

# Use Case for Docker - To get a refresher

Docker allows you to "containerize" a service/installation and deploy them without the complexity of setup

There are many pre-configured containers on the docker hub

**An image is file you create with all of the resources of your application**

**A Container is a **running** instance of the image**

If you pull an image such as a database, you have to create a network to attach it to  
Run this command to create your network  
`docker network create mongo-network`

To view your network(s)  
`docker network ls`

**`docker network rm my-network`**

**To view your images**

Docker images

**To view logs for any image (to see what the status and what it is doing)**

`docker logs < container ID-guid of the running container >`

**For the last set of logs**

`docker logs <guid of the running container> | tail` (tail only works on UNIX)

**To view running process/instances**

`docker ps`

**To run an image**

`docker run <image>`

To see a list of containers

`docker container ls`

**To stop a container**

`docker stop <container id>`

**To delete a container**

`docker rm <container id>`

**To delete an image**

`docker rmi <image id>`

To build your own image (At the end of the day, this is what you want to do below)

First Create a file in project, the file must be called

Dockerfile

**Below is the contents of a file in a sample Golang web app**

FROM golang:1.17

WORKDIR /go/src/app

COPY . .

RUN mkdir /cmd

RUN go build -o /cmd/app /go/src/app/main.go

ENTRYPOINT ["/cmd/app"]

Save the file in your project directory

Issue the and

docker build -t my-app:1.0 .

The :1.0 is the version

The . means current directory

After you run the command, you can go see the app in docker repo by typing at the command prompt

docker images

**To run the image**

docker run my-app:1.0

To run the image against a port (to be able to browse)

docker run -p 3000:3000 my-app:1.0

## Use Case (building my Rock Paper Scissors app)

docker build -t mygoapprockpaper:1.0 .

After I built the image

Docker images

lioneljones@lionels-MacBook-Air ~ % **docker images**

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mygoapprockpaper	1.0	7aaa03f6baa5	42 seconds ago	813MB

lioneljones@lionels-MacBook-Air ~ %

Now that I have an image, I am going to **containerize** it

**docker run -p 8085:8085 mygoapprockpaper:1.0**

lioneljones@lionels-MacBook-Air ~ % **docker run -p 8085:8085 mygoapprockpaper:1.0**

2022/03/23 01:43:42 Starting web server on port 8085

I ran the commands below to view statuses

Last login: Tue Mar 22 20:04:23 on ttys003

lioneljones@lionels-MacBook-Air ~ % **docker images**

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mygoapprockpaper	1.0	7aaa03f6baa5	9 minutes ago	813MB

lioneljones@lionels-MacBook-Air ~ % **docker ps**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
0e3775d495d5	mygoapprockpaper:1.0	"/cmd/app"	About a minute ago	Up	About a minute
0.0.0.0:8085->8085/tcp	intelligent_ride				

lioneljones@lionels-MacBook-Air ~ % **docker container ls**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
0e3775d495d5	mygoapprockpaper:1.0	"/cmd/app"	About a minute ago	Up	About a minute
0.0.0.0:8085->8085/tcp	intelligent_ride				

