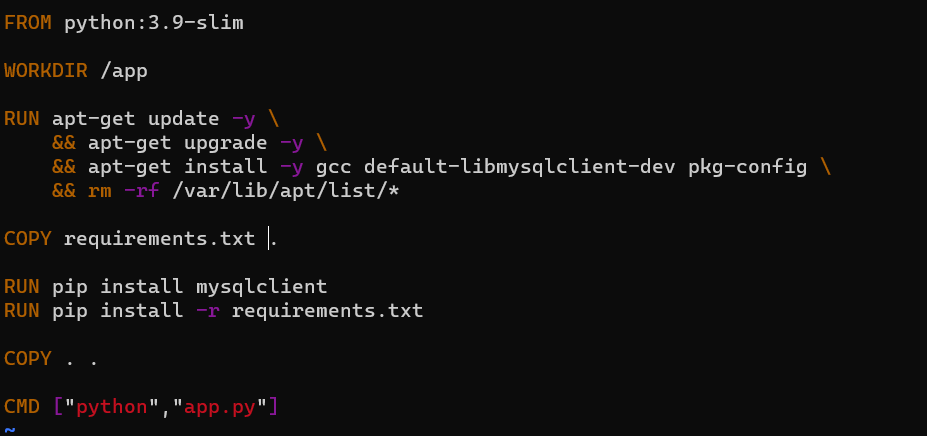
In this project I have deployed a two-tier flask app by

1. Dockerizing the app- the flash app as well as the database
2. Pushing the build images on DockerHub
3. Deploying the same app on Kubernetes using kubeadm with Helm templates and
4. Uploading it to AWS EKS
5. DOCKERISING THE FLASK APP AND THE DATABASE
6. Provision and EC2 Instance
7. Install Docker
8. Make a Dockerfile for the flask application



1. Create the image of the container by following command:

*docker build . -t flaskapp*

1. Create the container by the following command:

*docker run -d -p 5000:5000 --network=twotier -e MYSQL\_HOST=mysql -e MYSQL\_USER=admin -e MYSQL\_PASSWORD=admin -e MYSQL\_DB=myDB --name=flaskapp flaskapp:latest*

1. Create an SQL container. We will use a direct approach from the CLI this time with the following command:

*docker run -d -p 3306:3306 --network=twotier -e MYSQL\_DATABASE=myDB -e MYSQL\_USER=admin -e MYSQL\_PASSWORD=admin -e MYSQL\_ROOT\_PASSWORD=admin --name=MYSQL mysql:5.7*

1. Although, we can get our container running with the above command we will use docker compose.
2. Now, since our dockerfile is correct, we will push the dockerfile to our DockerHub repo. So that we can use it with Docker Compose
3. In order to push the dockerfile, we do the following:
   * + - First login to docker hub by entering the below command and enter Username and Password

*docker login*

* + - * Rename the docker image in this format:

*docker tag flaskapp:latest zamanmd/flaskapp:latest*

* + - * Use the following command to push the image on DockerHub:

*Docker push zamanmd/flaskapp:latest*

Now, if you have multiple containers to make, instead of creating a container one after the other, we can make a docker-compose file.

Ensure that you have installed Docker Compose on your system. To install docker compose, use the following command:

*sudo apt install docker compose*

1. Now we create the Docker compose file as follows:

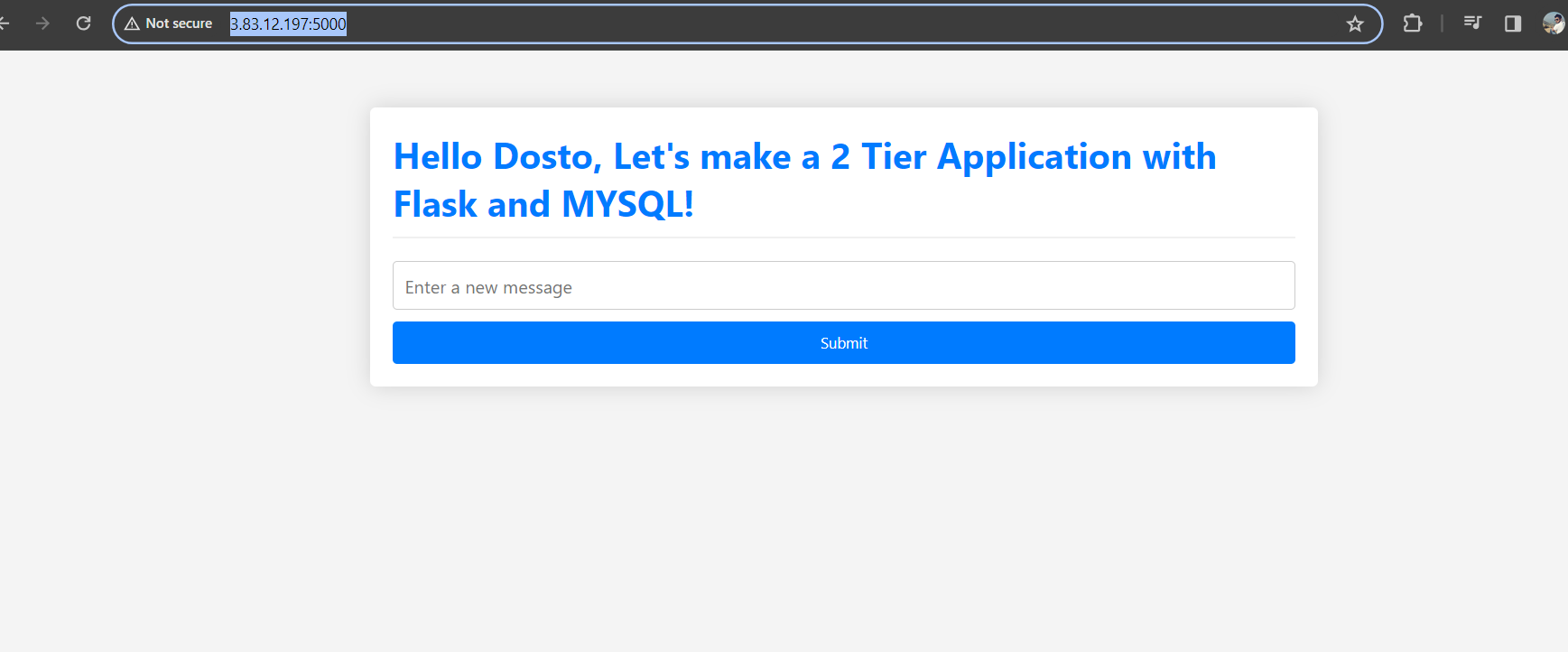
A computer screen shot of text

Description automatically generated

After create the docker compose file, we ca run the run both containers using the following command:

*docker-compose up -d*

Now both your containers will be running successfully. Use the IP address followed by the port number to access you application on the web-browser. E.g., <http://3.83.12.197:5000/>



With this our application has been dockerised.

1. SETTING UP KUBERNETES KUBEADM ON EC2 INSTANCE
2. Provision two EC2 instances on AWS. Preferred resource type for learning is t2.medium and use Ubuntu
3. Rename the instances (if you have not already) to Master Node and Worker Node for your own reference
4. Use the following commands to setup your k8s cluster on Worker and Master both:

*# using 'sudo su' is not a good practice.*

*-----*

*sudo apt update*

*sudo apt-get install -y apt-transport-https ca-certificates curl*

*sudo apt install docker.io -y*

*sudo systemctl enable --now docker # enable and start in single command.*

*# Adding GPG keys.*

*curl -fsSL "https://packages.cloud.google.com/apt/doc/apt-key.gpg" | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/kubernetes-archive-keyring.gpg*

*# Add the repository to the sourcelist.*

*echo 'deb https://packages.cloud.google.com/apt kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list*

*sudo apt update*

*sudo apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y*

-------

1. On master or control plane node initialize the kubeadm by the following command:

*kubeadm init*

1. Set up local kubeconfig (both for root user and normal user:

*mkdir -p $HOME/.kube*

*sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config*

*sudo chown $(id -u):$(id -g) $HOME/.kube/config*

1. Find out how many nodes are running on this server:

*Kubectl get pods*

Right now our worker not is not connected. So now we will connect the worker node with the controlplane

1. Create a network for controlplane and worker node. This is known as Weave network:

*kubectl apply -f* [*https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml*](https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml)

