Big Data, organisation and analysis

Building a multi container app using REST API

Steffen M. Noe, Spring 2024

What components we will use?

- We need a web server (REST API) ...
- ... and a database
- The REST API will be programmed in GO because it has very performant networking libraries
- The database should be relational and programmable with SQL (structured query language) and we choose Postgresql.

Which language to use?

- We want to use Go as programming language
- Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.
- It was designed in 2007 at Google to improve programming productivity in the era of multicore and network connected machines and large codebases.
- The language was formed by Robert Griesemer, Rob Pike and Ken Thompson (creator of UNIX and C)
- The language is:
  - statically typed and runtime efficient like C
  - well readable and usable (like Python)
  - allows high-performance networking and multiprocessing

Development inside od the container

- We will map the docker internal file system to out local file system

our “current” working directory on the local computer

These directories will be tied together, all changes in the container are visible on the local machine and vice-versa!

File structure in the container:

```
/-+-root/
   .
   |
   +-usr/src/app/-+-cmd/
   |              +-database/
   |              +-handlers/
   |              +-models/
   .
   +-lib/…
```

Create the minimal setup for our project

- make a directory called server3
- open VSCode and choose this directory as starting point
- open the terminal in VSCode
- We will create the Dockerfile and docker-compose.yml
- After entering the code (shown life in lesson) use the command:

```
docker compose up
```

Let’s enter the container via commandline

- to facilitate changes in the container use the command:

```
docker compose run --service-ports web bash
```

Let’s check if we can run commands inside the container

- run the command:

```
go version
```

- it should return your go version (1.22.2)
- We now know, the container can be used to load needed libraries to run our project

congrats, you managed the first milestone of the project

Installing dependencies

Preparing the module loading procedure

- we need to follow the Go procedure and first initialise the modules
- We also give the project a name that matches where it can be downloded, I use my Github for this
- enter to the container as before:

```
docker compose run --service-ports web bash
```

- and run the module initialisation:

```
go mod init github.com/stenoe/server3
```

replace with your github account name!

Installing dependencies

Installing the Fiber framework

- Fiber is a framework to build REST APIs in Go
- we will use the current stable version 2

```
go get github.com/gofiber/fiber/v2
```

Make the “Hello World” program

based on Go and Fiber

- we linked in the first step one directory on our computer to /usr/src/app inside the container!
- using the mkdir command we can create adirectory called cmd inside the container and it will be visible in VSCode that is running on our computer

```
mkdir cmd
```

- using the touch command we can create empty files

```
touch cmd/main.go
```

Run and test the “Hello World” program inside the container

based on Go and Fiber

- after we have been using VSCode to edit main.go and eventually had run go mod tidy to resolve dependencies we can start the REST API webserver inside the container

```
go run cmd/main.go -b 0.0.0.0
```

- ok, what’s happening here?