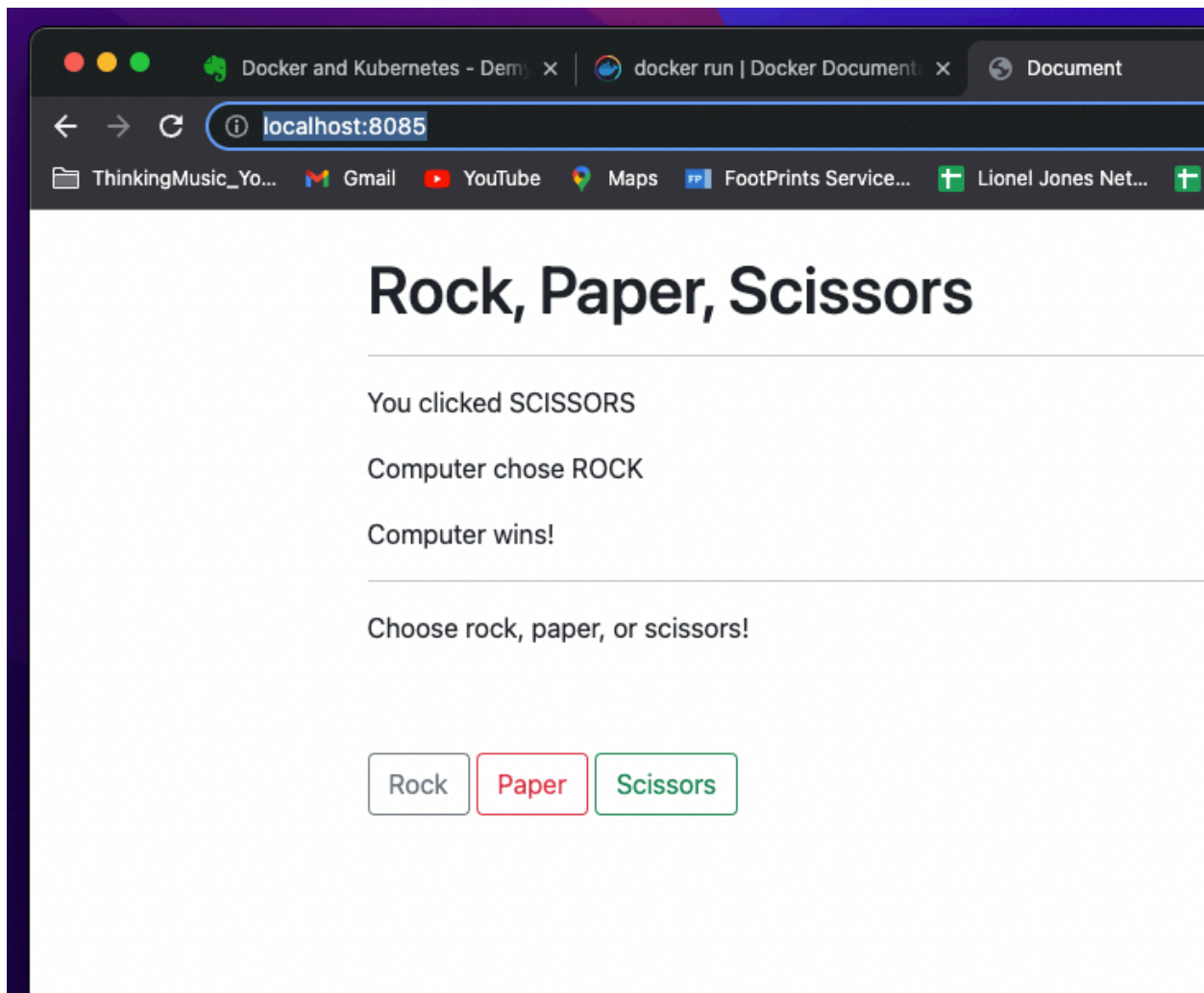
lioneljones@lionels-MacBook-Air ~ %

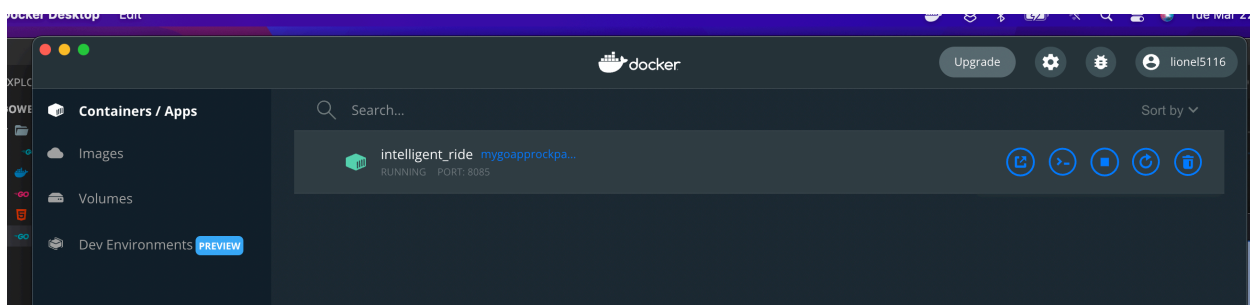
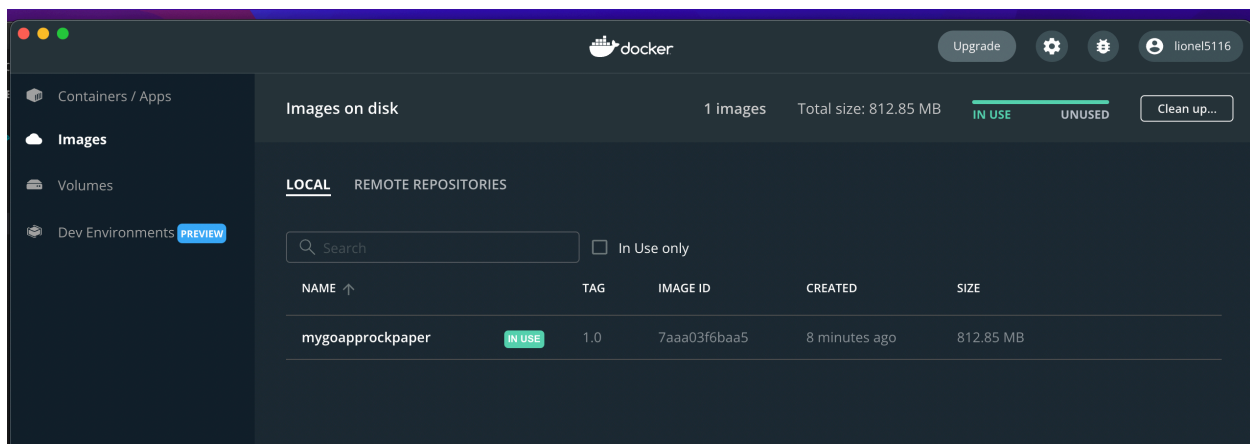
As you can see above, after I did a docker run ..., it created a container for my image to run on and the port  
Was forwarded to 8085

