**Infrastructure Optimization**

DESCRIPTION

Create a DevOps infrastructure for an e-commerce application to run on high-availability mode.

**Background of the problem statement:**  
A popular payment application, **EasyPay**where users add money to their wallet accounts, faces an issue in its payment success rate. The timeout that occurs with  
the connectivity of the database has been the reason for the issue.  
While troubleshooting, it is found that the database server has several downtime instances at irregular intervals. This situation compels the company to create their own infrastructure that runs in high-availability mode.  
Given that online shopping experiences continue to evolve as per customer expectations, the developers are driven to make their app more reliable, fast, and secure for improving the performance of the current system.

**Implementation requirements:**

1. **Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)**
2. **Automate the provisioning of an EC2 instance using Ansible or Chef Puppet**
3. **Install Docker and Kubernetes on the cluster**
4. **Implement the network policies at the database pod to allow ingress traffic from the front-end application pod**
5. **Create a new user with permissions to create, list, get, update, and delete pods**
6. **Configure application on the pod**
7. **Take snapshot of ETCD database**
8. **Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured**

**The following tools must be used:**

1. EC2
2. Kubernetes
3. Docker
4. Ansible or Chef or Puppet

**The following things to be kept in check:**

1. You need to document the steps and write the algorithms in them.
2. The submission of your GitHub repository link is mandatory. In order to track your tasks, you need to share the link of the repository.
3. Document the step-by-step process starting from creating test cases, then executing them, and recording the results.
4. You need to submit the final specification document, which includes:

* Project and tester details
* Concepts used in the project
* Links to the GitHub repository to verify the project completion
* Your conclusion on enhancing the application and defining the USPs (Unique Selling Points)

**Prerequisites:** kubeadm and kubectl should be installed

Contents

[1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS) 3](#_Toc82460401)

[2. Automate the provisioning of an EC2 instance using Ansible 5](#_Toc82460402)

[**Step 1** - Create AWS IAM User for provisioning through anisible 5](#_Toc82460403)

[**Step 2** - Install python 5](#_Toc82460404)

[**Step 3** - Install python3-pip 5](#_Toc82460405)

[**Step 4** - Install ansible 6](#_Toc82460406)

[**Step 5** - Configure ansible “localhost” 6](#_Toc82460407)

[**Step 6** - Create ansible playbook to create ec2 nodes: playbook.html 6](#_Toc82460408)

[**Step 7** - Run the playbook to get general facts and build ec2 instances 8](#_Toc82460409)

[**Step 8** - Create ansible host resource file 8](#_Toc82460410)

[**Step 9** - Test the connection to ec2 instances 9](#_Toc82460411)

[3. Install Docker and Kubernetes on the cluster 10](#_Toc82460412)

[**Step 1** – Create Kubernetes user with Ansible 10](#_Toc82460413)

[**Step 2** – Install Docker 10](#_Toc82460414)

[**Step 3** - Install Kubernetes 12](#_Toc82460415)

[4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod 15](#_Toc82460416)

[**Step 1** – Create the database 15](#_Toc82460417)

[**Step 2** – Create the application 19](#_Toc82460418)

[5. Create a new user with permissions to create, list, get, update, and delete pods 27](#_Toc82460419)

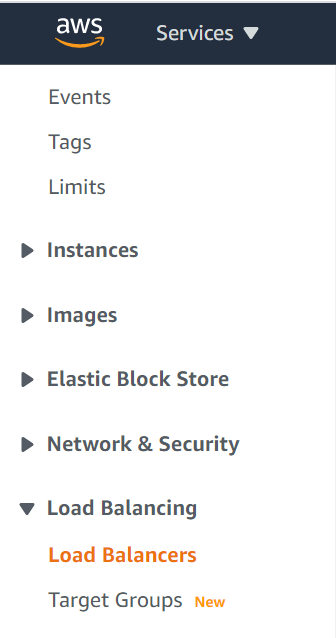
[6. Configure application on the pod 29](#_Toc82460420)

[7. Take snapshot of ETCD database 29](#_Toc82460421)

[8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured 31](#_Toc82460422)

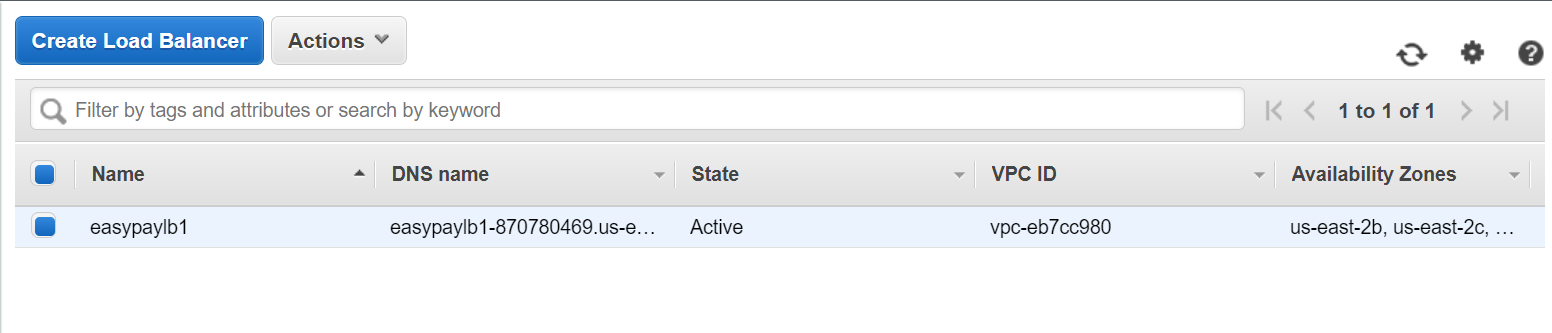
[9. Project Repositories, Credits and Reference Examples 34](#_Toc82460423)

# 1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)



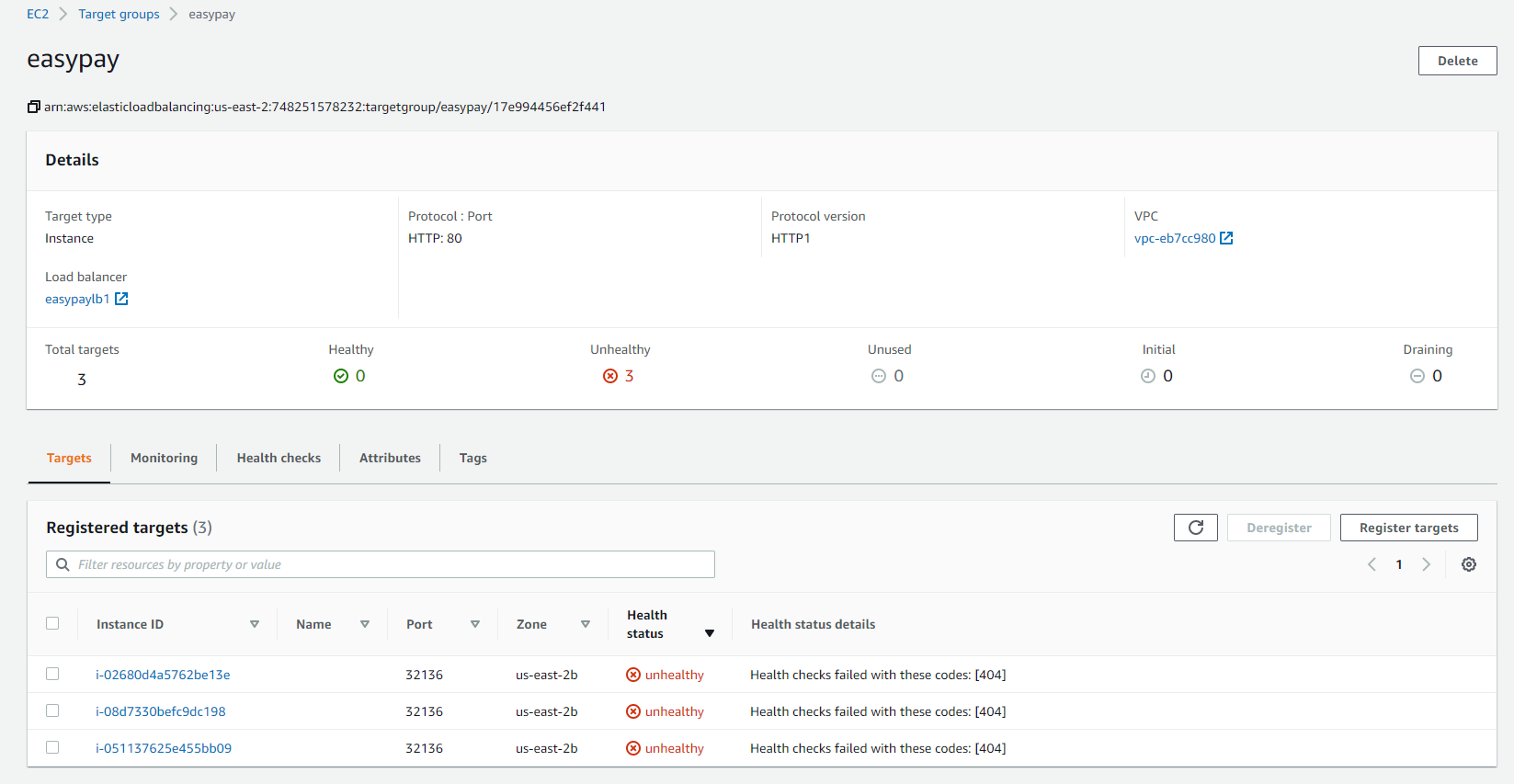
**Step 1** - Log into the AWS console and under ec2 dashboard navigate to the “Load Balancing” section, expand, and click on “Load Balancers”

