

29 JANUARY 2018

My First Go Microservice using MongoDB and Docker Multi-Stage Builds

I started learning Go a few months back but I don't really get time to experiment since I am very busy with work. The only time I get to do some "real" learning is when I am on vacation leave. I believe the best way to learn is to try out some examples and implement simple use cases. Microservices are the buzzword lately so I thought of creating a simple microservice using the Go Language which connects to a MongoDB NoSQL database, compiled using Docker a multi-stage build, and deployed as a Docker container.

Here are the requirements:

1. The microservice needs to expose 2 HTTP endpoints
 - GET /jobs - this endpoint should return a list of jobs retrieved from a MongoDB database
 - POST /jobs - this endpoint should be able to accept a json string and save it to a MongoDB database
2. The microservice needs to be deployed as a Docker container
 - The docker image should be small

3. MongoDB should be running as a Docker container as well

Since it is important to be able to test the microservice with MongoDB, let's start with #3.

Running a MongoDB docker container is pretty straightforward. To make things simpler, let's use docker-compose. Here is the docker-compose.yml file we'll use.

```
version: '3'
services:
  mongodb:
    image: mongo
    ports:
      - "27017:27017"
    volumes:
      - "mongodata:/data/db"
    networks:
      - network1

volumes:
  mongodata:

networks:
  network1:
```

File can be copied from here:

<https://github.com/donvito/learn-go/blob/master/mongo-microservice/mongodb/docker-compose.yml>

Save this as docker-compose.yml. This compose file binds the host machine's(your laptop or VM) port to MongoDB's. If this port is used in the host machine, just change the port no. - "**27017**:27017". The left one is the host machine's port.

If you'd like to just run MongoDB as a container without bothering with docker-compose, you can do so as well. Just follow the steps in [MongoDB's dockerhub](#).

After saving the file, spin up the the MongoDB docker container using this command. Make sure you are in the same directory as the compose file.

```
$ docker-compose up
```

