then run the command:**

docker run --name my-apache-app -p 8080:80 -d -v ${PWD}:/usr/local/apache2/htdocs/ httpd:latest

**# 4 SQL Server Container**

docker run -e ACCEPT\_EULA=Y --name sql -e MSSQL\_SA\_PASSWORD=yourStrongPassword12# -p 1433:1433 -d -v sqldata:/var/opt/mssql mcr.microsoft.com/mssql/server

**5 \*MariaDB Client and Server in a Network**

docker run -d --network mariadb\_network --name mariadb\_server --env MARIADB\_USER=mariadb\_user --env MARIADB\_PASSWORD=mariadb\_password --env MARIADB\_ROOT\_PASSWORD=mariadb\_root\_password mariadb:latest

docker run -d --network mariadb\_network --name mariadb\_client -e APP\_DB\_HOST=mariadb -e APP\_DB\_USER=mariadb\_user -e APP\_DB\_PASSWD=mariadb\_password -e MARIADB\_ROOT\_PASSWORD=mariadb\_root\_password mariadb:latest

# open container

docker exec -it mariadb\_client bash

# grant access in db

mariadb -h mariadb\_server -u maria\_user -p

# in db

SELECT VERSION();

**7 Tracker App**

1 - docker login

2 - docker push georgilukanov87/track-app

3 - docker build -t georgilukanov87/track-app

4 - docker build -t georgilukanov87/track-app .

5 - docker push georgilukanov87/track-app

6 - docker run -d --name track-app -p 8080:80 georgilukanov87/track-app

**8 TaskBoard App: Connect Containers in a Network**

# create network

1 - docker network create taskboard\_network

#run server. NAME and PASSWORD are IMPORTANT!

2 - docker run -e ACCEPT\_EULA=Y -e MSSQL\_SA\_PASSWORD=yourStrongPassword12# -p 1433:1433 -v sqldata:/var/opt/mssql --network taskboard\_network --name sqlserver -d mcr.microsoft.com/mssql/server

# change settings in appsetings.json in /TaskBoard.WebApp

3 - "ConnectionStrings": {

"DefaultConnection": "Server=sqlserver;Database=MyDB;User Id=sa;Password=yourStrongPassword12#;MultipleActiveResultSets=true"

}

#build image locally

4 - docker build . -f ./TaskBoard.Webapp/Dockerfile -t georgilukanov87/taskboard\_app

# run both together! check with : docker network inspect taskboard\_network

5 - docker run -p 5000:80 --name web\_app --network

taskboard\_network georgilukanov87/taskboard\_app

# test <http://localhost:5000/>

dsada

**9 TODO App**

docker network create react-express

docker network create express-mongo

docker build frontend -f frontend/Dockerfile -t frontend

docker build backend -f backend/Dockerfile -t backend

docker run -d --name frontend -p 3000:3000 --network react-express frontend

docker run -d --name backend --network react-express backend

docker network connect backend express-mongo

docker run -d --name mongo --network express-mongo -v ./data:/data/db mongo:latest

**10 TaskBoard App: Orchestrating Containers with Docker Compose**

# Create docker-compose.yaml and custom network

1 - docker network create taskboard\_network

2 - docker-compose build

3 - docker-compose up

4 - docker-compose down --rmi all --volumes

**11 Blue VS Green App**

11.Blue-VS-Green-App/result/Dockerfile

FROM node:10-slim

# Add Tini for proper init of signals

ENV TINI\_VERSION v0.19.0

ADD [https://github.com/krallin/tini/releases/download/${TINI\_VERSION}/tini /tini](https://github.com/krallin/tini/releases/download/$%7BTINI_VERSION%7D/tini%20/tini)

RUN chmod +x /tini

WORKDIR /app

# have nodemon available for local dev use (file watching)

RUN npm install -g nodemon

COPY result/package\*.json ./

RUN npm ci \

&& npm cache clean --force \

&& mv /app/node\_modules /node\_modules

COPY . .

ENV PORT 80

EXPOSE 80

CMD ["/tini", "--", "node", "server.js"]

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

11.Blue-VS-Green-App/vote/Dockerfile

# Using official python runtime base image

FROM python:3.9-slim

# Set the application directory

WORKDIR /app

# Install our requirements.txt

COPY vote/requirements.txt /app

RUN pip install --upgrade pip

RUN pip install --no-cache-dir -r requirements.txt

# Copy our code from the current folder to /app inside the container

COPY . .

# Make port 80 available for links and/or publish

EXPOSE 80

# Define our command to be run when launching the container

CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]

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

11.Blue-VS-Green-App/worker/Dockerfile

FROM mcr.microsoft.com/dotnet/core/sdk:3.1 as builder

# Create a working directory

WORKDIR /worker\_app

# Copy the .csproj file and restore

COPY worker/src/Worker/Worker.csproj /worker\_app

RUN dotnet restore

# Copy source files to the image

COPY worker/src/Worker /worker\_app

# Build the project

RUN dotnet publish -c Release -o /out Worker.csproj

# Specify app image

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1

# Specify working directory for this stage

WORKDIR /app

# Tell Docker what command to run when our image is executed inside a container

ENTRYPOINT ["dotnet", "Worker.dll"]

# Copy the /out directory from the build stage into the runtime image

COPY --from=builder /out .

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

Docker-compose.yaml:

version: "3.8"

services:

vote:

image: vote\_img

container\_name: vote

build:

dockerfile: vote/Dockerfile

command: python vote/app.py

volumes:

- pg-data:/var/lib/postgresql/data

ports:

- 5000:80

networks:

- backend

- frontend

restart: on-failure

result:

image: result\_img

container\_name: result

build:

dockerfile: result/Dockerfile

command: nodemon result/server.js

volumes:

- pg-data:/var/lib/postgresql/data

ports:

- 5001:80

networks:

- backend

- frontend

restart: on-failure

worker:

image: worker\_img

container\_name: worker

build:

dockerfile: worker/Dockerfile

networks:

- backend

restart: on-failure

redis:

image: redis

container\_name: redis

networks:

- backend

restart: on-failure

db:

image: postgres

container\_name: postgres\_db

environment:

- POSTGRES\_USER=postgres

- POSTGRES\_PASSWORD=postgres

networks:

- backend

volumes:

- pg-data:/var/lib/postgresql/data

restart: on-failure

volumes:

pg-data:

networks:

backend:

frontend:

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

docker network create frontend

docker network create backend

docker -d up --build

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

SimplePageApp/Dockerfile

# Use an official Python runtime as a parent image

FROM python:3.8-slim-buster

# Set the working directory in the container

WORKDIR /app

# Set environment variables

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

# Copy the requirements file into the container at /app

COPY requirements.txt /app/

# Install any needed packages specified in requirements.txt

RUN pip install --upgrade pip && pip install --no-cache-dir -r requirements.txt

# Copy the current directory contents into the container at /app

COPY . /app/

RUN mkdir /app/staticfiles

COPY static /app/staticfiles

# Expose port 8000 for the Django development server

EXPOSE 8000

# Run the application

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

* **13 Fruitipedia App**

#build image

docker build -t frutipedia-app .

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

#Create docker-compose.yaml in the same folder

SimplePageApp/docker-compose.yaml

version: '3.8'

services:

web:

container\_name: Frutipedia\_app

build: .

volumes:

- ./static:/app/staticfiles

ports:

- "8000:8000"

environment:

- DEBUG=False

#build docker-compose.yaml

docker-compose up -d --build

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||