**ECE 530 Homework 3: Linux Containers**

Manoj Kumar Nannapaneni 101897131 [manojnannapaneni@unm.edu](mailto:manojnannapaneni@unm.edu)

Manideep Potluru 101879531 [mpotluru@unm.edu](mailto:mpotluru@unm.edu)

T Sai Aditya 101863023 [aditya369thalluri@unm.edu](mailto:aditya369thalluri@unm.edu)

**Introduction**

The purpose of this assignment is to create a distributed network of containers with database instances connecting each other by deploying Docker instances un a host machine. In this assignment we have installed docker engine on our host machine and generated three docker machines. Inside each machine a mongodb image file as a container along with a docker volume has been created. Admin, replica and key authentication files were created and passed to the docker container. This will be called as docker file which will build the images automatically for mongodb instance. Later, the same configurations of mangobd containers were created for the other docker machines using the replica and the containers were connected to the replica set of the first container.

**Docker**

Docker [1] is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers is an environment which allows a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code. In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application. Importantly, Docker is open source. This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that are not available out of the box.

**Containers**

A Container [2] is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging. So basically, Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.

**Virtual Machine vs Container**

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient. Containers take up less space than VMs can handle more applications and require fewer VMs and Operating systems. Difference in the architecture between virtual machine and container can be seen below.

A screenshot of a cell phone

Description automatically generated

Figure 1 Comparison between the architectures of virtual machine and contaniner

**Docker Installation**

* Existing packages must be updated in the operating system

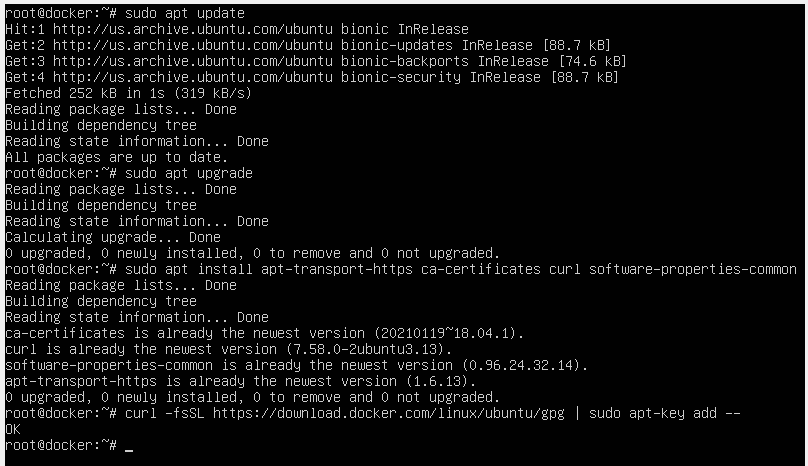
$ sudo apt update

* Prerequisite packages have to be installed

sudo apt install apt-transport-https ca-certificates curl software-properties-common

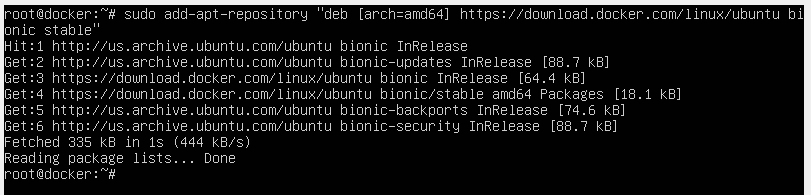
* GPG key has to be added for the official Docker repository to the system

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –



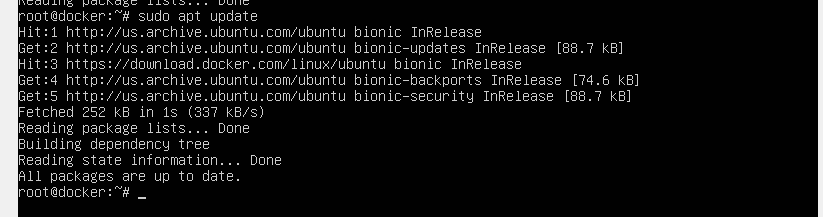
* Docker repository should be added to APT sources

$ sudo add-apt-repository “deb [arch=amd64] <https://download.docker.com/linux/ubuntu> bionic stable”