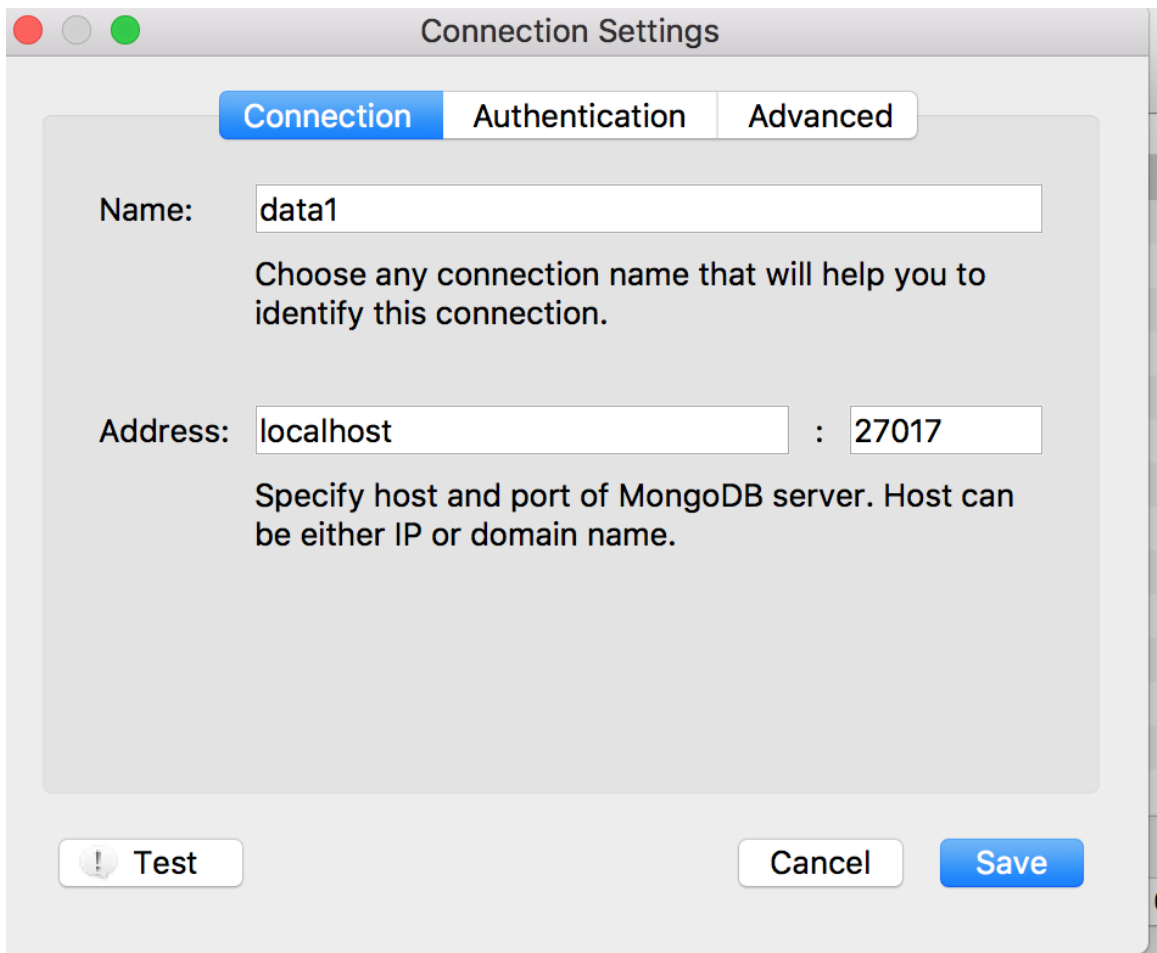
You'll see some similar output below. For now, let's not set the username and password. You can do that later.

```

networks:
Mon Jan 29 05:23 PM[melvin@Melvin-MacBook:docker-compose]$ docker-compose up
WARNING: Dependency conflict: an older version of the 'docker-py' package may be polluting the namespace. If you're experiencing crashes, run the following command to remedy the
pip uninstall docker-py; pip uninstall docker; pip install docker
Starting dockercompose_mongodb_1 ... done
Attaching to dockercompose_mongodb_1
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=855daed587f9
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] db version v3.6.2
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1t 3 May 2016
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] allocator: tcmalloc
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] modules: none
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] build environment:
mongodb_1 | 2018-01-29T09:28:55.007+0000 I CONTROL [initandlisten] distmod: debian81
mongodb_1 | 2018-01-29T09:28:55.008+0000 I CONTROL [initandlisten] distarch: x86_64
mongodb_1 | 2018-01-29T09:28:55.008+0000 I CONTROL [initandlisten] target_arch: x86_64
mongodb_1 | 2018-01-29T09:28:55.008+0000 I CONTROL [initandlisten] options: { net: { bindIpAll: true } }
mongodb_1 | 2018-01-29T09:28:55.009+0000 I CONTROL [initandlisten] Detected data files in /data/db created by the 'wiredTiger' storage engine, so setting the active storage engine
e to 'wiredTiger'.
mongodb_1 | 2018-01-29T09:28:55.009+0000 I STORAGE [initandlisten] WiredTiger message [1517218135:170196][1:0x7f7a79b01a00], txn-recover: Main recovery loop: starting at 2/55936
mongodb_1 | 2018-01-29T09:28:55.170+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
mongodb_1 | 2018-01-29T09:28:55.273+0000 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-filesystem
mongodb_1 | 2018-01-29T09:28:55.355+0000 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=487M,session_max=20000,eviction=(threads_min=4,threads_max=4),config-
base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress
),
mongodb_1 | 2018-01-29T09:28:55.433+0000 I CONTROL [initandlisten] WiredTiger message [1517218135:273411][1:0x7f7a79b01a00], txn-recover: Recovering log 2 through 3
mongodb_1 | 2018-01-29T09:28:55.433+0000 I STORAGE [initandlisten] WiredTiger message [1517218135:355601][1:0x7f7a79b01a00], txn-recover: Recovering log 3 through 3
mongodb_1 | 2018-01-29T09:28:55.433+0000 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
mongodb_1 | 2018-01-29T09:28:55.433+0000 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
mongodb_1 | 2018-01-29T09:28:55.433+0000 I CONTROL [initandlisten]
mongodb_1 | 2018-01-29T09:28:55.445+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
mongodb_1 | 2018-01-29T09:28:55.446+0000 I NETWORK [initandlisten] waiting for connections on port 27017
mongodb_1 | 2018-01-29T09:28:55.802+0000 I NETWORK [listener] connection accepted from 172.19.0.3:57076 #1 (1 connection now open)
mongodb_1 | 2018-01-29T09:29:18.697+0000 I NETWORK [listener] connection accepted from 172.19.0.1:57824 #2 (2 connections now open)
mongodb_1 | 2018-01-29T09:29:22.210+0000 I NETWORK [listener] connection accepted from 172.19.0.1:57826 #3 (3 connections now open)