**Step 2** – Click on “Create Load Balancer” and following the directions and fill out the necessary fields accordingly.



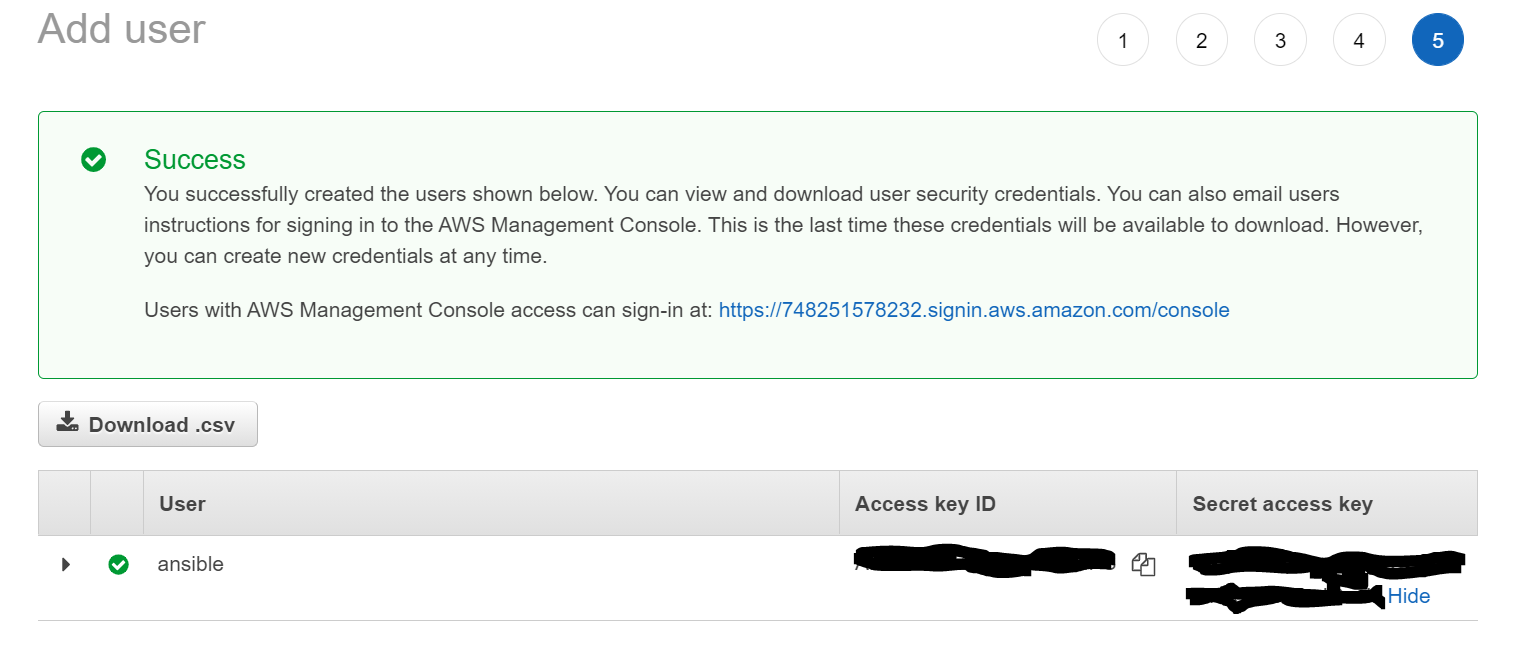
For detail information see Credits and Reference examples at the end, at “Create a Class Load Balancer”.

Set up target group for backend port connection. The target port is determined after the application is deployed in the Kubernetes cluster. Below the target port is “32126”.

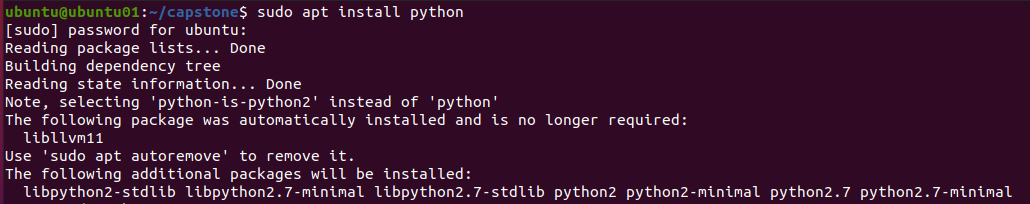


# 2. Automate the provisioning of an EC2 instance using Ansible

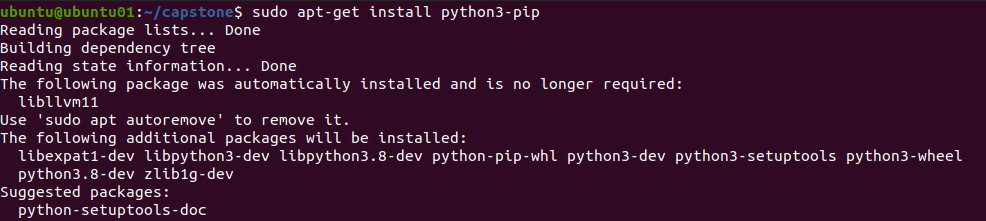
## **Step 1** - Create AWS IAM User for provisioning through anisible



## **Step 2** - Install python



## **Step 3** - Install python3-pip



## **Step 4** - Install ansible



## **Step 5** - Configure ansible “localhost”





## **Step 6** - Create ansible playbook to create ec2 nodes: playbook.html

# AWS playbook

---

- name: Create ec2 instances

hosts: localhost

connection: local

gather\_facts: False

vars:

key\_name: easypay

region: us-east-2

image: ami-0f93b5fd8f220e428

id: "payment-app"

sec\_group: "{{ id }}-sec"

tasks:

- name: Facts

block:

- name: Get instances facts

ec2\_instance\_facts:

aws\_access\_key: "{{ec2\_access\_key}}"

aws\_secret\_key: "{{ec2\_secret\_key}}"

region: "{{ region }}"

register: result

- name: Instances ID

debug:

msg: "ID: {{ item.instance\_id }} - State: {{ item.state.name }} - Public DNS: {{ item.public\_dns\_name }}"

loop: "{{ result.instances }}"

tags: always

- name: Provisioning EC2 instances

block:

- name: Upload public key to AWS

ec2\_key:

name: "{{ key\_name }}"

key\_material: "{{ lookup('file', '/home/ubuntu/.ssh/{{ key\_name }}.pub') }}"

region: "{{ region }}"

aws\_access\_key: "{{ec2\_access\_key}}"

aws\_secret\_key: "{{ec2\_secret\_key}}"

- name: Create security group

ec2\_group:

name: "{{ sec\_group }}"

description: "Sec group for app {{ id }}"

# vpc\_id: 12345

region: "{{ region }}"

aws\_access\_key: "{{ec2\_access\_key}}"

aws\_secret\_key: "{{ec2\_secret\_key}}"

rules:

- proto: tcp

ports:

