

# Docker/Docker compose

## Assignment #2




---

You are given a project whose infrastructure consists of two backends and one database. Both backends read the same record and return it when called on their port.

### Objective

- 1- Write Dockerfiles for each container including the database. (each container will be discussed in detail later)
- 2- Write a docker-compose.yml file orchestrating the containers

### Containers

-  Flask backend container
-  Go backend container
-  MySQL database container

## Project Lifecycle

---

### ▼ Go Backend Server

First thing you need to know about go before getting your hands dirty with dockerfiles is Go is not an interpreted language. Meaning it's not like node.js or python as explained before, which means its source code needs to be built and the server should run the output files.

So that you can have a peaceful time without losing your minds in tutorial hells

each Go directory consists of:

- 1- server.go | this is the main file of our source code which would be built to output files
- 2- go.mod | this is where you store your dependencies list

To build your application from the source code you can use this command

```
go build -o <output-file-name> <context-of-source-files>
```

To run the output file use this command

```
./<output-file-name>
```

just a simple reminder: we have an image which is almost plain called "scratch" on dockerhub that can fit some needs.

## ▼ **Flask Backend Server**

Flask is a python backend framework. If you remember what we said about python before, we said it's an interpreted language where you run it pointing to the source code without the need to build or do anything.

There's nothin more that needs to be clarified here for now. Just focus on the image versions you are using, some lightweight images can be not sufficient system dependencies. Using bigger images can be a better approach, you can google all the info you need about images.

## ▼ **MySQL Database Server**

Here in our application we have an initializer query that needs to be executed once the server is up and running. you can find it in init.sql, you can use some feature in docker-entrpoints and if that doesn't work you can use the bash script [init.sh](#) as an initializer to the database. Other than that everything is crystal clear.

## ▼ **Docker Compose**

Here in this docker compose file you need to keep track of the dependencies to know which service depends on which and to specify that in the compose file, also you can use either .env file or inject the env variables directly to the compose. Just don't forget the volume for the database, and env variables for configurations also the ports. Some containers might need a restart: always property so keep that in mind.

Okay now you're ready to go. Submission will be a zip file to the whole project of course don't forget to delete any built files in the process. Have fun!