```

Once MongoDB is up, you can now test it using a MongoDB client. I use [Robomongo](#). Since we've configured it to bind to our local host machine, you can configure Robomongo to connect to it. Name does not matter, what is important is the address and port.



Good! So now we have our MongoDB running. Next step is to create the ff:

- Create a DATABASE named "db"
- Create a COLLECTION named "jobs"

So MongoDB is ready!

Next step is to create the microservice. I've already created the application, you can just download it from [my learn-go github repo](https://github.com/donvito/learn-go/blob/master/mongo-microservice/multistage/main.go).

Copy the main.go file from here.

<https://github.com/donvito/learn-go/blob/master/mongo-microservice/multistage/main.go>

The dependencies we need are the ff.:

```
import (  
    "encoding/json"  
    "fmt"  
    "log"  
    "net/http"  
    "time"  
    "io/ioutil"  
  
    "github.com/gorilla/mux"  
    "gopkg.in/mgo.v2/bson"  
    "gopkg.in/mgo.v2"  
  
)
```

The MongoDB driver used is from labix <https://labix.org/mgo>

To put configuration in one place, I've added it as constants

```
const (  
    hosts      = "dockercompose_mongodb_1:27017"  
    database   = "db"  
    username   = ""  
    password   = ""  
    collection = "jobs"  
  
)
```

I also created a struct which will be the data structure of the data to be saved in the database.

```
type Job struct {  
    Title      string `json:"title"`  
    Description string `json:"description"`  
    Company    string `json:"company"`  
    Salary     string `json:"salary"`  
  
}
```

Here is the method which initialises the MongoDB session. Timeout is set to 60 secs.

```
func initialiseMongo() (session *mgo.Session){  
  
    info := &mgo.DialInfo{  
        Addrs:    []string{hosts},  
        Timeout:   60 * time.Second,  
        Database:  database,  
        Username:  username,  
        Password: password,  
    }  
  
    session, err := mgo.DialWithInfo(info)  
    if err != nil {  
        panic(err)  
    }  
  
    return  
}
```

So here are the HTTP endpoints required for requirement #1. I used [gorilla mux](#) router for URL routing. I separated the 2 handlers so I don't have to do an if/else inside a common handler.

```
router := mux.NewRouter().StrictSlash(true)  
router.HandleFunc("/jobs", jobsGetHandler).Methods("GET")  
router.HandleFunc("/jobs", jobsPostHandler).Methods("POST")
```

