

Getting GoLangAPI to work with MYSQL

Good reference(s)

Creating a RESTful API

Below is a link to the docs that explains how to write an API with Go

<https://go.dev/doc/tutorial/web-service-gin>

The GitHub Project is located: (I coded it in the Udemmy Coarse)

<https://github.com/lionel5116/golangApiExample.git>

```
//you have to name your fields "Uppercase" so that they will be
exportable *
//add the json:"<tags>" in lowercase
type Product struct {
    Product_id    int    `json:"product_id"`
    Name          string `json:"name"`
    Quantity_in_stock int    `json:"quantity_in_stock"`
    Unit_price    float32 `json:"unit_price"`
}
```

One of the key things to note is that when you declare your structure, you have to have your field names in UpperCase in order for the structure to be exportable when you write it out to your response.

In order to append objects to an array:

```
91
92 //the syntax below is how you append objects to a struct
93 func addProduct(_product Product) {
94     var my_product = new(Product)
95     my_product.Product_id = _product.Product_id
96     my_product.Name = _product.Name
97     my_product.Quantity_in_stock = _product.Quantity_in_stock
98     my_product.Unit_price = _product.Unit_price
99     arrProducts = append(arrProducts, *my_product)
100 }
101
```

The code for calling MYSQL

```
func getProducts(c *gin.Context) {  
    //declare connection  
    db, err := sql.Open("mysql", "root:Mag17615@tcp(127.0.0.1:3306)/sql_store")  
    if err != nil {  
        panic(err.Error())  
    }  
    defer db.Close()  
  
    //query the database  
    results, err := db.Query("SELECT product_id,name,quantity_in_stock,unit_price FROM products")  
    if err != nil {  
        panic(err.Error())  
    }  
  
    var _product Product  
    //loop through the resultset  
    for results.Next() {  
        var product Product  
        err = results.Scan(&product.Product_id, &product.Name, &product.Quantity_in_stock, &product.Unit_price)  
  
        _product.Product_id = product.Product_id  
        _product.Name = product.Name  
        _product.Quantity_in_stock = product.Quantity_in_stock  
        _product.Unit_price = product.Unit_price  
        addProduct(_product) //append each product object to the array  
  
        if err != nil {  
            panic(err.Error())  
        }  
    }  
    c.IndentedJSON(http.StatusOK, arrProducts)  
}
```

Your imports for MYSQL

When you browse:

The screenshot shows a web browser window with the address bar displaying `localhost:8090/products/`. The page content is a JSON array of eight product objects. Each object contains the following fields: `product_id`, `name`, `quantity_in_stock`, and `unit_price`. The products listed are:

- product_id: 1, name: "Foam Dinner Plate", quantity_in_stock: 70, unit_price: 1.21
- product_id: 2, name: "Pork - Bacon,back Peameal", quantity_in_stock: 49, unit_price: 4.65
- product_id: 3, name: "Lettuce - Romaine, Heart", quantity_in_stock: 38, unit_price: 3.35
- product_id: 4, name: "Brocolinni - Gaylan, Chinese", quantity_in_stock: 90, unit_price: 4.53
- product_id: 5, name: "Sauce - Ranch Dressing", quantity_in_stock: 94, unit_price: 1.63
- product_id: 6, name: "Petit Baguette", quantity_in_stock: 14, unit_price: 2.39
- product_id: 7, name: "Sweet Pea Sprouts", quantity_in_stock: 98, unit_price: 3.29
- product_id: 8, name: "Island Oasis - Raspberry", quantity_in_stock: 98, unit_price: 3.29

The browser's bookmark bar includes links to "ThinkingMusic_Yo...", "Gmail", "YouTube", "Maps", "FootPrints Service...", and "Lionel Jones Net...". A notification for "FootPrints Service Core - IT Service" is visible in the top right corner.