I enter the URL

<http://localhost:8085/>

**And the site comes up**



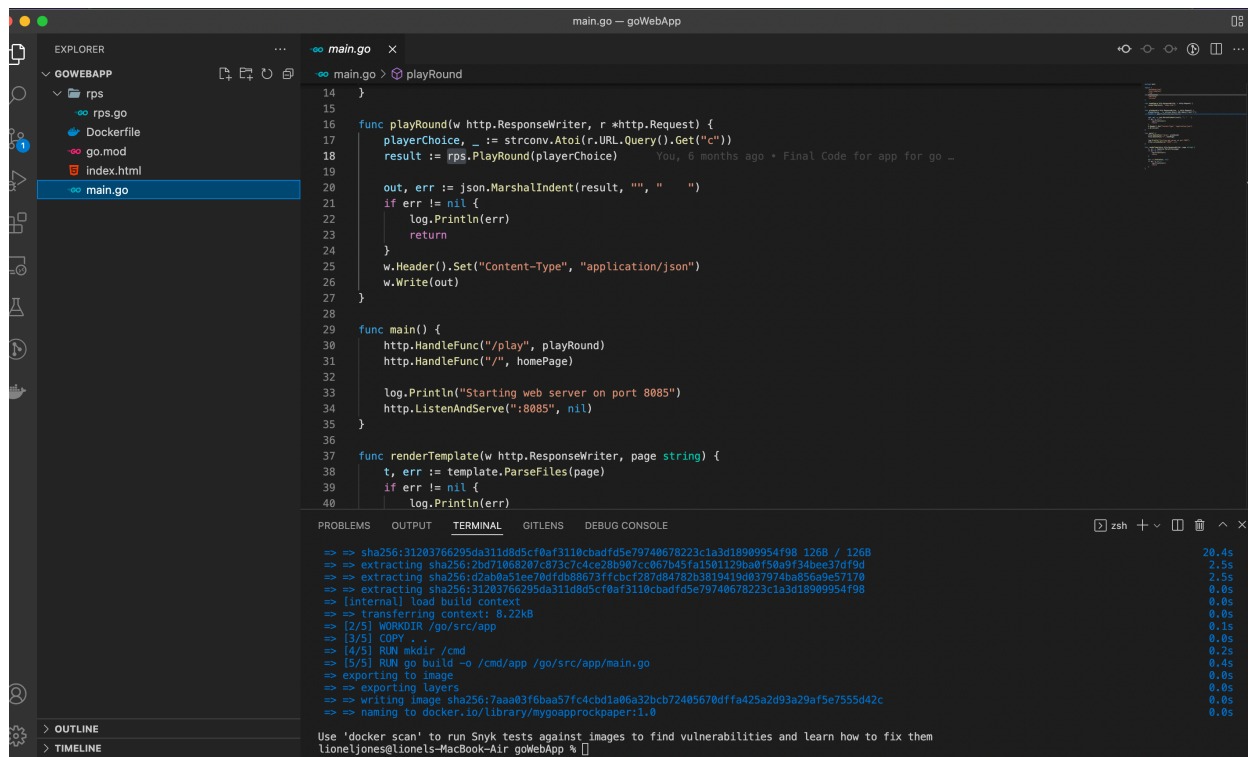


You can view the GUI above to see the statuses of your image/container

### So in a nutshell:

I created a sample Go Project (From Udemy Course)

I created a docker file



I run docker command to create an image  
I ran a command to run the image in a container  
I was able to browse the application through the docker image  
Project Name:  
GoWebApp

## Explanation of the Docker File

The explanation of the file below:

The WORKDIR:

This is the directory that your build files will be copied to after your project is built

The COPY ..