These are handled by 2 additional methods.

```
func jobsGetHandler(w http.ResponseWriter, r *http.Request) {
```

```
col := mongoStore.session.DB(database).C(collection)

results := []Job{}
col.Find(bson.M{"title": bson.RegEx{"", ""}}).All(&results)
jsonString, err := json.Marshal(results)
if err != nil {
    panic(err)
}
fmt.Fprint(w, string(jsonString))
}

func jobsPostHandler(w http.ResponseWriter, r *http.Request) {

    col := mongoStore.session.DB(database).C(collection)

    //Retrieve body from http request
    b, err := ioutil.ReadAll(r.Body)
    defer r.Body.Close()
    if err != nil {
        panic(err)
    }

    //Save data into Job struct
    var _job Job
    err = json.Unmarshal(b, &_job)
    if err != nil {
        http.Error(w, err.Error(), 500)
        return
    }

    //Insert job into MongoDB
    err = col.Insert(_job)
    if err != nil {
        panic(err)
    }

    //Convert job struct into json
    jsonString, err := json.Marshal(_job)
    if err != nil {
        http.Error(w, err.Error(), 500)
        return
    }

    //Set content-type http header
    w.Header().Set("content-type", "application/json")
}
```

```
//Send back data as response  
w.Write(jsonString)  
  
}
```

Lastly, here is the code that spins up the http server of the microservice. You can change the port as you wish. But, note that you need to do some modifications to the Dockerfile to build and deploy the microservice to Docker.

```
log.Fatal(http.ListenAndServe(":9090", router))
```

For requirement #2, since we are using Go and Docker already supports multi-stage builds, I've created a Dockerfile to build this Go microservice.

Here is the single Dockerfile. The first block of code builds the application. The second block of code creates the docker image which can be deployed as a docker standalone container or as a service in Swarm or Kubernetes. Note that the EXPOSED PORT 9090 here should be the same port the microservice is running on.

```
FROM golang:1.9.2 as builder  
ARG SOURCE_LOCATION=/  
WORKDIR ${SOURCE_LOCATION}  
RUN go get -d -v github.com/gorilla/mux \  
    && go get -d -v gopkg.in/mgo.v2/bson \  
    && go get -d -v gopkg.in/mgo.v2  
COPY main.go .  
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .  
  
FROM alpine:latest
```



```
ARG SOURCE_LOCATION=/
RUN apk --no-cache add curl
EXPOSE 9090
WORKDIR /root/
COPY --from=builder ${SOURCE_LOCATION} .
CMD ["/app"]
```

This Dockerfile can be download from the same repo

<https://github.com/donvito/learn-go/blob/master/mongo-microservice/multistage/Dockerfile>

Cool, so we now have our building blocks! We can now build and run the microservice in Docker.

To make things simpler, here are the steps to get the microservice up and running in Docker.

1. First, do a git clone of the repo, let's assume you download it in the /Users/melvin/Downloads directory

```
$ git clone https://github.com/donvito/learn-go.git
$ cd learn-go/mongo-microservice/multistage/
```

After you've cloned and moved to the multistage directory, you can now build the microservice source code using the Dockerfile. Here is the command. SOURCE_LOCATION is the multistage directory of the project. It is where the source code of the microservice can be found.

```
$ docker build --build-arg SOURCE_LOCATION=/Users/melvin/Downloads/learn-go
```

Steps 1 to 6 is the compiling of the microservice's source code into a binary.

```
Mon Jan 29 06:02 PM[melvin@Melvin-MacBook:multistage]$ docker build --build-arg SOURCE_LOCATION=/Users/melvin/Downloads/learn-go-mongo-microservice/multistage --no-cache -t donvito/go-mongo-microservice:latest .
Sending build context to Docker daemon 5.12kB
Step 1/13 : FROM golang:1.9.2 as builder
--> 138bd936fa29
Step 2/13 : ARG SOURCE_LOCATION=/
--> Running in 6391caf71db4
Removing intermediate container 6391caf71db4
--> 8740ac5c3dab
Step 3/13 : WORKDIR ${SOURCE_LOCATION}
--> 8740ac5c3dab
Removing intermediate container d64c47583131
--> f89ffa3d0c08
Step 4/13 : RUN go get -d -v github.com/gorilla/mux && go get -d -v gopkg.in/mgo.v2/bson && go get -d -v gopkg.in/mgo.v2
--> Running in 604df26d57ce
github.com/gorilla/mux (download)
Fetching https://gopkg.in/mgo.v2/bson?go-get=1
Parsing meta tags from https://gopkg.in/mgo.v2/bson?go-get=1 (status code 200)
get "gopkg.in/mgo.v2/bson": found meta tag get.metaImport(Prefix:"gopkg.in/mgo.v2", VCS:"git", RepoRoot:"https://gopkg.in/mgo.v2") at https://gopkg.in/mgo.v2/bson?go-get=1
get "gopkg.in/mgo.v2/bson": verifying non-authoritative meta tag
Fetching https://gopkg.in/mgo.v2?go-get=1
Parsing meta tags from https://gopkg.in/mgo.v2?go-get=1 (status code 200)
gopkg.in/mgo.v2 (download)
Removing intermediate container 604df26d57ce
--> 7edc31cf6335
Step 5/13 : COPY main.go .
--> b26de1de3f00
Step 6/13 : RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
--> Running in 7e1184ab8d56
Removing intermediate container 7e1184ab8d56
--> f4cacd9e49fb
```