Overview of Go

Every Go program is made up of packages

In your main file (i.e `main.go`), you import package `main`

Dependencies

When you start a new go project, you want to add a module to it by typing:

go mod init *myapp*

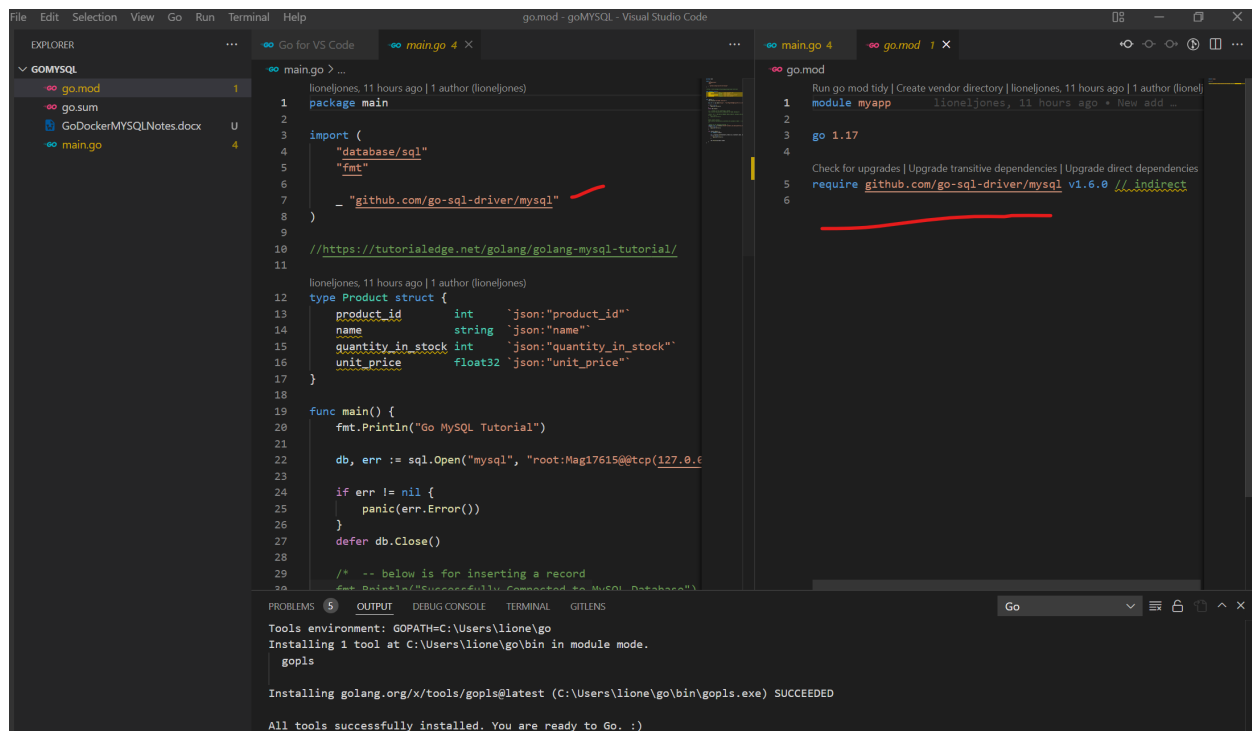
What this does is:

The go mod init command **creates a go.mod file to track your code's dependencies**

This is like a node_modules file (it is called **go.mod**)

When you add imports to your program, if go does not find the correct dependency in your project, it will download it and add the reference to the go.mod file

For example:



In my MySQL go program, I added an import reference to mysql, when I tried to run the program, go did not recognize mysql. It prompted me to install it, once I installed it, it placed the dependency in the go.mod file.

The import keyword

import (

```
    "fmt"
    "math/rand"
```

)

This is the same as any other import statement in other languages. It is where you add the packages that you want to use in your program

Data Types:

https://www.w3schools.com/go/go_data_types.php

Go has three basic data types:

bool: represents a boolean value and is either true or false

Numeric: represents integer types, floating point values, and complex types

string: represents a string value

Integers

https://www.w3schools.com/go/go_integer_data_type.php

int,int8,int16,int32,int64

Floats

https://www.w3schools.com/go/go_float_data_type.php

float32,float64

Strings

https://www.w3schools.com/go/go_string_data_type.php

Variables

https://www.w3schools.com/go/go_variables.php

In Go, there are two ways to declare a variable

```
var variablename type = value
```

or

```
variablename := value (this method uses type inference)
```

Example

```
package main
import ("fmt")

func main() {
    var student1 string = "John" //type is string
    var student2 = "Jane" //type is inferred
    x := 2 //type is inferred

    fmt.Println(student1)
    fmt.Println(student2)
    fmt.Println(x)
}
```

Try it Yourself »

Functions

functions are declared as below:

The way functions work is a little different in go:

```
functions.go
1 package main
2
3 import "fmt"
4
5 func add(x int, y int) int {
6     return x + y
7 }
8
9 func main() {
10     fmt.Println(add(42, 13))
11 }
12
```

Notice how the type comes after the variable name, also the return type comes after the method signature

Loops

https://www.w3schools.com/go/go_loops.php

The **for** loop is **the only** loop available in Go.

```

package main
import ("fmt")

func main() {
    for i:=0; i < 5; i++ {
        fmt.Println(i)
    }
}

```

Classes

Go **does not** have Classes, but it has **Structures**

https://www.w3schools.com/go/go_struct.php

Syntax:

```

type Person struct {
    name string
    age int
    job string
    salary int
}

```

Structures are declared with the **type** keyword

When structs are used, we don't declare them with a "new" keyword or with a (). We use the var keyword. And remember since the structure is a type, we note the type after the variable name.

Example:

```

type Person struct {
    name string
    age int
    job string
    salary int
}

```

```

func main() {
    var pers1 Person

    // Pers1 specification
    pers1.name = "Hege"
}

```

```
pers1.age = 45
pers1.job = "Teacher"
pers1.salary = 6000

// Access and print Pers1 info
fmt.Println("Name: ", pers1.name)
fmt.Println("Age: ", pers1.age)
fmt.Println("Job: ", pers1.job)
fmt.Println("Salary: ", pers1.salary)

}
```

go Build

See the command list below

This builds an executable of your program. You can distribute the program and the target computer “**does not**” have to have Go installed to run the go program

Quick List

Tip: Go Commands

Command	Description
<code>go run <u>file.go</u></code>	runs your go program
<code>go mod init <u>myapp</u></code>	creates a module for you to use your own custom modules
<code>var <u><variablename></u> string</code>	creating variables
<code><u><variablename>:= ""</u></code>	creating variables using type inference
<code>go build <u>main.go</u></code> or mac <code>go build -o <u>eliza</u> <u>main.go</u></code> windows <code>go build -o <u>eliza.exe</u> <u>main.go</u></code>	builds your program To build with a different name
<code>.\main</code>	to run your program
<code>go mod init <u>myapp</u></code>	command for Creating a custom mod file for your application's modules

Adding and calling your own package

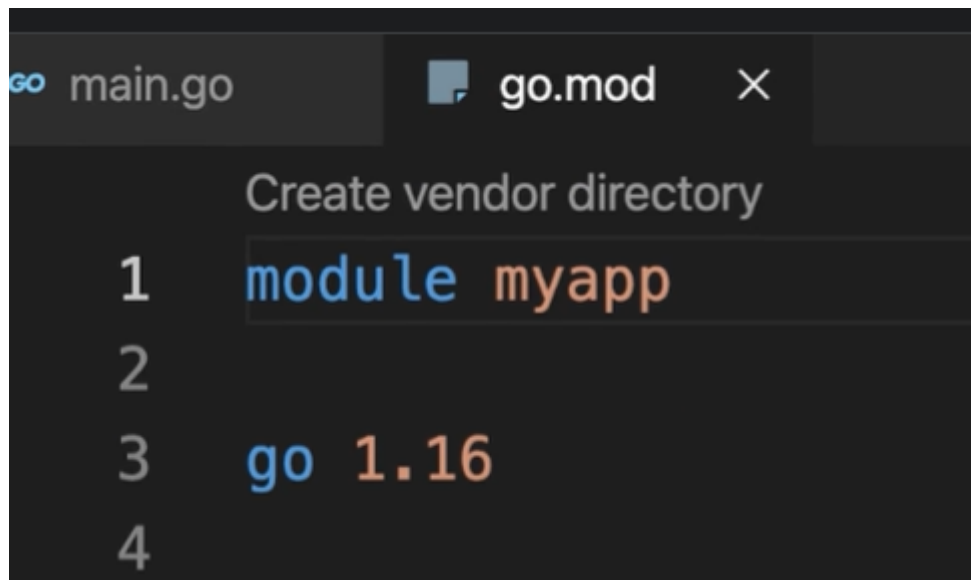
<https://www.udemy.com/course/go-programming-language-crash-course/learn/lecture/26161716#overview>

To allow for this, you have to create the mod file as discussed earlier

At the command line, type:

`go mod init myapp`

Once you do that, it creates the module file:

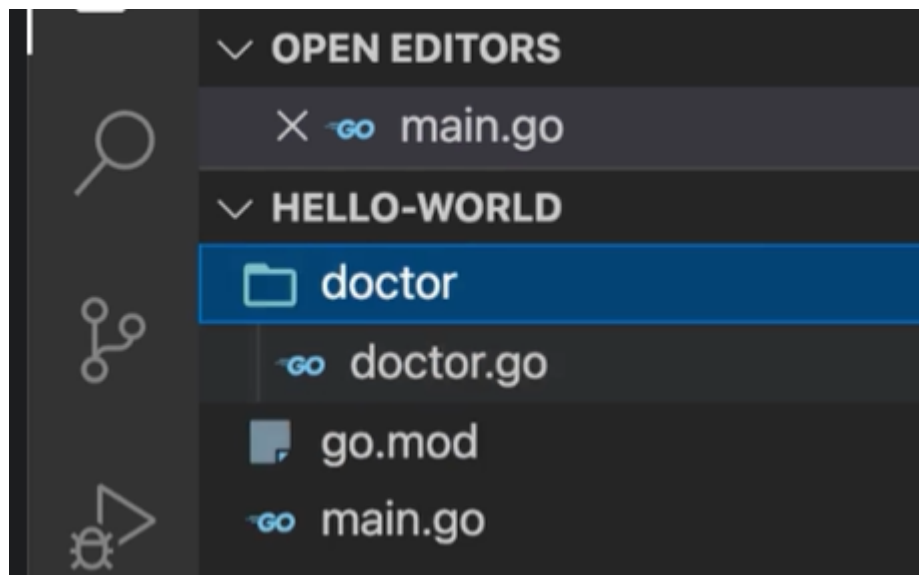


The screenshot shows a code editor with two tabs at the top: 'main.go' and 'go.mod'. The 'go.mod' tab is active, displaying the following content:

```
Create vendor directory
1 module myapp
2
3 go 1.16
4
```

Notice the module is named “myApp” based on the name you used at the command line.

Next create a directory and create .go file in it:



Paste some code in the file

In your main program, when you want to use this module, you make a reference to the module in your import statement (closure)

```

1  package main
2
3  import (
4      "fmt"
5      "myapp/doctor"
6  )
7
8  func main() {
9      var whatToSay string
10
11     whatToSay = doctor.Intro()
12 }
13

```

{LP} Learn Programming academy

This allows you to reference a method in the package you created. (Notice how we use the “myapp” keyword along with reference to the package)

```

// Intro returns the intro text
func Intro() string {
    return `
I'm Eliza
-----

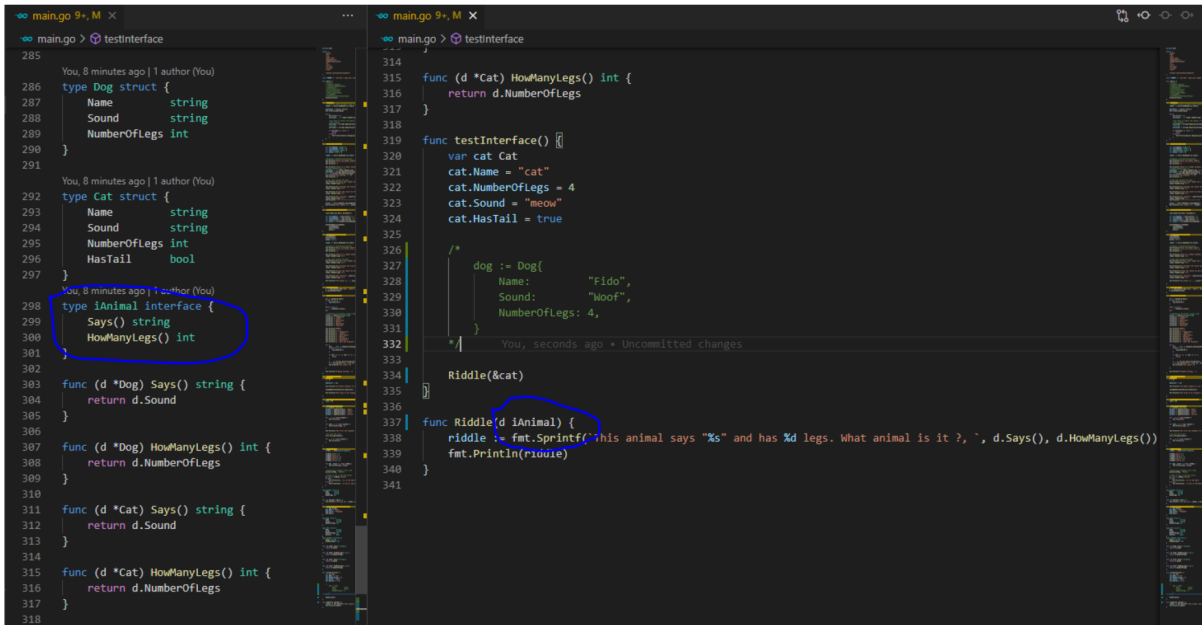
Talk to the program by typing in plain English, using normal
and lower-case letters and punctuation. Enter 'quit' when done.

Hello. How are you feeling today?`
}