1. Generate a token for worker nodes to join:

*sudo kubeadm token create --print-join-command*

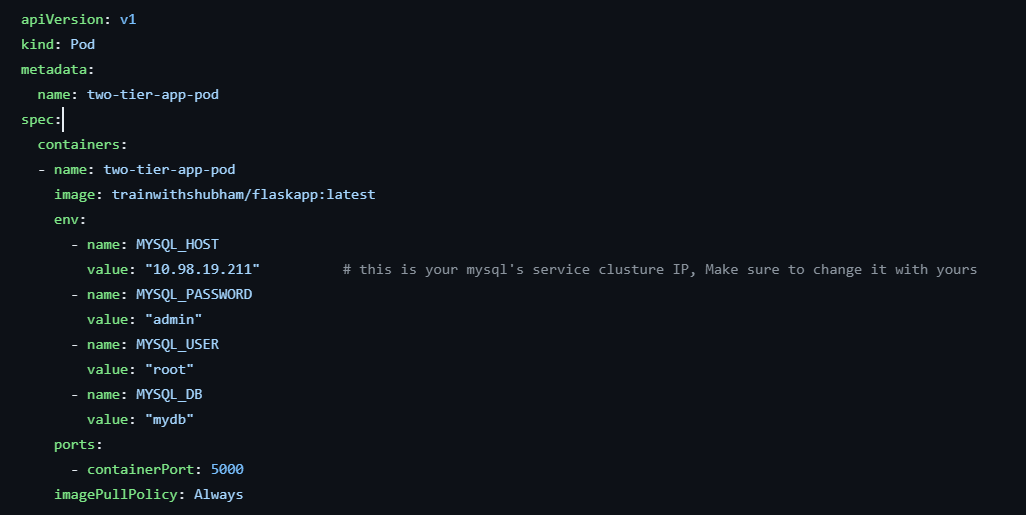
The above command will generate a token which can be used by any node to join this server. So, copy this token and go to the worker node which you want to attach to this control plane.

1. ON THE WORKER NODE.

**Ensure that we have not initialized kubeadm here** on the worker node:

*sudo kubeadm reset pre-flight checks*

1. Now paste the token generated from control plane to joing this worker node. Use sudo and –v=5
2. If it is failing and it is bound to because you have to setup the inbound traffic on your controlplane ec2 resource to accept traffic on this port. So do that and it will work
3. This Node will join the cluster now.
4. Go to control plane and now type kubectl get nodes. You will get this new node now.
5. SETTING UP OUR APP ON KUBERNETES
6. On the worker node, create a directory and cd into it to make all the required K8s resources.
7. POD: Use the VIM editor to make the following Pod



1. Now we can run the pod by:

*kubectl apply -f two-tier-app-pod.yml*

1. To scale we will make a deployment.

A computer screen shot of a program code

Description automatically generated

The above screenshot container the service ip which we will get later

kubectl apply -f two-tier-app-deployment.yml

1. Now lets create a service manifest for the outside word to access our containers

vi two-tier-app-service.yml

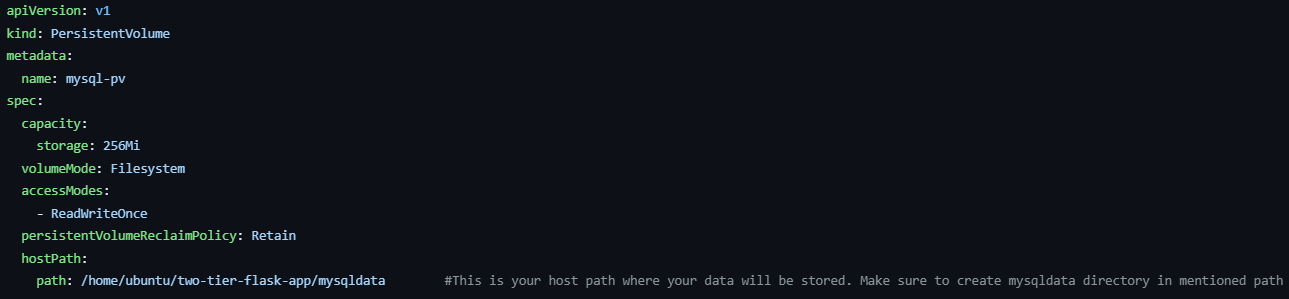
A screen shot of a computer program

Description automatically generated

*kubectl apply -f two-tier-app-service.yml*

1. Create the mysql container/ pod. But before creating the mysql pod we need a persistent volume and persistent volume claim.