- 22

cidr\_ip: 0.0.0.0/0

rule\_desc: allow all on ssh port

- proto: tcp

from\_port: 8080

to\_port: 8080

cidr\_ip: 0.0.0.0/0

rule\_desc: allow all on port 8080

- proto: tcp

from\_port: 6443

to\_port: 6443

cidr\_ip: 0.0.0.0/0

rule\_desc: allow all on port 6443

register: result\_sec\_group

- name: Provision instance(s)

ec2:

aws\_access\_key: "{{ec2\_access\_key}}"

aws\_secret\_key: "{{ec2\_secret\_key}}"

key\_name: "{{ key\_name }}"

group\_id: "{{ result\_sec\_group.group\_id }}"

image: "{{ image }}"

instance\_type: t2.medium

region: "{{ region }}"

wait: true

count: 3

# exact\_count: 2

# count\_tag:

# Name: App

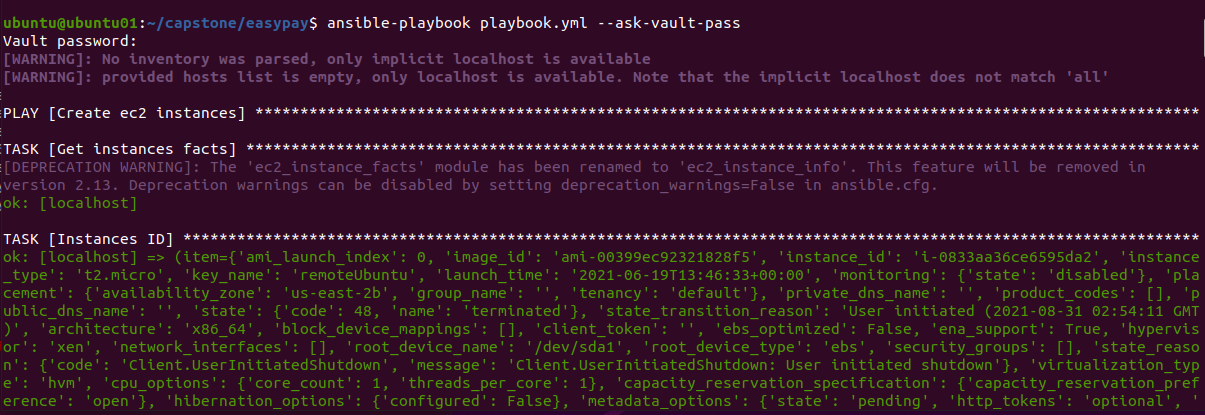
# instance\_tags:

# Name: App

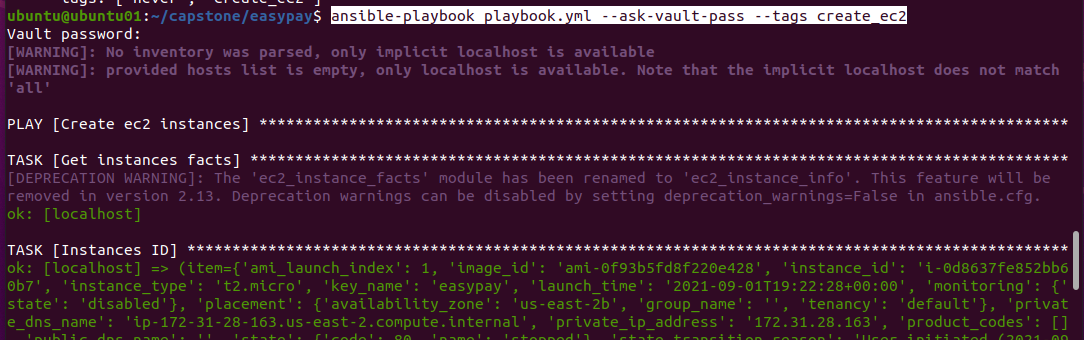
tags: ['never', 'create\_ec2']

## **Step 7** - Run the playbook to get general facts and build ec2 instances

Get facts run: ansible-playbook playbook.yml --ask-vault-pass



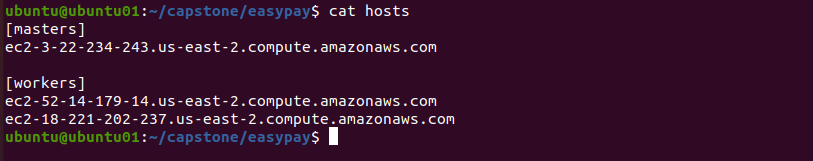
Build the ec2s: ansible-playbook playbook.yml --ask-vault-pass --tags create\_ec2



## **Step 8** - Create ansible host resource file

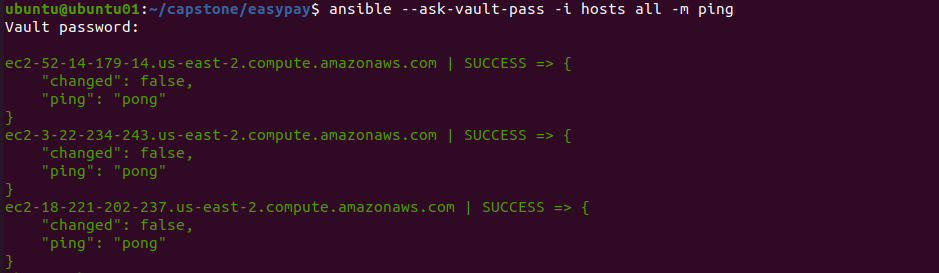
$ ansible-playbook playbook.yml --ask-vault-pass |grep msg | grep DNS | grep ec2 >> hosts

vi hosts



## **Step 9** - Test the connection to ec2 instances

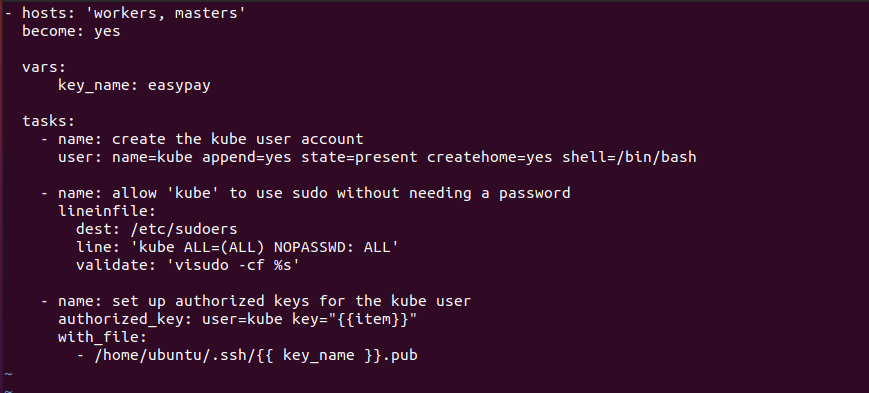
Connect to each hosts using ssh and pub key



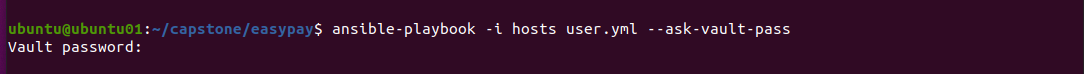
# 3. Install Docker and Kubernetes on the cluster

## **Step 1** – Create Kubernetes user with Ansible

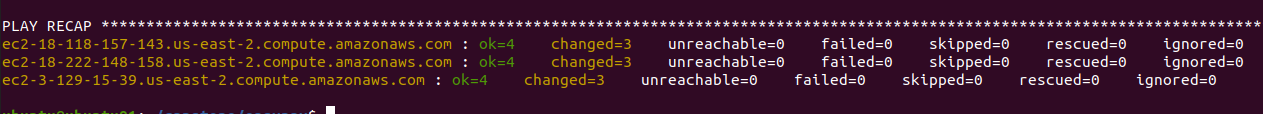
****



ansible-playbook -i hosts user.yml --ask-vault-pass



Should get the following results



## **Step 2** – Install Docker



docker.yml

---

- hosts: "masters, workers"

remote\_user: ubuntu

become: yes

become\_method: sudo

become\_user: root

gather\_facts: yes

connection: ssh

tasks:

- name: patch the environment

shell: |

sudo apt-get -y update && sudo apt-get install

- name: install packages to allow apt to user repository over https

shell: |

sudo apt-get -y install apt-transport-https ca-certificates curl gnupg lsb-release

- name: Add Docker's official GPG key

shell: |

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --batch --yes --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

- name: Set up the stable repository

shell: |

echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

- name: Install Docker Engine and configure cgroup driver to systemd

shell: |

sudo apt-get -y update

sudo apt-get -y install docker-ce docker-ce-cli containerd.io

echo '{"exec-opts": ["native.cgroupdriver=systemd"]}' >> /tmp/daemon.json

sudo mv /tmp/daemon.json /etc/docker/.