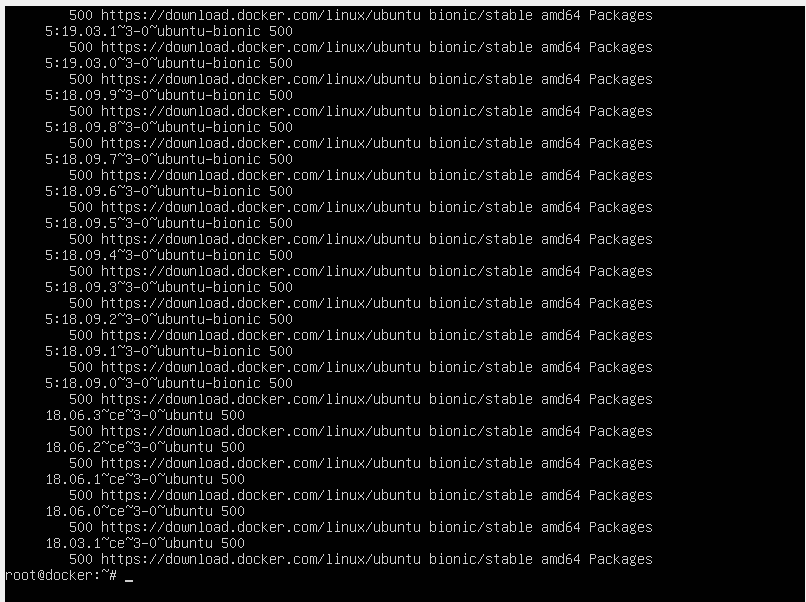
* Update the package directory with the Docker packages from the newly added repo

$ sudo apt update



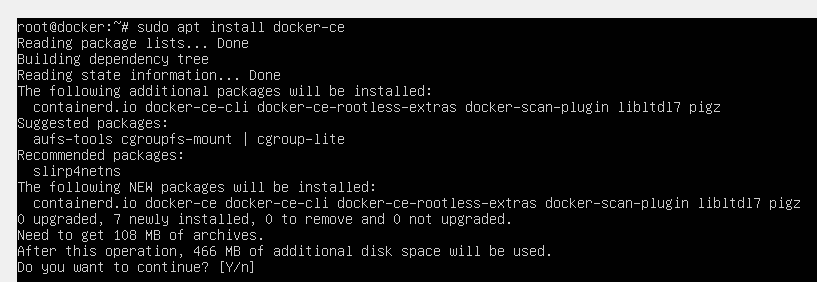
* Verify that installation has to be done from the Docker repo instead of the default Ubuntu repo