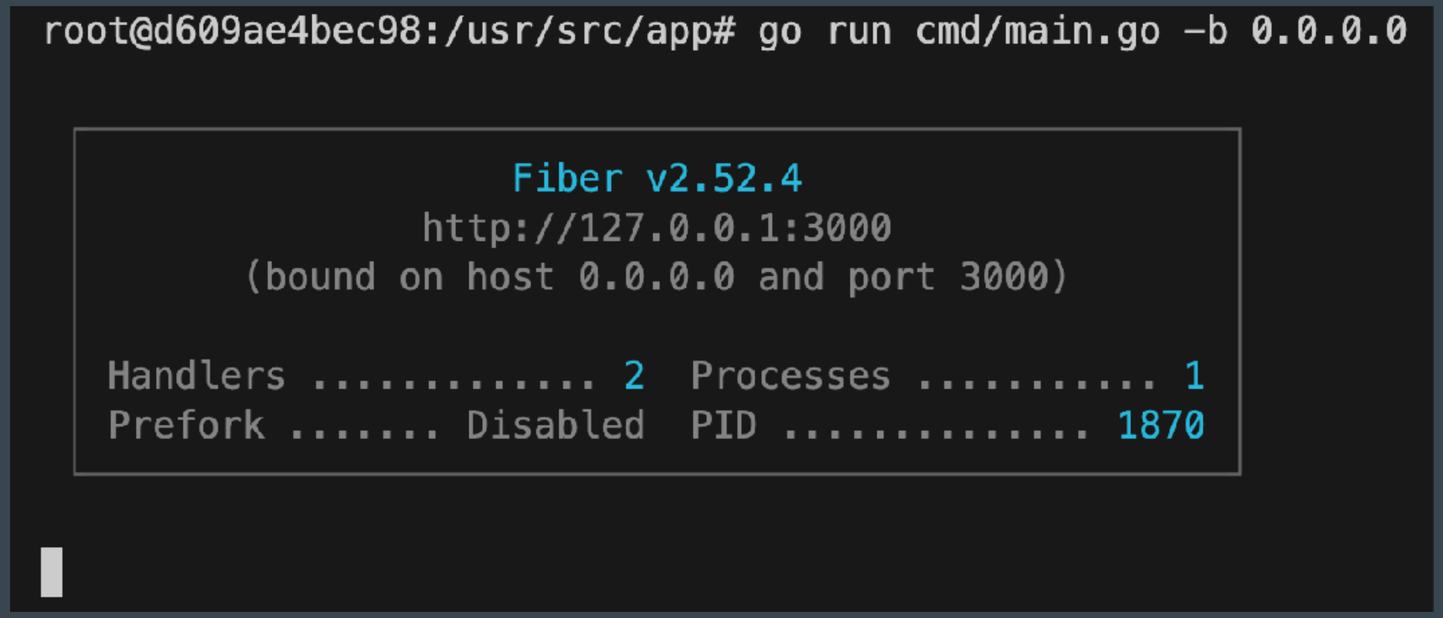
we are in the server3 directory!

inside the container,
this is our “dummy” IP address!

Run and test the “Hello World” program inside the container

based on Go and Fiber

- if everything works, you can see this output



- now we can test our server on http://localhost:3000

Starting the app from the host machine

changing the configuration files

- at this moment, we need always to enter the container to start the REST API
- we need to change our Dockerfile and the docker-compose.yml file such that we can run the app from our own computer (host)
- first, we add a COPY command to ensure all files are always updated between the container and the host
- second, we add a RUN command to update libraries
- third, we need to add the starting command to the .yml file

Starting the app from the host machine

- we now can run the command:

```
docker compose up
```

- and get

```
(env) (base) steffen@Aki server3 % docker compose up
[+] Running 1/0
 ✔ Container server3-web-1  Recreated
Attaching to web-1
web-1  | go: downloading github.com/gofiber/fiber/v2 v2.52.4
web-1  | go: downloading github.com/google/uuid v1.5.0
web-1  | go: downloading github.com/mattn/go-colorable v0.1.13
web-1  | go: downloading github.com/mattn/go-isatty v0.0.20
web-1  | go: downloading github.com/mattn/go-runewidth v0.0.15
web-1  | go: downloading github.com/valyala/bytebufferpool v1.0.0
web-1  | go: downloading github.com/valyala/fasthttp v1.51.0
web-1  | go: downloading golang.org/x/sys v0.15.0
web-1  | go: downloading github.com/rivo/uniseg v0.2.0
web-1  | go: downloading github.com/valyala/tcplisten v1.0.0
web-1  | go: downloading github.com/andybalholm/brotli v1.0.5
web-1  | go: downloading github.com/klauspost/compress v1.17.0
web-1  |
web-1  |
web-1  |                    Fiber v2.52.4
web-1  |                http://127.0.0.1:3000
web-1  |       (bound on host 0.0.0.0 and port 3000)
web-1  |
web-1  |  Handlers ............. 2  Processes ........... 1
web-1  |  Prefork ....... Disabled  PID ............... 1793
web-1  |
web-1  |
```

Starting the app from the host machine

adding “hot reloading”

- at this point, any change to the app source need a full rebuild of the app.
- we can use the Air package by Cosmtrek to enable automatic build/restart of our app
- first, we have to add another line to our Dockerfile to install the package
- second, create an emtpy configuration file for the air package

```
touch .air.toml
```

- third, copy the example file from github.com/cosmtrek/air/blob/master/air_example.toml and edit the line with cmd to match our cmd directory
- fourth, change the start command to use air instead of go