sudo chmod 600 /etc/docker/daemon.json

sudo systemctl restart docker

- name: Test docker

shell: |

sudo docker run hello-world

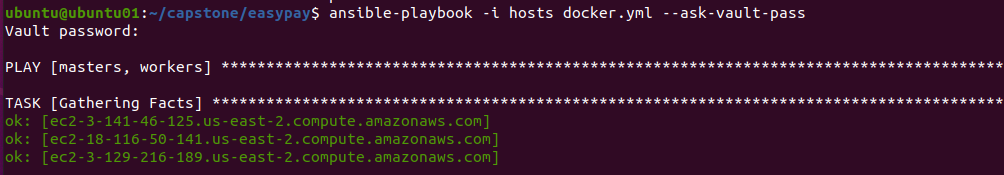
- name: Create Docker group and add user and reboot

shell: |

sudo groupadd docker

sudo usermod -aG docker $USER

ansible-playbook -i hosts docker.yml --ask-vault-pass

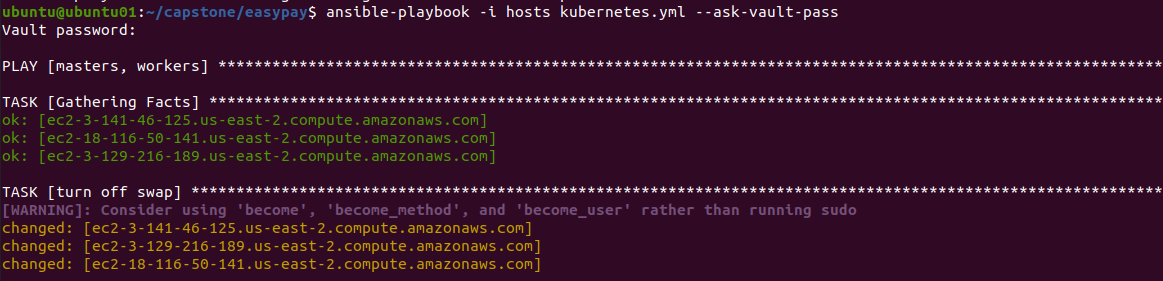


## **Step 3** - Install Kubernetes

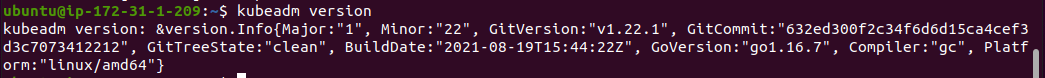
**Step 3.1 Create Kubernetes.yml to install the required dependencies**

Run ansible playbook

ansible-playbook -i hosts kubernetes.yml --ask-vault-pass



SSH to master and check version and node status



**Step 3.2 - Create the master node**

vi createK8sMaster.yml – use the below contents

---

- hosts: masters

remote\_user: ubuntu

gather\_facts: yes

connection: ssh

vars:

calicoVer: v3.20

tasks:

- name: initialize the cluster

shell: sudo kubeadm init --pod-network-cidr=10.244.0.0/16

args:

creates: cluster\_initialized.txt

- name: give non-root users access to use kubeadm

shell: mkdir -p $HOME/.kube

- name: give non-root users access to use kubeadm

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

- name: give non-root users access to use kubeadm

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

- name: install Pod network

shell: sudo kubectl apply -f https://docs.projectcalico.org/{{ calicoVer }}/manifests/calico.yaml

- name: Get the token for joining the worker nodes

shell: kubeadm token create --print-join-command

register: kubernetes\_join\_command

- name: Get join command

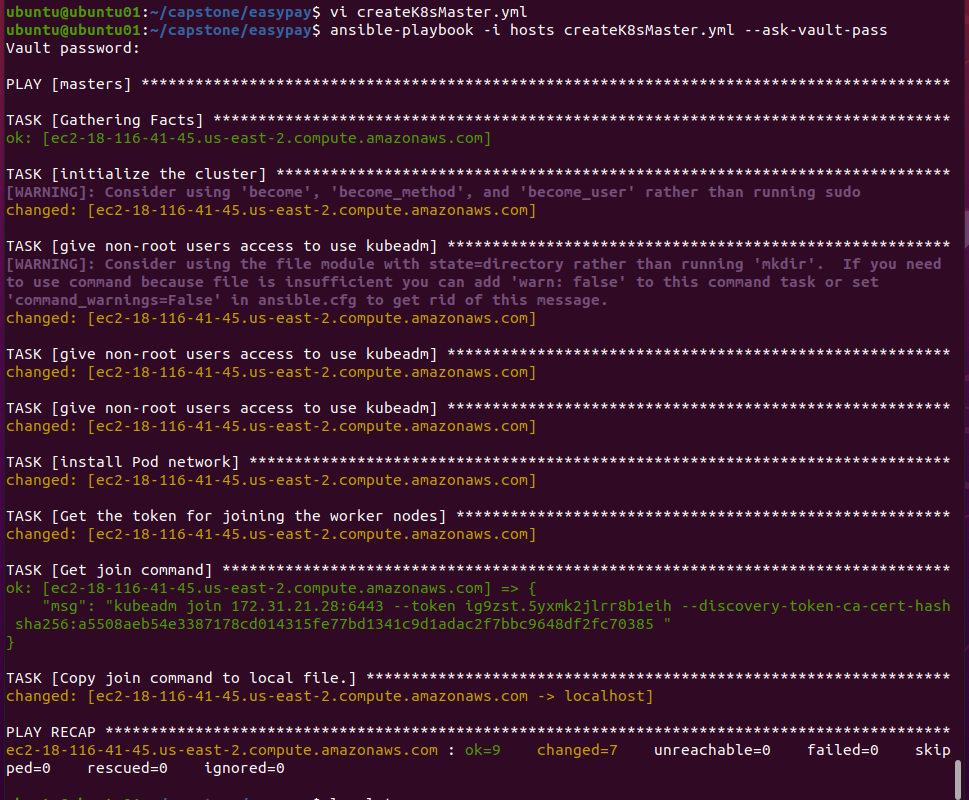
debug:

msg: "{{ kubernetes\_join\_command.stdout }}"

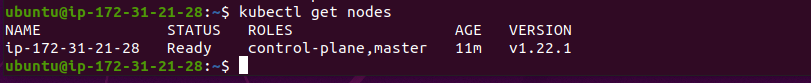
- name: Copy join command to local file.

local\_action: copy content="{{ kubernetes\_join\_command.stdout\_lines[0] }}" dest="/tmp/kubernetes\_join\_command" mode=0777

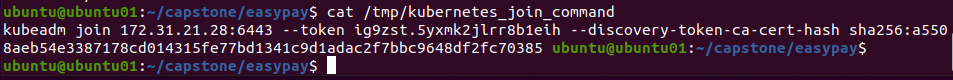
Run the ansible playbook



Log into master and check the nodes



Verify the join command on local ansible host:



**Step 3.3 - Create the worker nodes and join to pod network**

vi joingK8sworkers.yml, use the below contents

---

- hosts: workers

remote\_user: ubuntu

become: yes

become\_method: sudo

become\_user: root

gather\_facts: yes

connection: ssh

tasks:

- name: Copy join command to worker nodes

copy:

src: /tmp/kubernetes\_join\_command

dest: /tmp/kubernetes\_join\_command

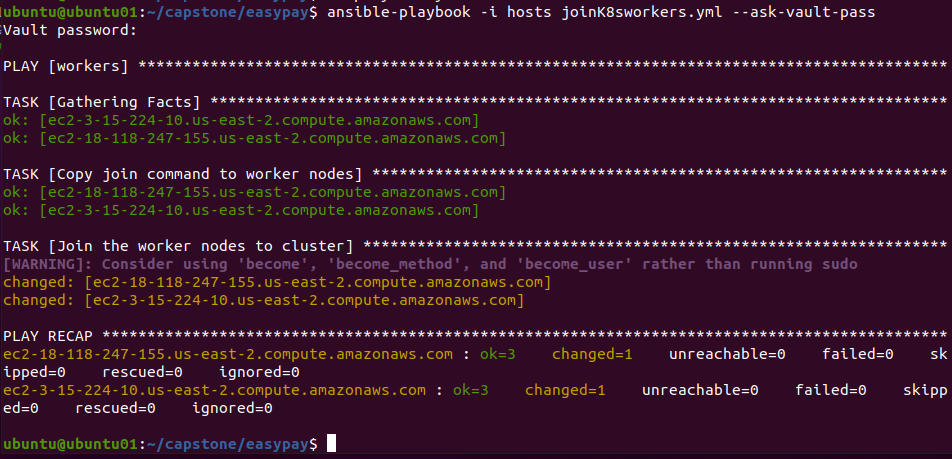
mode : 0777

- name: Join the worker nodes to cluster

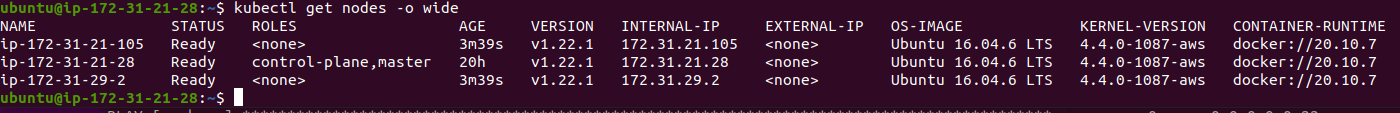
command: sh /tmp/kubernetes\_join\_command

register: joined\_or\_not

Run the playbook



Verify the cluster nodes:



# 4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod

**In this section ingress traffic to the database pod will be handled by a database service called mysql-service.**

## **Step 1** – Create the database

Create a secret: Create a file mysql-secret.yml with these contents:

apiVersion: v1

kind: Secret

metadata:

name: mysql-pass

type: Opaque

data:

password: YWRtaW4=

MySQL Database: Create a file mysql-pod.yml with these contents:

apiVersion: v1

kind: Pod

metadata:

name: mysql

labels:

name: mysql

spec:

containers:

- name: mysql

image: mysql:latest

env:

- name: MYSQL\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-pass

key: password

ports:

- name: mysql

containerPort: 3306

protocol: TCP

volumeMounts:

- name: mysql-storage

mountPath: /var/lib/mysql

volumes:

- name: mysql-storage

emptyDir: {}

Service: Create mysql-service.yml YML file with these contents:

apiVersion: v1

kind: Service

metadata:

name: mysql-service

labels:

name: mysql

spec:

ports:

- port: 3306

selector:

name: mysql

type: ClusterIP

Deploy database: Create a playbook, mysql-createDB.yml with the following contents

---

- hosts: masters

remote\_user: ubuntu

gather\_facts: yes

connection: ssh

vars:

srcdir: "/home/ubuntu/capstone/easypay"

destdir: "/home/ubuntu"

tasks:

- name: Copy secret file

copy:

src: "{{ srcdir }}/mysql-secret.yml"

dest: "{{ destdir }}/mysql-secret.yml"

mode : 0664

- name: Copy pod file

copy:

src: "{{ srcdir }}/mysql-pod.yml"

dest: "{{ destdir }}/mysql-pod.yml"

mode : 0664

- name: Copy service file

copy:

src: "{{ srcdir }}/mysql-service.yml"

dest: "{{ destdir }}/mysql-service.yml"

mode : 0664

- name: Create the secret

command: kubectl create -f mysql-secret.yml

register: result\_secret

- name: Create the pod

command: kubectl create -f mysql-pod.yml

register: result\_pod

- name: Create the service

command: kubectl create -f mysql-service.yml

register: result\_service

- name: Get secret command

debug:

msg: "{{ result\_secret.stdout }}"

- name: Get pod command

debug:

msg: "{{ result\_pod.stdout }}"

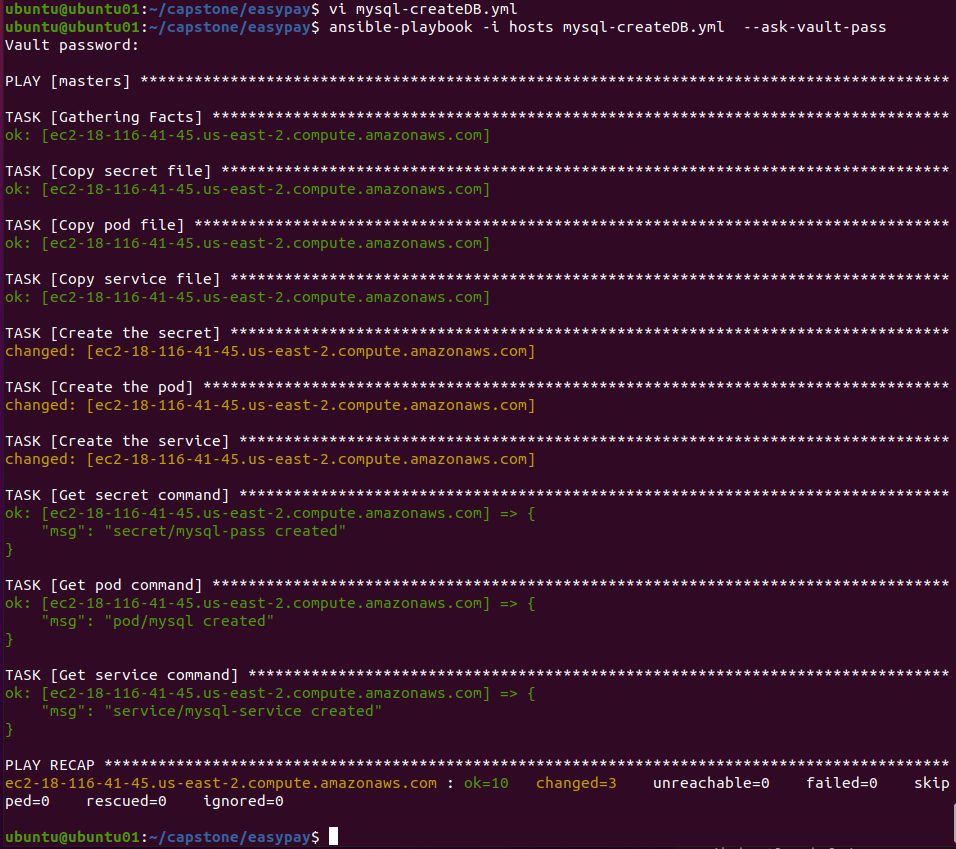
- name: Get service command

debug:

msg: "{{ result\_service.stdout }}"

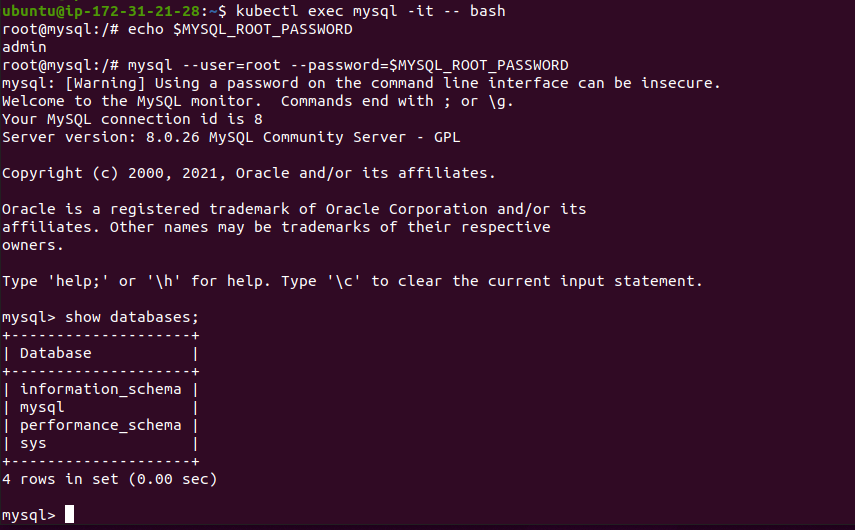
Run the playbook

ansible-playbook -i hosts mysql-createDB.yml --ask-vault-pass



Verify the database is up and available for remote connection:

kubectl exec mysql -it -- bash



How to test pod to pod connection of a mysql database pod. Deploy a mysql client pod and try connecting with the clusterd:

$ kubectl run -it --rm --image=mysql:5.6 --restart=Never mysql-client -- mysql -h 10.100.137.83 -padmin

## **Step 2** – Create the application

Create the myapp-pod.yml to deploy the application

ubuntu@ip-172-31-21-28:~$ cat myapp-pod.yml

apiVersion: v1

kind: Pod

metadata:

name: myapp

labels:

name: myapp

spec:

containers:

- name: msg-api

image: tmarroquin/myapp:latest

ports:

- name: msg-api

containerPort: 8080

Run the following to get the database service IP the application will use:

ubuntu@ip-172-31-21-28:~$ kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 3d5h

mysql-service ClusterIP 10.100.137.83 <none> 3306/TCP 22h

Result the podIP of the mysql database as the host value in the below java script.