Steps 7 to 13 is creating the final docker image with the binary.

```
--> f4cacd9e49fb
Step 7/13 : FROM alpine:latest
--> 3fd9065eaf02
Step 8/13 : ARG SOURCE_LOCATION=/
--> Running in c703cdcbf754
Removing intermediate container c703cdcbf754
--> 523255491dea
Step 9/13 : RUN apk --no-cache add curl
--> Running in 766fa50c2010
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/4) Installing ca-certificates (20171114-r0)
(2/4) Installing libssh2 (1.8.0-r2)
(3/4) Installing libcurl (7.57.0-r0)
(4/4) Installing curl (7.57.0-r0)
Executing busybox-1.27.2-r7.trigger
Executing ca-certificates-20171114-r0.trigger
OK: 6 MiB in 15 packages
Removing intermediate container 766fa50c2010
--> d94baeb9483c
Step 10/13 : EXPOSE 9090
--> Running in 3d86119358b4
Removing intermediate container 3d86119358b4
--> bcbb07ada838
Step 11/13 : WORKDIR /root/
Removing intermediate container d7e8e6abfe8e
--> da3fb412eca0
Step 12/13 : COPY --from=builder ${SOURCE_LOCATION} .
--> d0f59a27baf6
Step 13/13 : CMD ["/app"]
--> Running in 460fc0f6bb67
Removing intermediate container 460fc0f6bb67
--> 3b718465ff7c
Successfully built 3b718465ff7c
Successfully tagged donvito/go-mongo-microservice:latest
Mon Jan 29 06:02 PM[melvin@Melvin-MacBook:multistage]$
```

To check if the docker image is created, just do a `$docker image ls`.
Notice that the docker image is only 13MB!

```
Mon Jan 29 06:09 PM[melvin@Melvin-MacBook:multistage]$ docker image ls
REPOSITORY              TAG          IMAGE ID          CREATED           SIZE
donvito/go-mongo-microservice  latest       3b718465ff7c     15 minutes ago   13MB
```

To run the microservice, use this command. Note that I binded to my machine's port 8000 since 9090 is not available anymore.