Starting the app from the host machine

rebuilding, restarting and testing

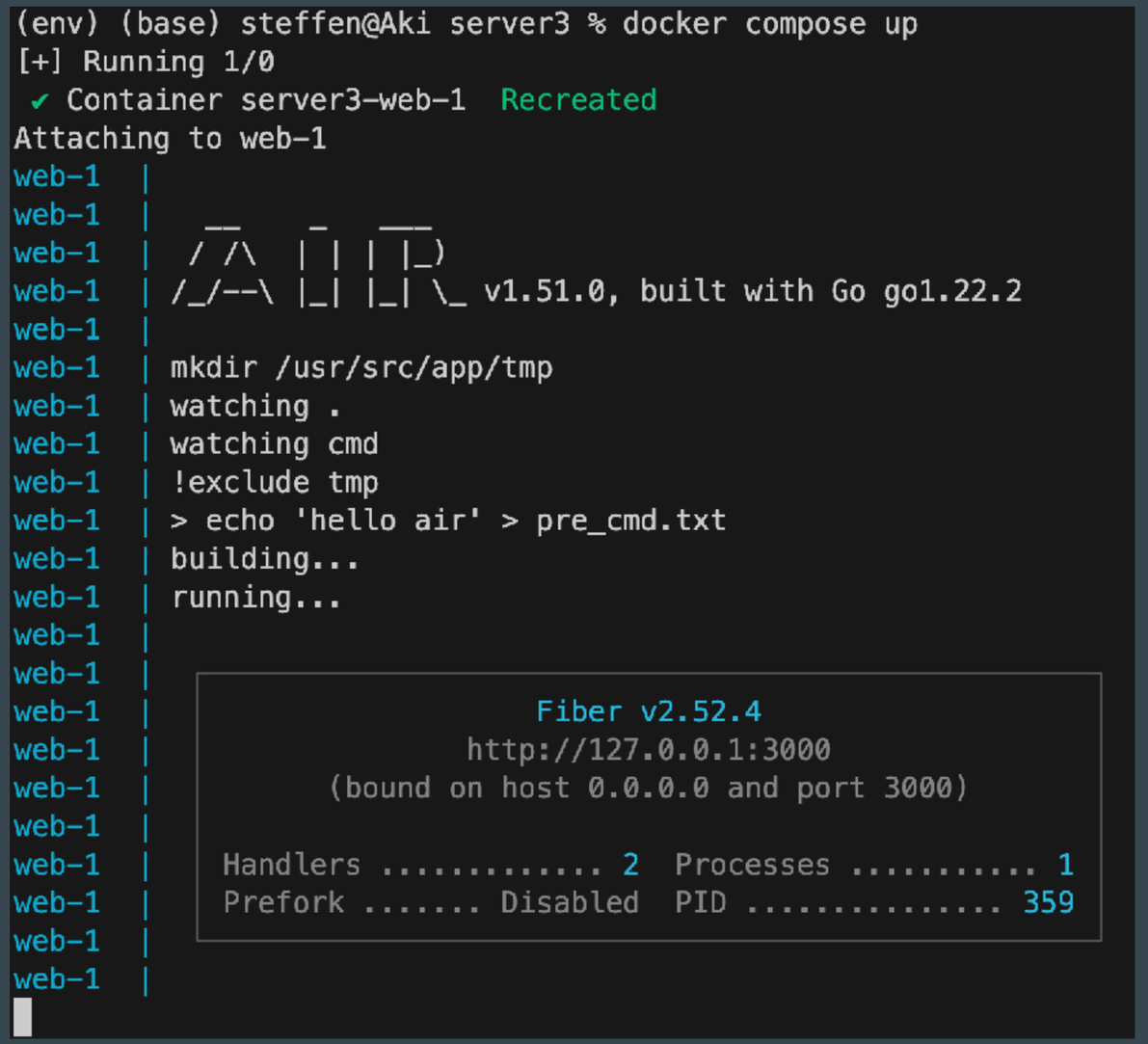
- now we can rebuild the app

```
docker compose build
```

- and restart it

```
docker compose up
```

- and we get
- we can now see changes immediately



congrats, you managed the second milestone of the project