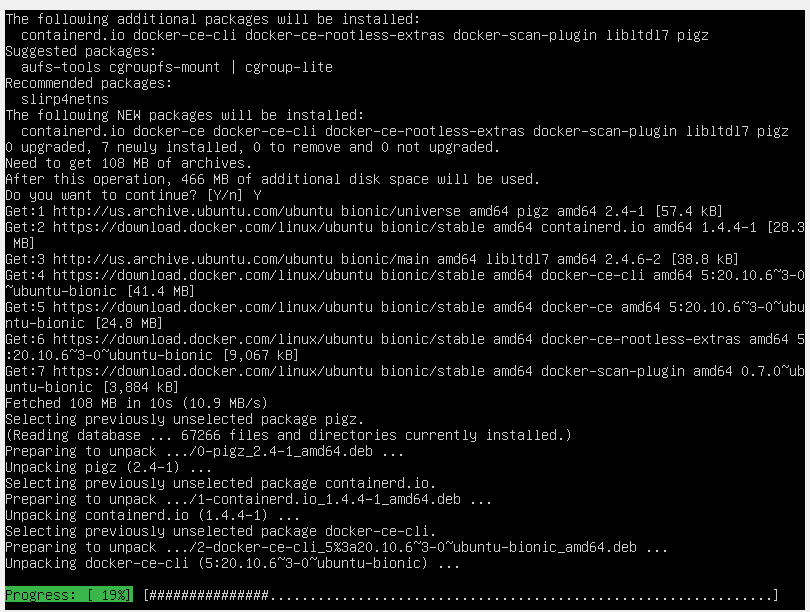
$ apt-cache policy docker-ce

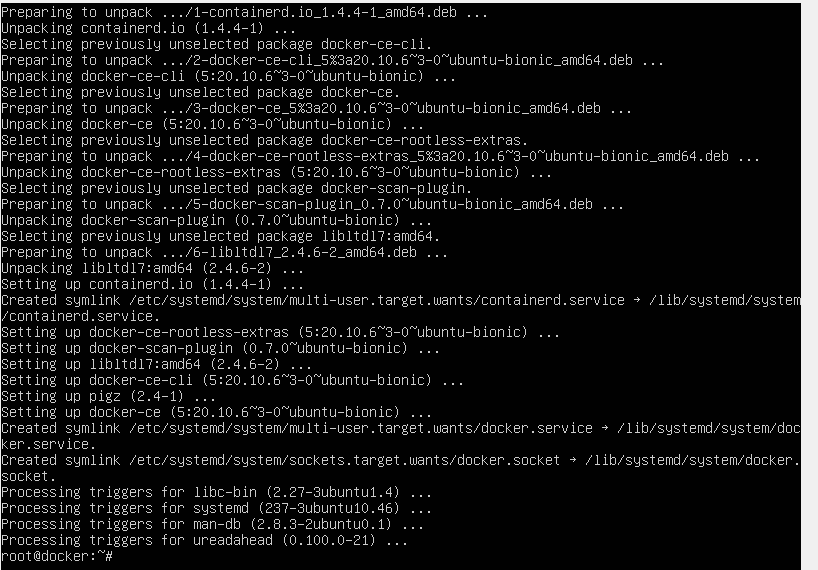


* Now, install Docker

$ sudo apt install docker-ce

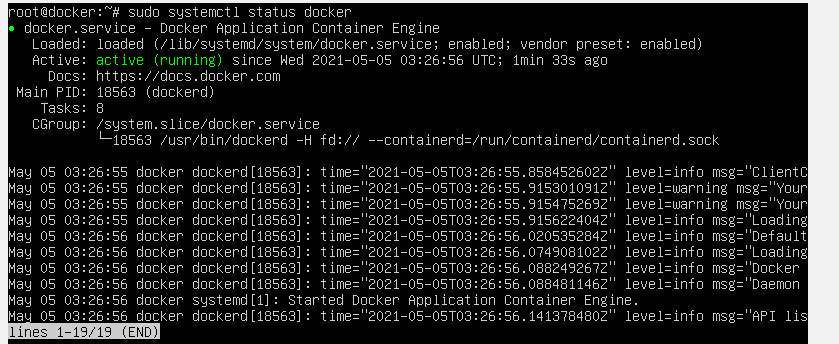






* Docker should be installed now so need to check the status of Docker

$ sudo systemctl status docker

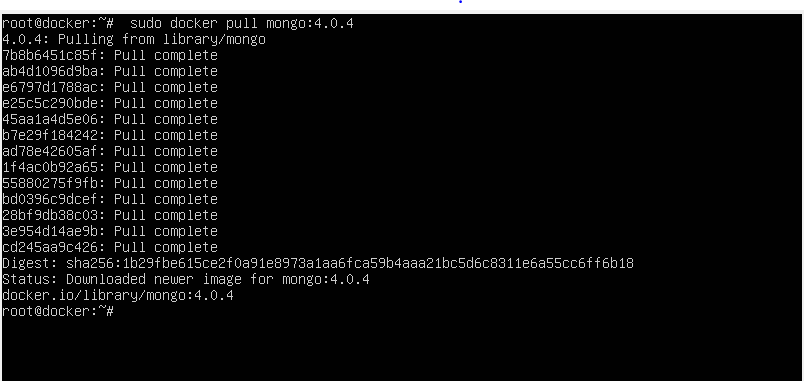


**Docker file for MongoDB**

A docker file has to be created using the debian as base image from the ‘debian:jessie-slim’ on docker hub. On top of that we installed certificates, setup gpg keys and installed mongo 4.0.4 from mongodb repo.We have setup the data directory, work directory and exposed 27017 port.

We then built the image from docker file using command

$ sudo docker pull mongo:4.0.4



**Creating a Network:**

* A network must be created for the containers to be able to communicate with each other. So, a network named `mongo-cluster` has been created using the command

$ sudo docker​ network create mongo-cluster

* Verify the network creation ‘mongo-cluster’

$ sudo docker network ls



**Running Containers:**

By using the image created from docker file, run three containers with mongodb set x to run with replication set `myrepl`. the mongo-cluster network using `--net` option for all three containers so we have to set port forwarding for all there containers.