```

Tip: Interfaces

As you see below, interfaces "can" have methods as opposed to structures. Also you will note below that this demonstrates a good use case to use a pointer in [golang](https://golang.org/). Interfaces use the "attach" a function to a structure methodology. You don't use the keyword "implement" or: like in inheritance. You just meet the requirements of creating the functions the interface requires.



The image shows two side-by-side windows of a Go IDE. The left window displays the definition of a `Dog` struct, a `Cat` struct, and an `IAAnimal` interface. The `IAAnimal` interface is circled in blue and contains two methods: `Says() string` and `HowManyLegs() int`. Below the interface, there are two functions: `Says()` for `Dog` and `Cat`, and `HowManyLegs()` for `Dog` and `Cat`. The right window shows the implementation of these functions. It defines a `Cat` struct with fields `Name`, `NumberOfLegs`, `Sound`, and `HasTail`. It then implements the `IAAnimal` interface methods for `Cat`. A `Riddle` function is also shown, which takes an `IAAnimal` pointer and prints a riddle based on the animal's `Says()` and `HowManyLegs()` methods. The `Riddle` function is also circled in blue.

The _ in Golang

What is Blank Identifier(underscore) in Golang?

_ (underscore) in Golang is known as the Blank Identifier.

Golang has a special feature to define and use the unused variable using Blank Identifier.

The real use of Blank Identifier comes when a function returns multiple values

```
func main ()
{
    reader:= bufio.NewReader(os.Stdin)
    whatToSay := doctor.Intro()
    fmt.Println(whatToSay)

    userInput, _ := reader.ReadString()

}
```

Above

The `reader.ReadString()` function returns "two" return values. But we "don't" have to use the value for the `_`, the underscore is just a "placeholder"

Functions that return two values

```
reg, err := regexp.Compile("[^a-zA-Z0-9"])
```

```
if err != nil {
```

```
log.Fatal(err)
}
userInput = reg.ReplaceAllString(userInut,"")
```

Above is typical, you will see error handling handled where you have the err as the 2nd return value

Stripping of the carriage return placeholder when reading a string from user input (from iO reader)

<https://www.udemy.com/course/go-programming-language-crash-course/learn/lecture/26161724#overview>

```
userInput = strings.Replace(userInput, "\r\n", "", -1)
```

DOCKER NOTES

<https://www.docker.com/>

You install Docker

<https://hub.docker.com/editions/community/docker-ce-desktop-windows>

See the Evernote note:

Docker and Kubernetes - Demystified 2021

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

A Container is a running instance of the image

Base steps to run a container

For an existing container, you run the following command to pull it from a docker hub (public or private)

docker pull <image> (i.e docker pull mongo)

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

To allow the container to run in your environment, you have run the following command to attach/run (you can get the pre-requisites from the docs online)

```
docker run -d -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=admin -e
MONGO_INITDB_ROOT_PASSWORD=password --name mongoddb --net mongo-network mongo
-d (run in detached mode - background)
```

-p (the port you assign - the left port-number is the port for your local machine, the right port is the port for the instance) - This called "Port-forwarding", You forward your local port to the container port

For every image, you run the docker run command and attach a port as shown above

```
docker run -d -p 8081:8081 -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin -e  
ME_CONFIG_MONGODB_ADMINPASSWORD=password --net mongo-network --name mongo-express -e  
ME_CONFIG_MONGODB_SERVER=mongodb mongo-express
```

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)