Create api.js

var express = require('express')

var mysql = require('mysql')

var Router = express.Router();

var ConnPool = mysql.createPool({

host: '10.100.137.83',

user: 'root',

password: 'admin',

database: 'mysql'

})

// create database and MESSAGE table if not exist

ConnPool.query('CREATE DATABASE IF NOT EXISTS k8smysqldb', function (err) {

if (err) throw Error('\n\t \*\*\*\* error creating database \*\*\*\* ' + err)

console.log('\n\t ==== database k8smysqldb created !! ====')

ConnPool.query('USE k8smysqldb', function (err) {

if (err) throw Error('\n\t \*\*\*\* error using database \*\*\*\* ' + err);

console.log('\n\t ==== database k8smysqldb switched !! ====')

ConnPool.query('CREATE TABLE IF NOT EXISTS messages('

+ 'id INT NOT NULL AUTO\_INCREMENT,'

+ 'PRIMARY KEY(id),'

+ 'text VARCHAR(100)'

+ ')', function (err) {

if (err) throw Error('\n\t \*\*\*\* error creating table \*\*\*\* ' + err);

})

})

})

/\*\*

\* /all

\*/

Router.get('/all', function (req, res) {

ConnPool.getConnection(function (errConn, conn) {

if (errConn) throw Error('error get connection : ' + errConn)

conn.query('SELECT \* FROM messages', function (errSelect, rows) {

if (errSelect) throw Error('error selecting messages : ' + errSelect)

res.writeHead(200, {

'Content-Type': 'application/json'

});

var result = {

success: true,

rows: rows.length,

}

res.write(JSON.stringify(rows));

res.end();

})

})

})

module.exports = Router

Create server.js

var express = require('express')

var msgApi = require('./api')

var app = express()

app.use('/msg-api', msgApi)

app.get('/ping', function (req, res) {

res.write("hello there! I m up and running!");

res.end();

})

app.listen(8080, function () {

console.log('\n\t ==== Message API listening on 8080! ====')

})

Create Docker file

FROM node:latest

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY package.json /usr/src/app/package.json

RUN npm i

COPY . /usr/src/app/

EXPOSE 8080

CMD [ "node", "server.js" ]

Create package.json

$ npm set init.author.email "trinidad.marroquin@gmail.com"

$ npm set init.author.name "Trinidad Marroquin, Jr."

$ npm set init.license "GNU General Public License"

$ npm init --yes

Wrote to /home/master/docker/SimplilearnCapstoneProject/package.json:

{

"name": "SimplilearnCapstoneProject",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"repository": {

"type": "git",

"url": "git+https://github.com/trinidadgithub/SimplilearnCapstoneProject.git"

},

"keywords": [],

"author": "Trinidad Marroquin, Jr. <trinidad.marroquin@gmail.com>",

"license": "GNU General Public License",

"bugs": {

"url": "https://github.com/trinidadgithub/SimplilearnCapstoneProject/issues"

},

"homepage": "https://github.com/trinidadgithub/SimplilearnCapstoneProject#readme"

}

# Move package.json to appropriate build directory for docker

$ ls -lrt

total 16

-rw-rw-r-- 1 master master 310 Sep 5 12:11 server.js

-rw-rw-r-- 1 master master 183 Sep 5 12:11 Dockerfile

-rw-rw-r-- 1 master master 1377 Sep 5 12:11 api.js

-rw-rw-r-- 1 master master 624 Sep 5 12:31 package.json

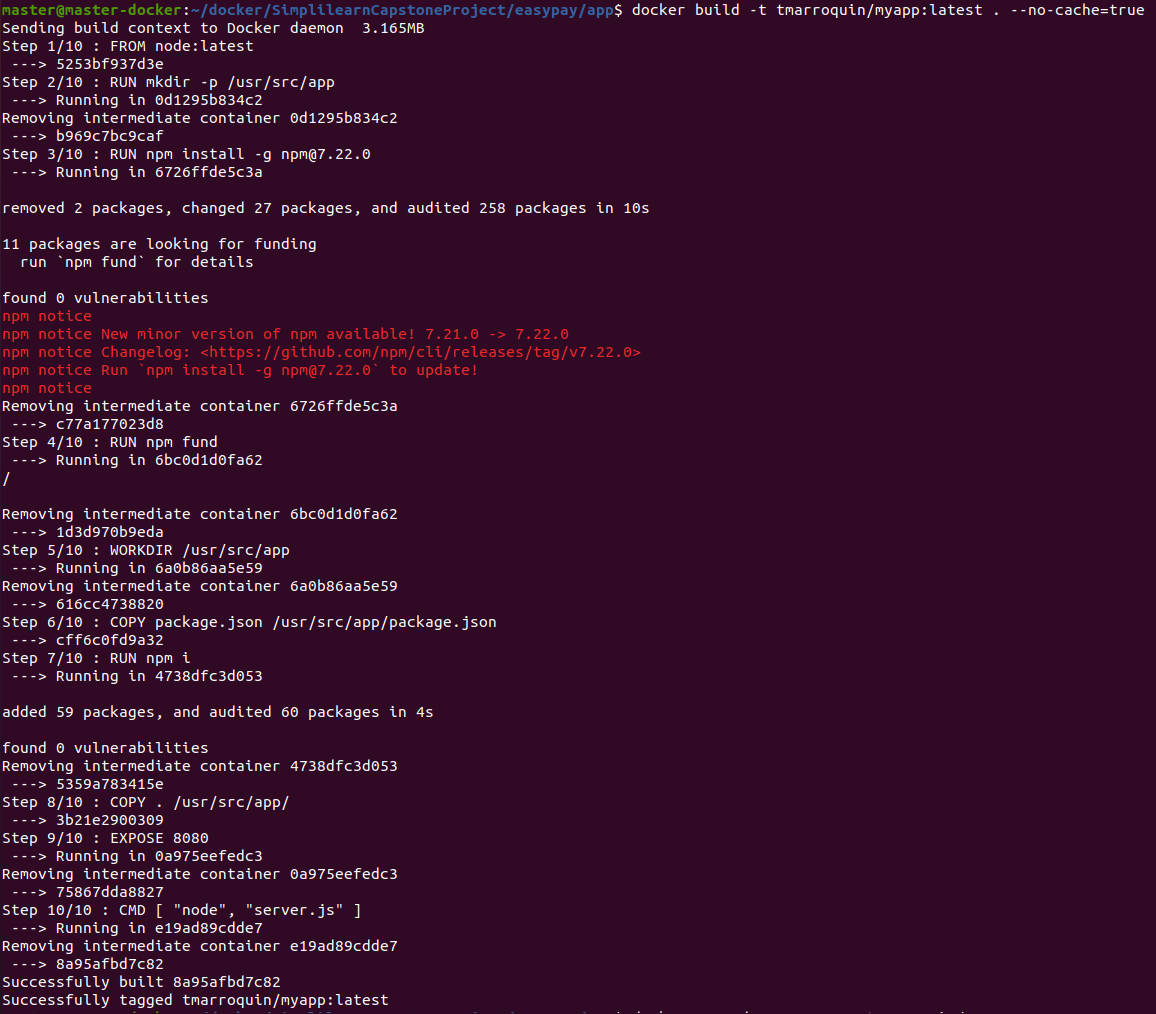
Install npm express and mysql module dependencies

$ npm install --save express

$ npm install --save mysql

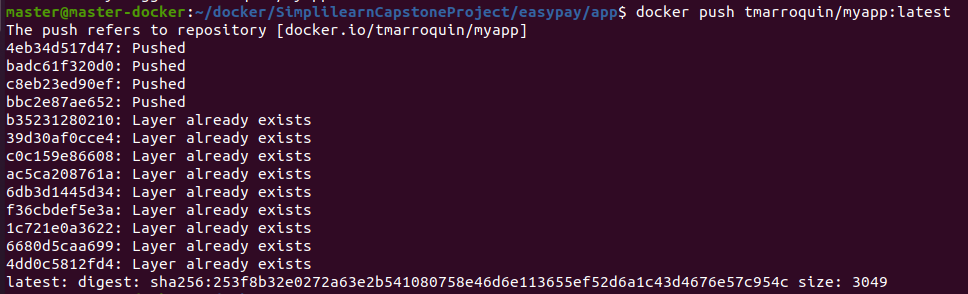
Build the application

$ docker build -t tmarroquin/myapp:latest . --no-cache=true



Push new image to docker hub.