**Container 1**

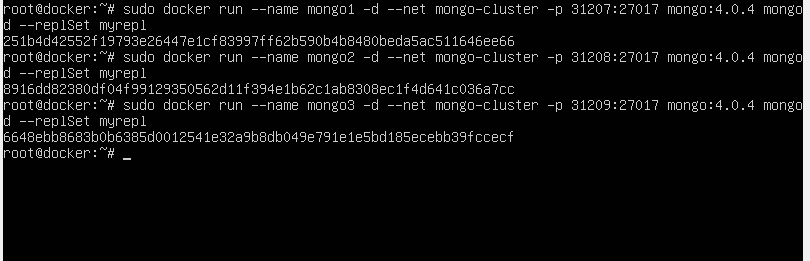
$ sudo docker run --name mongo1 -d --net mongo-cluster -p 31207:27017 mongo:4.0.4 mongod --replSet myrepl`

**Container 2**

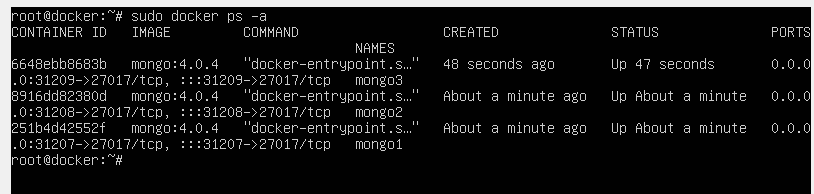
$ sudo docker run --name mongo2 -d --net mongo-cluster -p 31208:27017 mongo:4.0.4 mongod --replSet myrepl`

**Container 3**

$ sudo docker run --name mongo3 -d --net mongo-cluster -p 31209:27017 mongo:4.0.4 mongod --replSet myrepl`



Verify all three containers running successfully in the image below.

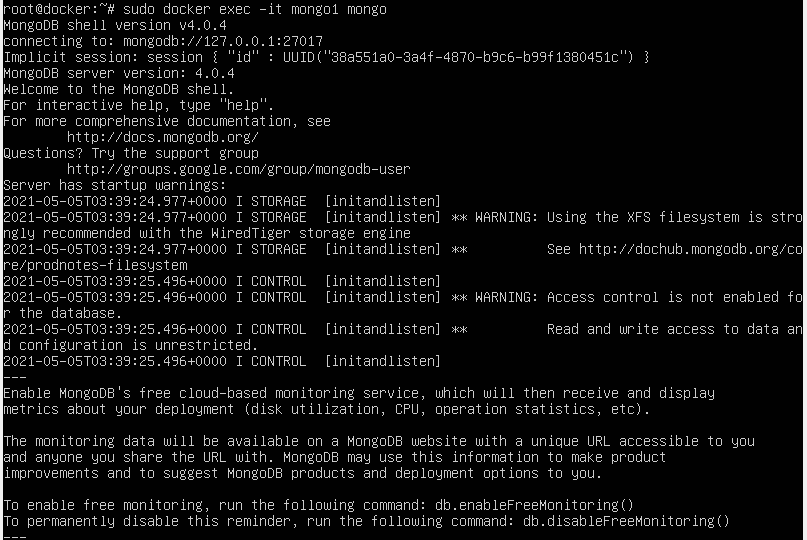


**Configuration of mongodb to replicate data**

There are three containers up and running. Now, the task is to set mongodb to replicate data over the three containers. We have chosen the container `**mongo1**​` to be master, `**mongo2**​` and ` **mongo3**​` to be slaves. We insert data into master and check if it will be replicates in slaves.

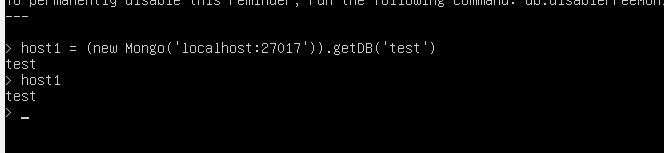
To do this, first open mongo prompt on master container.

$ sudo docker**​** exec -it mongo1 mongo



Now get the test database on the container using the command.