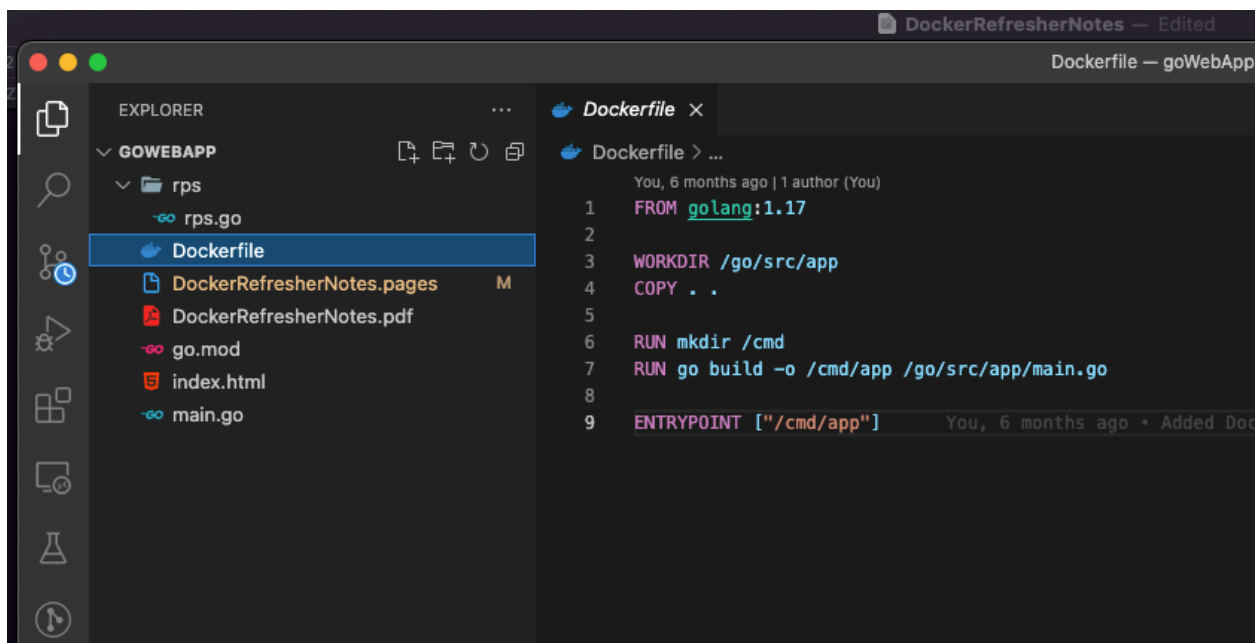
This means copy all of the build files to the working directory

The RUN

This runs a command - mkdir /cmd

RUN go ..

This issues the go run command to build the project, -o output the build files to the working directory you specified earlier in the file, and the build file is main.go