$ docker push tmarroquin/myapp:latest



Deploy the application to kubernetes and test

$ ansible-playbook -i hosts myapp-create.yml --ask-vault-pass

$ kubectl logs myapp

==== Message API listening on 8080! ====

==== database k8smysqldb created !! ====

==== database k8smysqldb switched !! ====

$ kubectl describe service myapp-service

Name: myapp-service

Namespace: default

Labels: name=myapp

Annotations: <none>

Selector: name=myapp

Type: NodePort

IP Family Policy: SingleStack

IP Families: IPv4

IP: 10.98.181.25

IPs: 10.98.181.25

Port: <unset> 8080/TCP

TargetPort: 8080/TCP

NodePort: <unset> 32136/TCP

Endpoints: 10.244.138.217:8080

Session Affinity: None

External Traffic Policy: Cluster

Events: <none>

$ hostname -i

172.31.21.28

$ curl 172.31.21.28:32136/ping

hello there! I m up and running!

$ curl 172.31.21.38:32136/msg-api/all

[{"id":1,"text":"this is the first msg!"},{"id":2,"text":"this is the second msg!"}]

Database records were manually entered into the database.

ubuntu@ip-172-31-21-28:~$ kubectl exec mysql -it -- bash

root@mysql:/# use k8smysqldb;

bash: use: command not found

root@mysql:/# mysql --user=root --password=$MYSQL\_ROOT\_PASSWORD

mysql: [Warning] Using a password on the command line interface can be insecure.

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 39

Server version: 8.0.26 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use k8smysqldb;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show tables;

+----------------------+

| Tables\_in\_k8smysqldb |

+----------------------+

| messages |

+----------------------+

1 row in set (0.00 sec)

mysql> insert into messages(text) values ('this is the first msg!');

Query OK, 1 row affected (0.01 sec)

mysql> insert into messages(text) values ('this is the second msg!');

Query OK, 1 row affected (0.01 sec)

mysql> select \* from messages

-> ;

+----+-------------------------+

| id | text |

+----+-------------------------+

| 1 | this is the first msg! |

| 2 | this is the second msg! |

+----+-------------------------+

2 rows in set (0.00 sec)