`host1 = (new Mongo('localhost:27017')).getDB('test')`



Then create and initiate the replication configuration

config={"\_id":"myrepl","members":[{"\_id":0,"host":" mongo1:27017"},{"\_id":1, "host":" mongo2:27017"},{"\_id":2,"host":" mongo3:27017"}]}

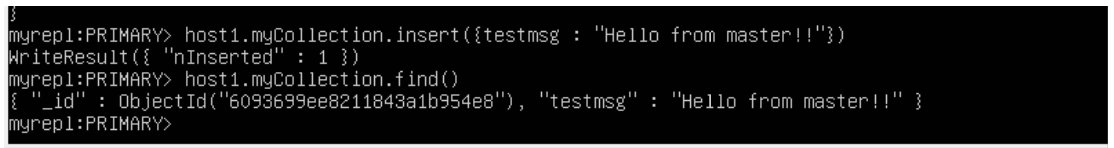
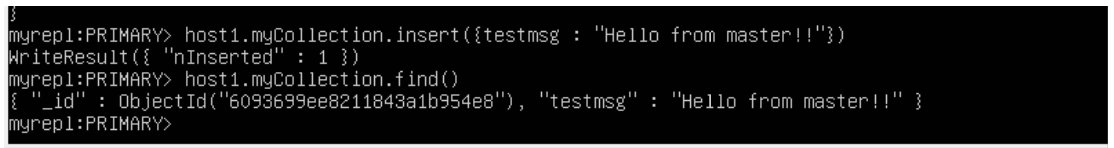
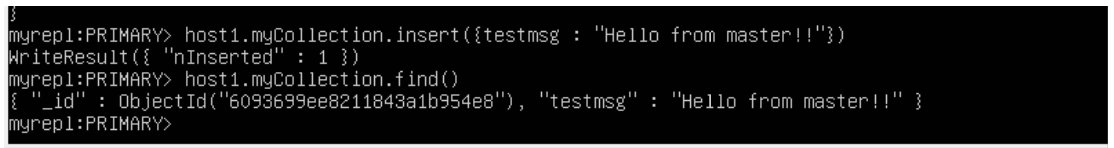
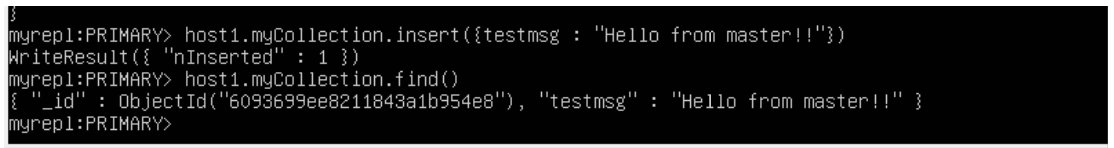
rs.initiate(config)



**Inserting data and checking replication**

Since **mongo1**​ is the master, insertion of data should be done only through this container.

Data can be inserted into **mongo1** using the command ‘host1.myCollection.inser({testmsg : “Hello from master!!”})’Once insertion of data is done in master container, it can be checked by using the command ‘host1**​**.myCollection.find()’. Now data should be replicated in other two slave containers.

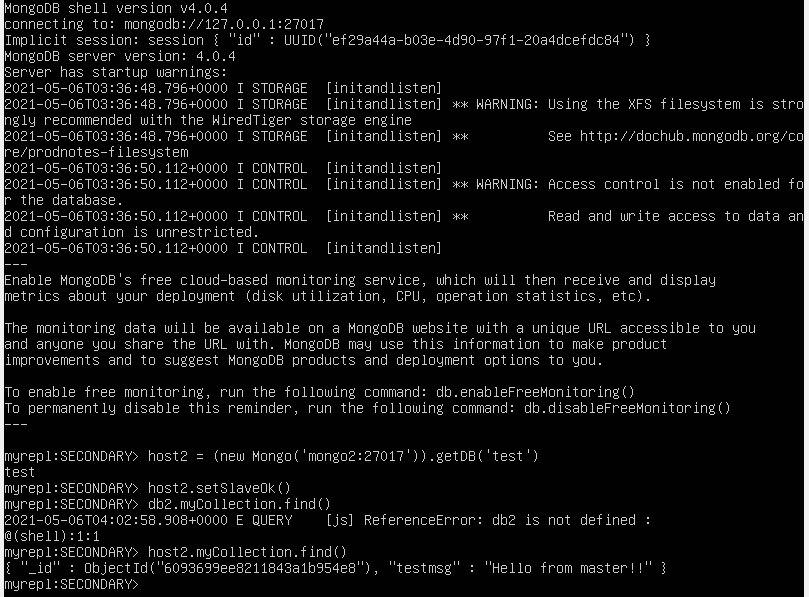
**Checking on mongo2**

$ sudo docker**​** exec -it mongo2 mongo

Create test database instance using the command ‘host2 = (new Mongo('localhost:27017')).getDB('test')’