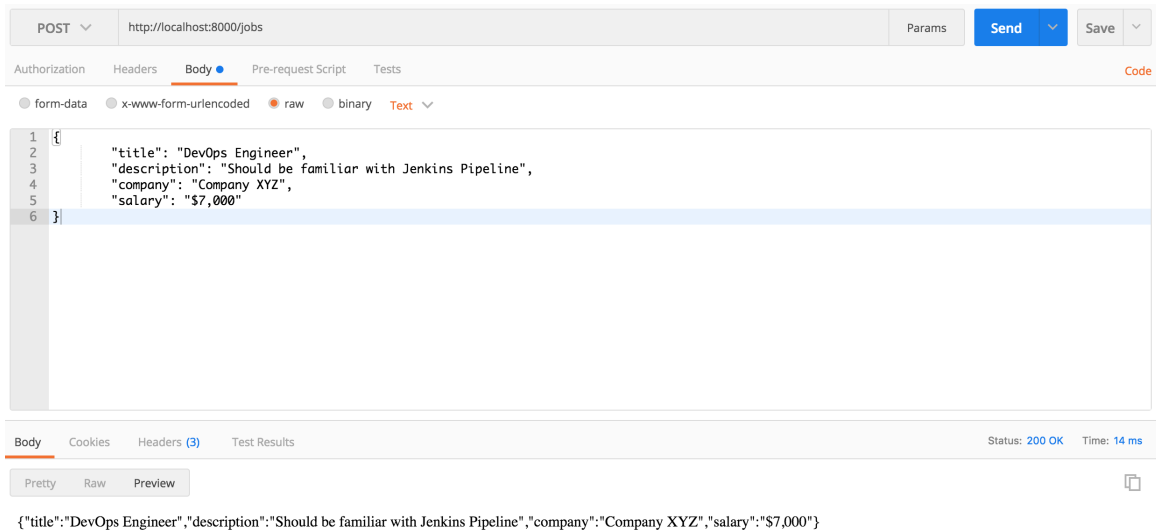
```
docker run --name go-mongo-microservice -d --rm -p 8000:9090 --network d
```

To check if the service is running, just do a `$docker ps`.

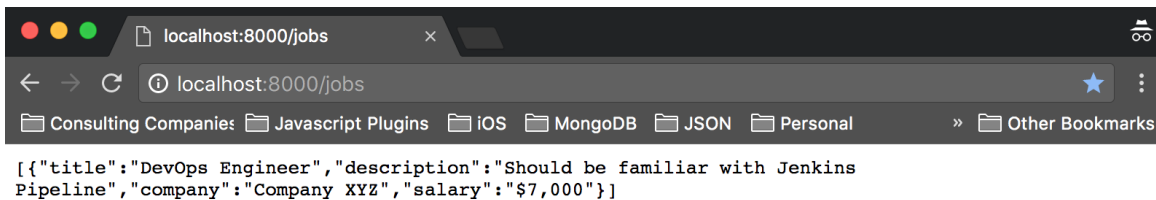
```
Mon Jan 29 06:09 PM[melvin@Melvin-MacBook:multistage]$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                    NAMES
d2a5f1dc7245   donvito/go-mongo-microservice:late  "/app"                  About a minute Up About a minute  0.0.0.0:8000->9090/tcp   go-mongo-microservice
855daea587f9   mongo                               "docker-entrypoint.s..." 5 hours ago    Up 40 minutes  0.0.0.0:27017->27017/tcp  dockercompose_mongodb_1
```

Since MongoDB does not have data yet, let's insert a sample record. I used [Postman](#) but curl will also work! You can use this payload.

```
{
  "title" : "DevOps Engineer",
  "description" : "Should be familiar with Jenkins Pipeline",
  "company" : "Company XYZ",
  "salary" : "$7,000"
}
```



To check if the microservice is able to retrieve data from MongoDB, access the GET `/jobs` endpoint. Using your browser will do of course :)



Entire source code is available in my github repo.

<https://github.com/donvito/learn-go/tree/master/mongo-microservice>

Feel free to give me a heads up if you have trouble making the example work!



Melvin Dave Vivas

Read [more posts](#) by this author.


[Read More](#)

0 Comments - powered by [utteranc.es](#)

Write

Preview

Sign in to comment

 Styling with Markdown is supported

Sign in to comment

DOCKER

Configuring Travis CI with My Go Projects

Travis CI is a CI(Continuous Integration) tool which can help you build your applications. We are currently using Jenkins at work but I

Kubernetes-powered Docker CE (Community Edition)

When Solomon Hykes, CTO of Docker, announced in DockerCon 2017 last October that they will be supporting Kubernetes natively, I was pretty curious

thought of giving Travis CI a spin since
it



MELVIN DAVE VIVAS

how that would work. Obviously, after
the announcement, I checked



MELVIN DAVE VIVAS

Melvin Vivas © 2020

[Latest Posts](#) [Facebook](#) [Twitter](#) [Ghost](#)