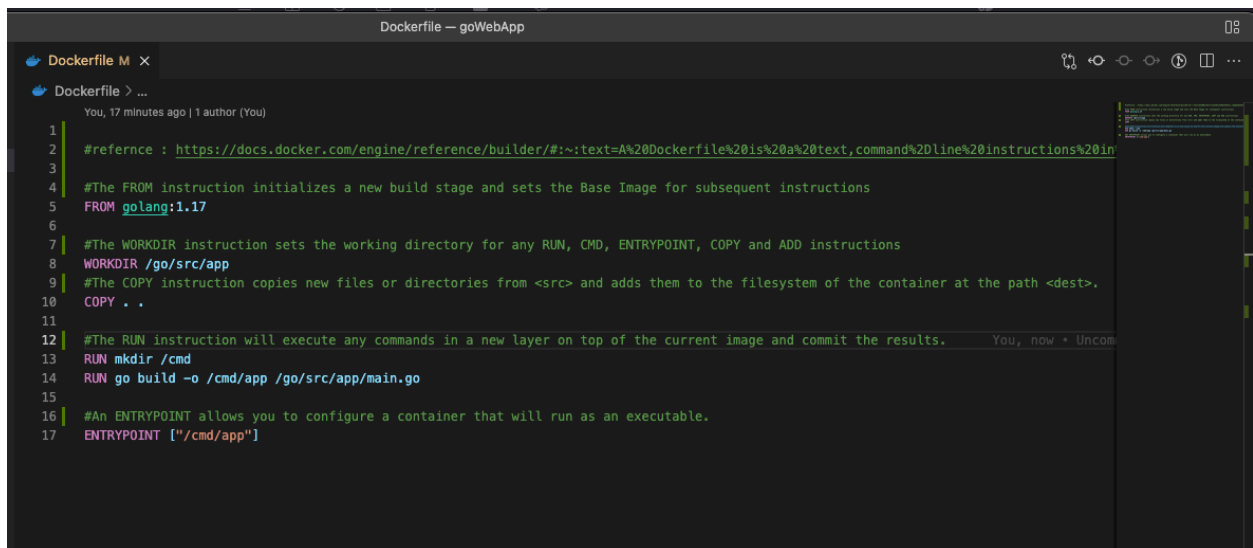


<https://docs.docker.com/engine/reference/builder/>

#:~:text=A%20Dockerfile%20is%20a%20text,command%2Dline%20instructions%20in%20su

ccession.

From the Docs (the link above)

A screenshot of a code editor window titled "Dockerfile — goWebApp". The editor shows a Dockerfile with 17 lines of code. The code defines a build stage for a Go web application. It starts with a reference to the Docker documentation, then sets the base image to "golang:1.17". It sets the working directory to "/go/src/app" and copies the current directory's contents into the container. Then, it creates a new layer with "mkdir /cmd" and builds the application with "go build -o /cmd/app /go/src/app/main.go". Finally, it sets the entrypoint to run the application as an executable at "/cmd/app".

```
1 #reference : https://docs.docker.com/engine/reference/builder/#:~:text=A%20Dockerfile%20is%20a%20text,command%20line%20instructions%20in
2
3
4 #The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions
5 FROM golang:1.17
6
7 #The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions
8 WORKDIR /go/src/app
9 #The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
10 COPY . .
11
12 #The RUN instruction will execute any commands in a new layer on top of the current image and commit the results.
13 RUN mkdir /cmd
14 RUN go build -o /cmd/app /go/src/app/main.go
15
16 #An ENTRYPOINT allows you to configure a container that will run as an executable.
17 ENTRYPOINT ["/cmd/app"]
```

## To see inside of a running container:

Once you started your image, follow the commands below

### To see your running docker processes

lioneljones@lionels-MacBook-Air goWebApp % **docker ps**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3d078a802ca9	7aaa03f6baa5	"/cmd/app"	49 seconds ago	Up 48 seconds	0.0.0.0:8090->8085/tcp	crazy_engelbart

### Execute the command to see inside of your container

lioneljones@lionels-MacBook-Air goWebApp % **docker exec -t -i 3d078a802ca9 /bin/bash**

### You will then be inside of your container

root@3d078a802ca9:/go/src/app# **ls**

Dockerfile go.mod index.html main.go rps

root@3d078a802ca9:/go/src/app# **ls -al**

```
total 44
drwxr-xr-x 1 root root 4096 Mar 23 01:35 .
drwxrwxrwx 1 root root 4096 Mar 23 01:35 ..
-rw-r--r-- 1 root root 6148 Mar 22 00:08 .DS_Store
drwxr-xr-x 7 root root 4096 Oct  2 20:21 .git
-rw-r--r-- 1 root root 132 Oct  2 20:13 Dockerfile
-rw-r--r-- 1 root root  22 Oct  2 20:13 go.mod
-rw-r--r-- 1 root root 2039 Oct  2 20:13 index.html
-rw-r--r-- 1 root root  908 Oct  2 20:13 main.go
drwxr-xr-x 2 root root 4096 Oct  2 20:21 rps
```

### Notice how the directory structure is the same as the structure outlined in your docker file explained above

root@3d078a802ca9:/go/src/app# **pwd**

/go/src/app

root@3d078a802ca9:/go/src/app#

After you have stopped your container and want to remove it  
To see a list of instances (even if not running) of your containers:

```
lioneljones@lionels-MacBook-Air goWebApp % docker ps -a
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS
NAMES
3d078a802ca9   7aaa03f6baa5  "/cmd/app"     11 minutes ago Exited (2)    About a minute ago
crazy_engelbart
```

### **Then to remove the container**

```
lioneljones@lionels-MacBook-Air goWebApp % docker rm 3d078a802ca9
3d078a802ca9
```

```
lioneljones@lionels-MacBook-Air goWebApp % docker ps -a
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS    NAMES
lioneljones@lionels-MacBook-Air goWebApp %
```