Docker Desktop

You can also see your running instances on the Docker Desktop app and also view logs. You can also stop/delete an instance as well

To view running process/instances

docker ps

Using the container in your code

After you have any running instance, you refer to it in your code just like any other installation (i.e mysql , postgres, mongo, etc)

Below, your running instance's network address is:

http://localhost:27017

When you map a port, as explained above, you map your own desired mapping on the left side in regards to what port you want to use. This is different from when you install mongo through an installation where your default port is 27017, you can map it as 27999 etc.. (http://localhost:27999)

```

6 app.post(['/update-profile', function (req, res) {
7     var userObj = req.body;
8     var response = res;
9
10    console.log('connecting to the db....')
11
12    MongoClient.connect('mongodb://admin:password@localhost:27017', function (err, client) {
13        if (err) throw err;
14
15        var db = client.db('user-account');
16        userObj['userid'] = 1;
17
18        var query = {userid: 1};
19        var newValues = {$set: userObj};
20
21        console.log('successfully connected to the user-account db')
22
23        db.collection('users').updateOne(query, newValues, {upsert: true}, function (err, res) {
24            if (err) throw err;
25            console.log('successfully updated or inserted')
26            client.close();
27            response.send(userObj);
28        });
29    });
30 });

```

Your own service (containerize)

You can create your own custom service/api and create a container for it, and store this container out on the public repository or private company repository

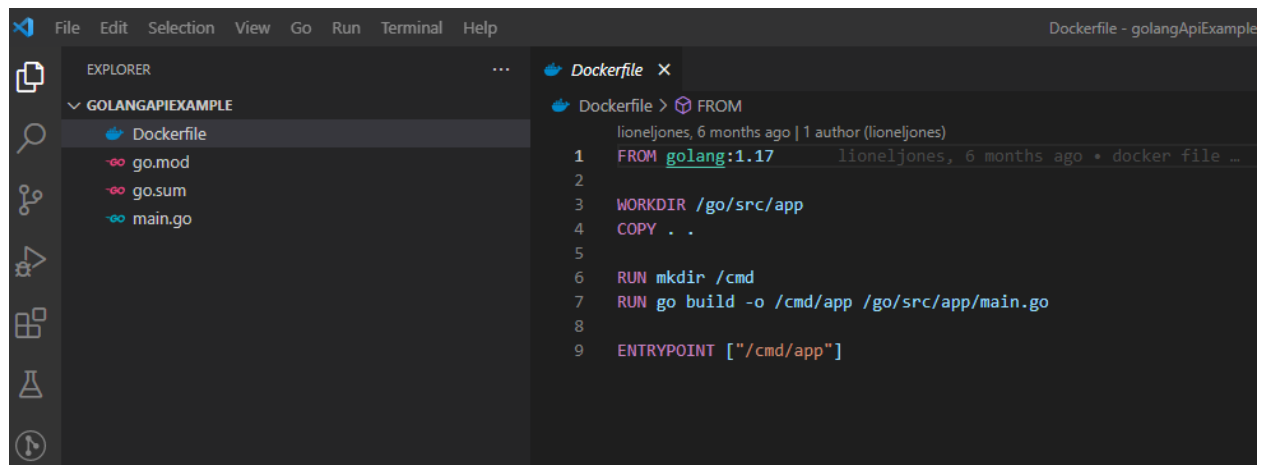
A good course:

Docker Tutorial for Beginners [FULL COURSE in 3 Hours]

<https://www.youtube.com/watch?v=3c-iBn73dDE>

If you look at the project:

git clone <https://github.com/lionel5116/golangApiExample.git>



It has a proper Docker File that dennis helped me with to create my own docker image

Docker Commands

::

To install/run an image	<code>docker run <image name>:version#</code> <code>docker run postgres:9.6</code>
To check the version of docker	<code>docker version</code>
To see running containers	<code>docker <u>ps</u></code>
docker images	Shows all of the images on your machine
docker run <image>	Creates a container to run the image and starts it
docker <u>ps</u>	Gives you status of all of the running containers
To stop a container	ctrl + c (if not in detached mode)
docker logs <container id> or docker logs <container name>	Check Logs
docker stop <container id>	stops a container
docker rm <container id>	deletes a container (needed sometimes to delete an image) - pre-step
docker <u>rmi</u> <image id>	deletes an image
To view a list of running containers docker container ls Command to clean up containers docker container ls -a	