*Vi mysql-pv.yml*

**

*Kubectl apply -f mysql-pv.yml*

*Vi mysql pv-claim.yml*

*A screen shot of a computer program

Description automatically generated*

*Vi mysql pv-claim.yml*

Now create the mysql pod

*vi mysql-deployment.yml*

*A screen shot of a computer program

Description automatically generated*

*Kubectl apply -f mysql-deployment.yml*

1. Create a service so that mysql pod can connect to our two-tier app

Vi mysql-svc.yml

A screen shot of a computer program

Description automatically generated

Kubectl apply -f mysql-svc.yml

1. Change the mysql server host in the two-tier-app-deployment

Vi two-tier-app-deployment.yml

A computer screen shot of a program code

Description automatically generated

If you are getting the error that the “messages” table does not exist, simply go to worker node and create the bale in the mysql container.

1. HELM SETUP
2. To install Helm:

*curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null*

*sudo apt-get install apt-transport-https --yes*

*echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list*

*sudo apt-get update*

*sudo apt-get install helm*

1. After installation, if you type “helm” you will get helm options.
2. USING THE HELM CHART FOR MYSQL
3. Let’s get a helm template of mysql. We will use this template to edit and create our own sql pod.

helm create mysql-chart

1. By default, we will get an nginx chart and we have to modify it.
2. Go to mysql directory
3. Modify the values.yml
4. ReplicaCount: 1
5. Image repository: mysql
6. Tag: latest
7. Service port: 3306
8. Save this file.
9. Now, add the environment variables. To do that, we enter all the environmental values in values.yml

Image\*

1. and then make the same changes in the deployment.yml file. We will basically enter the path where are values are stored.

Image\*

1. There is a small indentation error in the template where we have to edit in the template/serviceaccount. Go to the file and fix the indentation error

Comment out or remove the liveness probe and readinessprobe in the deployment.yml

1. We have our template ready now we will package the app:

*Helm package mysql-chart*

And install the same to deploy

*Helm install mysql-chart ./ mysql-chart*

1. Create the flask-app-chart

Go inside the flash-app-chart

Vi values.yml

Change the image name : zamanmd/flashapp

Tags: latest

Under tags enter all the env variables:

Mysqlhost: (get the IP of mysql service)

Mysqlpw: admin

Mysqldb: mydb

Under Service:

Type: NodePort

Port: 80

targetPort: 5000

nodePort: 30007

Save the values.yml

1. Now, these values will be added in the deployment.yml

Vi deployment.yml

Just above port:

Enter the environment variable:

* Name: MYSQL\_PASSWORD

Value: {{.values.env.mysqlhost}}

* Name: MYSQL\_PASSWORD

Value: {{.values.env.mysqlpw}}

* Name: MYSQL\_USER

Value: {{.values.env.mysqluser}}

* Name: MYSQL\_DB

Value: {{.values.env.mysqldb}}

* Under ports change the containerPort to TargetPort instead of port
* Comment or remove out the liveness prove and readinessprobe
* Fix the automount indentation error from service.yml and
* Change the target port to .vales.targetport
* Add the nodeport and enter the .values.nodeport

Save the file and

1. Check the manifest file:

*Helm template flask-app-chart*

This command will display all the manifest files. Just check here if everything is ok

If everything is ok then package your flaskapp

*Helm package flask-app-chart*

Install flask app to be deployed:

*Helm install flask-app-chart ./flask-app-chart*

1. DEPLOYMENT ON EKS
2. Create an EC2 instance with the above specs
3. Install AWS CLI
4. Create an IAM User giving him all the necessary permission to access and manage your instance and get the access key for the CLI
5. Open your EKS instance and login using the above CLI keys

Aws configure

Enter you user and key

1. Install kubectl

To check:

Kubectl version

1. Install eksctl

Eksctl version

1. To create the cluster here we will enter the following command

*Eks create cluster –name tws-cluster –region us-east-1 –node-type t3.small –node-min 2 –node-max 3*

It will take some time to make this cluster.

We have created the following manifest files here:

Mysql-secret.yml

Mysql-config.yml

Mysql-deployment.yml

Mysql-service.yml

So apply these files to get the mysql pod ready:

*Kubectl apply -f mysql-secret.yml -f mysql-config.yml -f mysql-deployment.yml -f mysql-service.ym*l

And now you can run the flask app

*Kubectl apply -f two-tier-app-deployment.yml two-tier-app-svc.yml*

To access the app

Kubectl get scv

You will get the address of the loadbalancer. Copy the address and paste it on your browser.

Our app is up and running.