Set mongo2 as slave using the command ‘host2.setSlaveOk()’**​**

Now check the data replication on mongo2 using the command host2.myCollection.find()



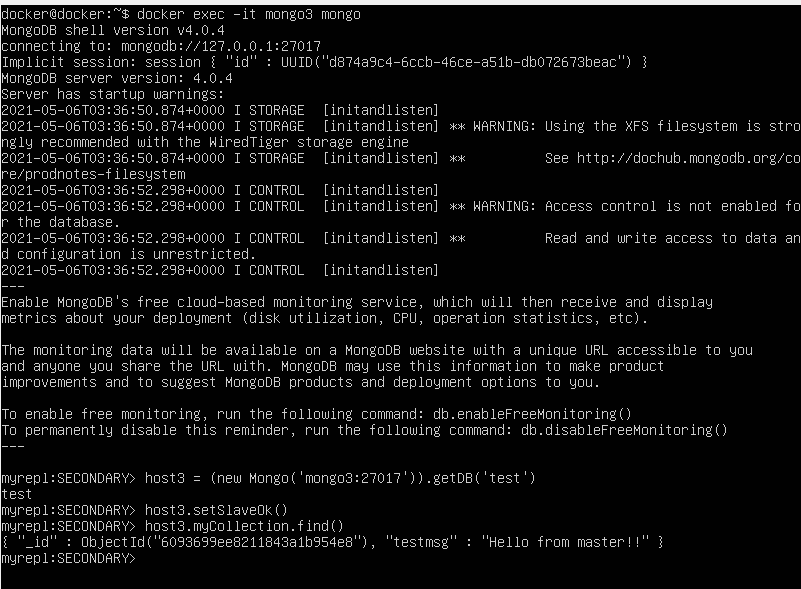
**Checking on mongo3**

$ sudo docker**​** exec -it mongo3 mongo

Create test database instance using the command ‘host3 = (new Mongo('localhost:27017')).getDB('test')’

Set mongo3 as slave using the command ‘host3.setSlaveOk()’**​**

Now check the data replication on mongo3 using the command host3.myCollection.find()

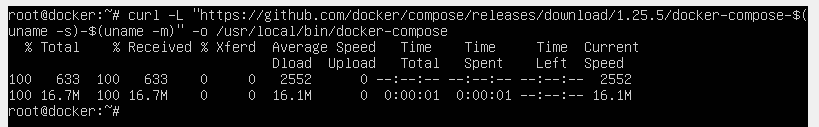


**Docker Compose (Extra Credit)**

Compose is a tool for defining and running multi-container Docker applications. With Compose, we use a YAML file to configure your application’s services. Then, with a single command, we create and start all the services from our configuration.

• Download stable release of Docker compose using the following command:

$ sudo curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose



• Apply executable permissions to the binary:

$ sudo chmod +x /usr/local/bin/docker-compose

• Following application can be used to count the number of times a person visits a page. The application is built using Python Flask framework and maintains a counter in Redis that increments each time the page is visited. The count is maintained in cache.

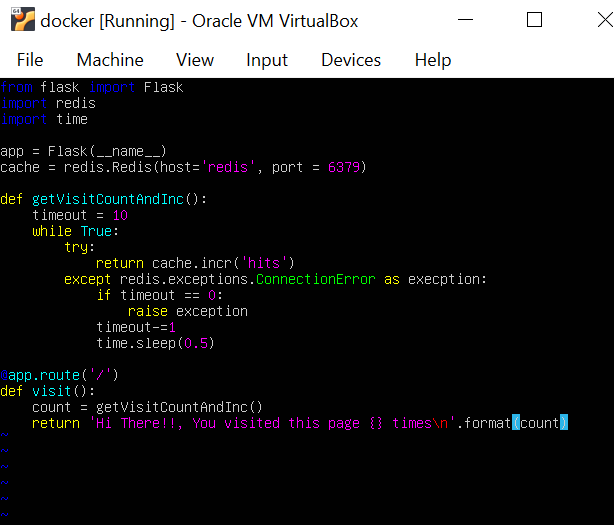
• Create a directory for the project:

$ mkdir composetest

$ cd composetest



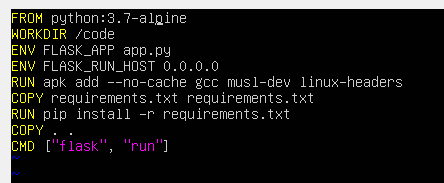
* Create the .py file with the application code in it. Place this file in the project directory:



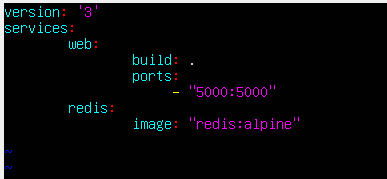
* Create another file called requirements.txt in your project directory that contains requirements needed to run the project. Place this in project directory:



• Create the Docker file that contains the workflow



* Create a file docker-compose.yml file in our project directory and paste the following. This file is used for application startup.

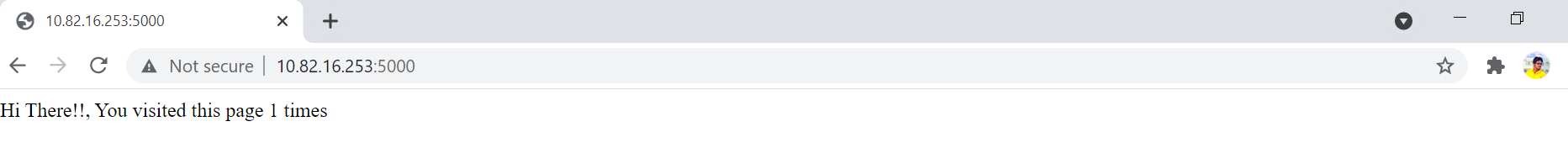


* From the project directory start the application by running the following command.

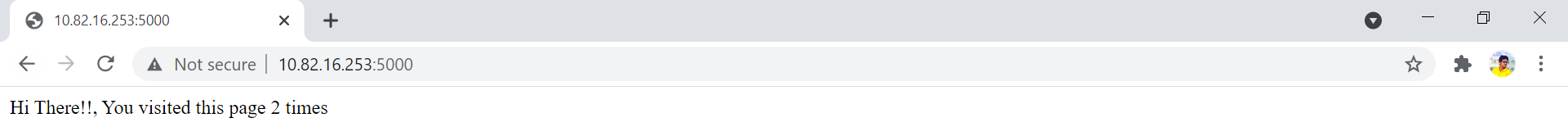
$ docker-compose up

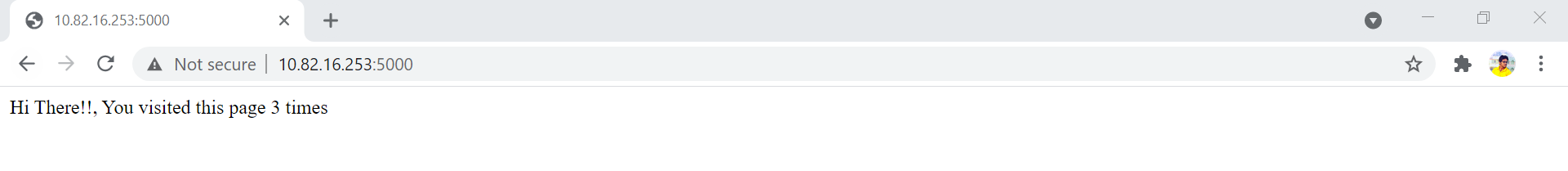


* I am using a VM, so I used the bridge network with local machine to hit the server on the VM. 10.82.16.253 is that bridge IP.



* Try to hit the page again or refresh the page to see the increments.





**Conclusion**

This assignment helped us to learn about containers through installation and creation of docker containers with mongodb using docker file. A network has been established between the three containers that are created in which one being primary and other two replicas and successfully verified how operations on primary containers reflected on slave containers. So, we realize that dockers can boost the deployment process as they are simple and scalable which can also be easily configured and maintained. It also avoids the dependency of runtime environment as containers are isolated from each other which results in better control over traffic flow and management.

**References**

[1] <https://opensource.com/resources/what-docker>

[2] <https://www.docker.com/resources/what-container>