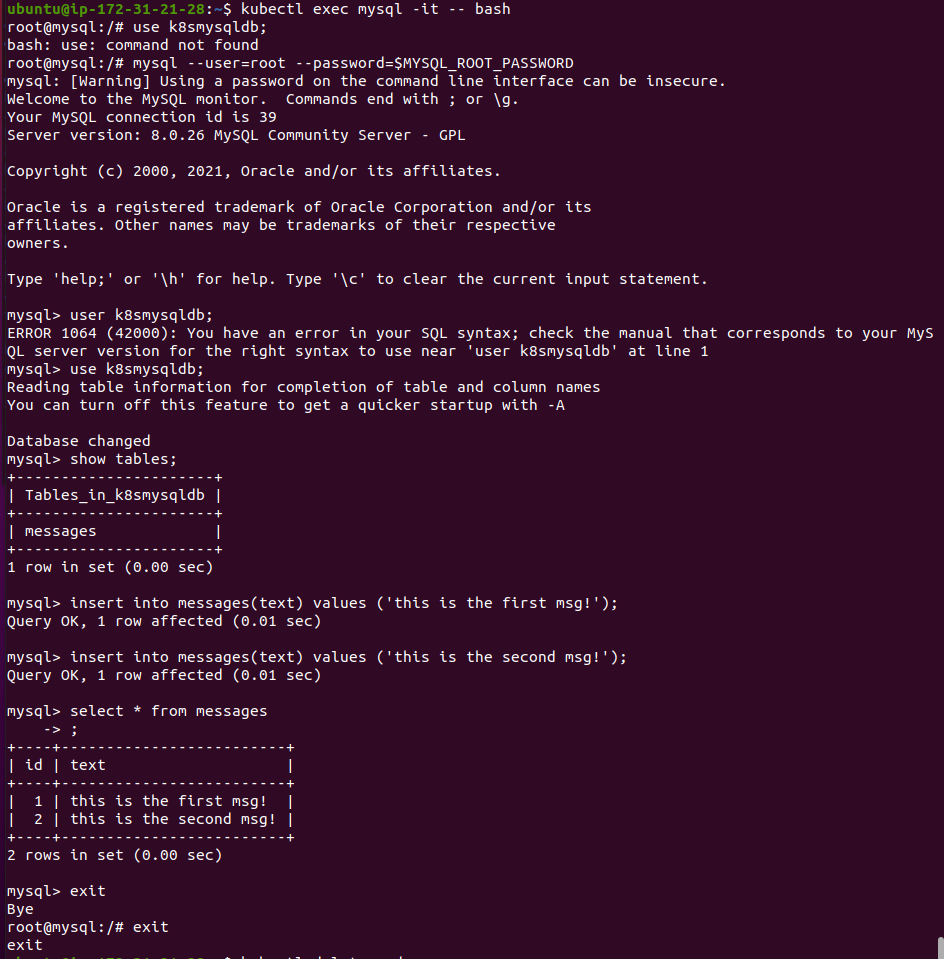
mysql> exit

Bye

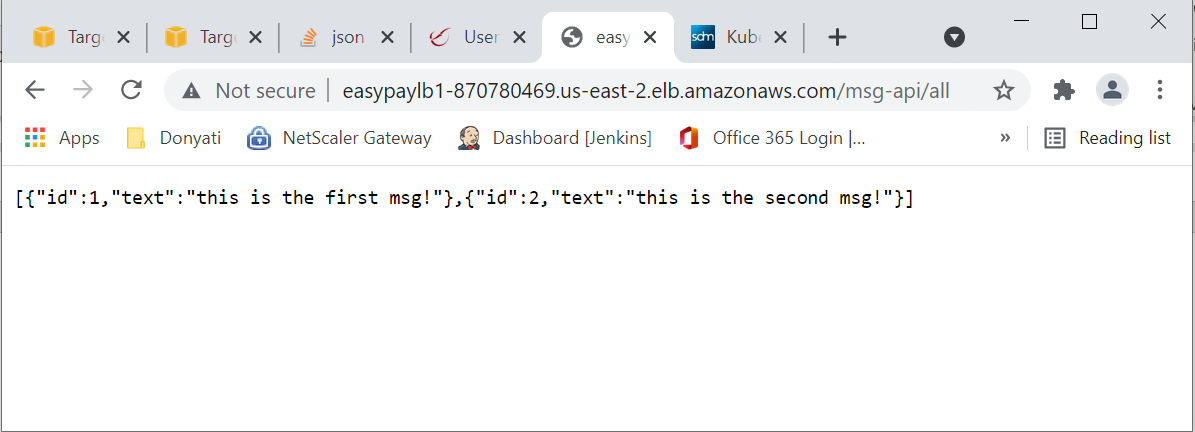
root@mysql:/# exit

exit

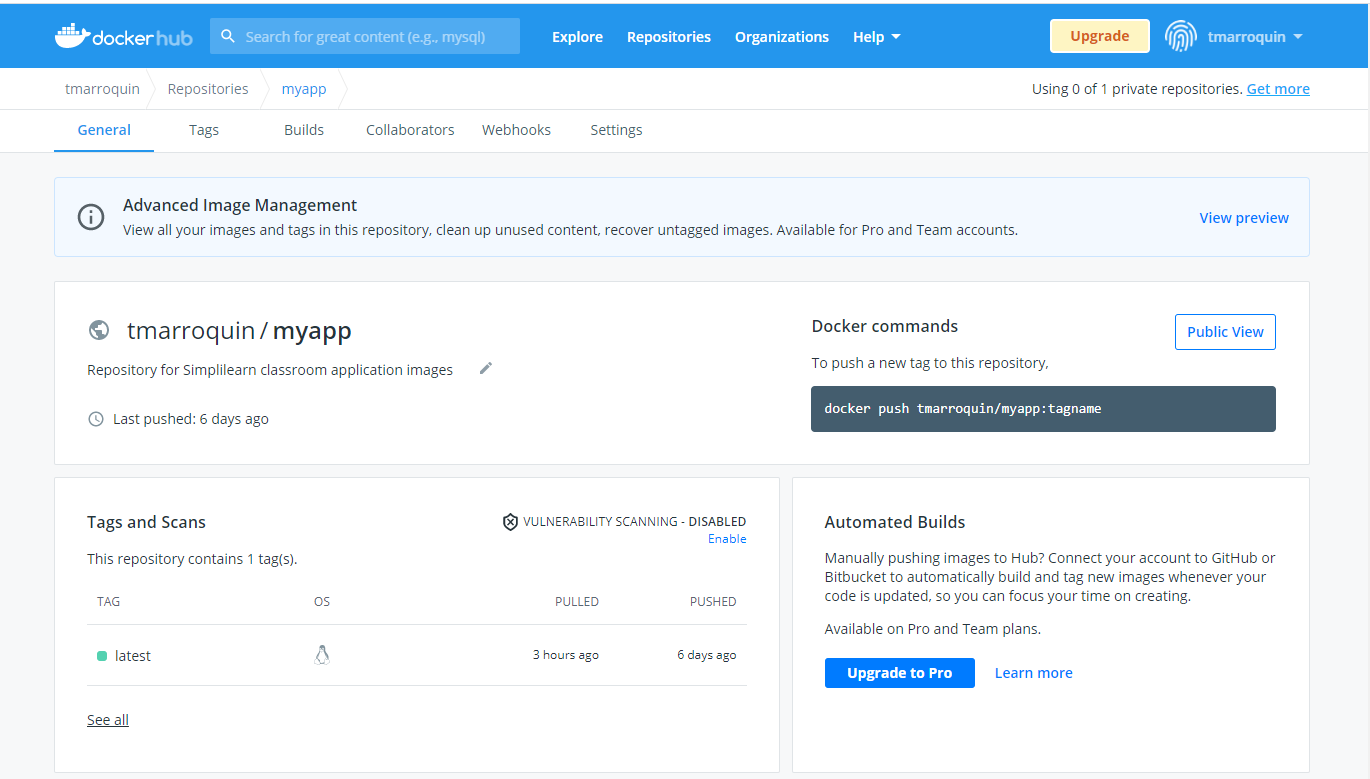
ubuntu@ip-172-31-21-28:~$



Test the application through AWS load balancer created in the beginning, Section 1, Create ec2 cluster….



Push the image to Docker hub



# 5. Create a new user with permissions to create, list, get, update, and delete pods

See “Install Kubernetes and Docker on the cluster, Step – 1 Create New Kubernetes with ansible” for creating a new Kubernetes user. In that section, created the “kube” unix account on master kubenetes server.

**Step 1** – Create the kubernetes service account, cluster role with allowed access, and the role binding definitions in a YAML file name easypay-role.yml.

apiVersion: v1

kind: ServiceAccount

metadata:

name: sa-clusteradmin

namespace: default

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: cr-maintainer

rules:

- verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]

resources: ["pods"]

apiGroups: [""]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: rb-maintainer

namespace: default

subjects:

- kind: ServiceAccount

name: sa-clusteradmin

namespace: default

roleRef:

kind: ClusterRole

name: cr-maintainer

apiGroup: rbac.authorization.k8s.io

**Step 2** – Run the yaml

# On kube master

ubuntu@ip-172-31-21-28:~$ kubectl create -f easypay-role.yml

serviceaccount/sa-clusteradmin created

clusterrole.rbac.authorization.k8s.io/cr-maintainer created

rolebinding.rbac.authorization.k8s.io/rb-maintainer created

**Step 3** – Verify the results

# On kube master

ubuntu@ip-172-31-21-28:~$ !712

kubectl get serviceaccount

NAME SECRETS AGE

default 1 6d17h

sa-clusteradmin 1 38h

ubuntu@ip-172-31-21-28:~$ kubectl get clusterrole | grep maintainer

cr-maintainer 2021-09-09T03:04:55Z

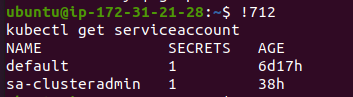
ubuntu@ip-172-31-21-28:~$ !710

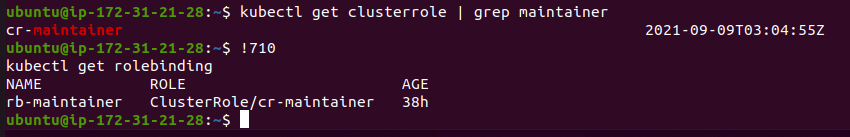
kubectl get rolebinding

NAME ROLE AGE

rb-maintainer ClusterRole/cr-maintainer 38h

Screen shots:





**Step 4** – Configure kube account access

ubuntu@ip-172-31-21-28:~$ sudo su - kube

kube@ip-172-31-21-28:~$ TOKEN=$(kubectl describe -n default secrets "$(kubectl describe -n default serviceaccount sa-clusteradmin | grep -i Tokens | awk '{print $2}')" | grep token: | awk '{print $2}')

kube@ip-172-31-21-28:~$ kubectl config set-credentials cluster-admin-user --token=$TOKEN

User "cluster-admin-user" set.

kube@ip-172-31-21-28:~$ kubectl config set-context cluster-admin --cluster=kubernetes --user=cluster-admin-user

Context "cluster-admin" created.

kube@ip-172-31-21-28:~$ export KUBECONFIG=~/.kube/kube-cluster-admin.kubeconfig

kube@ip-172-31-21-28:~$ kubectl config use-context cluster-admin

Switched to context "cluster-admin".

kube@ip-172-31-21-28:~$ kubectl auth can-i get pods

yes

kube@ip-172-31-21-28:~$ kubectl auth can-i create pods

yes

kube@ip-172-31-21-28:~$ kubectl auth can-i delete pods

yes

kube@ip-172-31-21-28:~$ kubectl auth can-i list pods

yes

#Continued

kube@ip-172-31-21-28:~$ kubectl auth can-i get pods --all-namespaces

no

kube@ip-172-31-21-28:~$ kubectl auth can-i create pods --all-namespaces

no

kube@ip-172-31-21-28:~$ kubectl auth can-i delete pods --all-namespaces

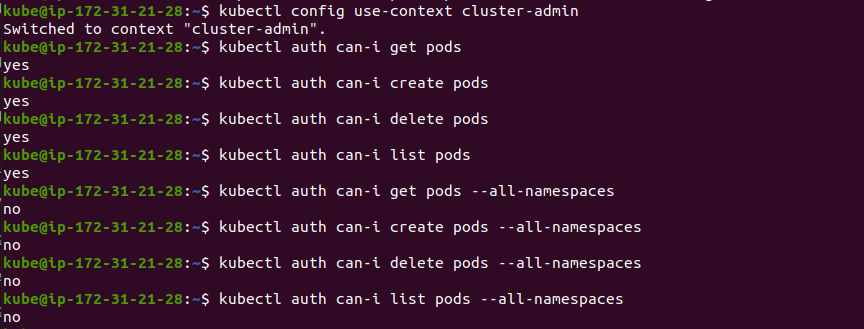
no

kube@ip-172-31-21-28:~$ kubectl auth can-i list pods --all-namespaces

no

kube@ip-172-31-21-28:~$

# Screen Capture



# 6. Configure application on the pod

See “Step – 2 Create the application” in Chapter 4, “Implement the network policies at the database pod to allow ingress traffic from the front-end application pod “ for application configuration.

# 7. Take snapshot of ETCD database

#On the master server

ubuntu@ip-172-31-21-28:~$ sudo docker run --rm -v $(pwd)/backup:/backup \

--network host \

-v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd \

--env ETCDCTL\_API=3 \

k8s.gcr.io/etcd:3.4.3-0 \

etcdctl --endpoints=https://127.0.0.1:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \

--key=/etc/kubernetes/pki/etcd/healthcheck-client.key \

snapshot save /backup/etcd-snapshot-latest.db

ubuntu@ip-172-31-21-28:~$ ls -lrt

total 28

drwxrwxr-x 2 ubuntu ubuntu 4096 Sep 7 18:48 bak

-rw-rw-r-- 1 ubuntu ubuntu 205 Sep 7 19:42 myapp-pod.yml

-rw-rw-r-- 1 ubuntu ubuntu 162 Sep 7 19:42 myapp-service.yml

-rw-rw-r-- 1 ubuntu ubuntu 591 Sep 9 02:47 easypay-role.yml

-rwx------ 1 ubuntu ubuntu 177 Sep 10 20:29 setetcdenv.sh

-rwx------ 1 ubuntu ubuntu 332 Sep 10 20:30 installetcd.sh

drwxr-xr-x 2 root root 4096 Sep 10 21:03 backup

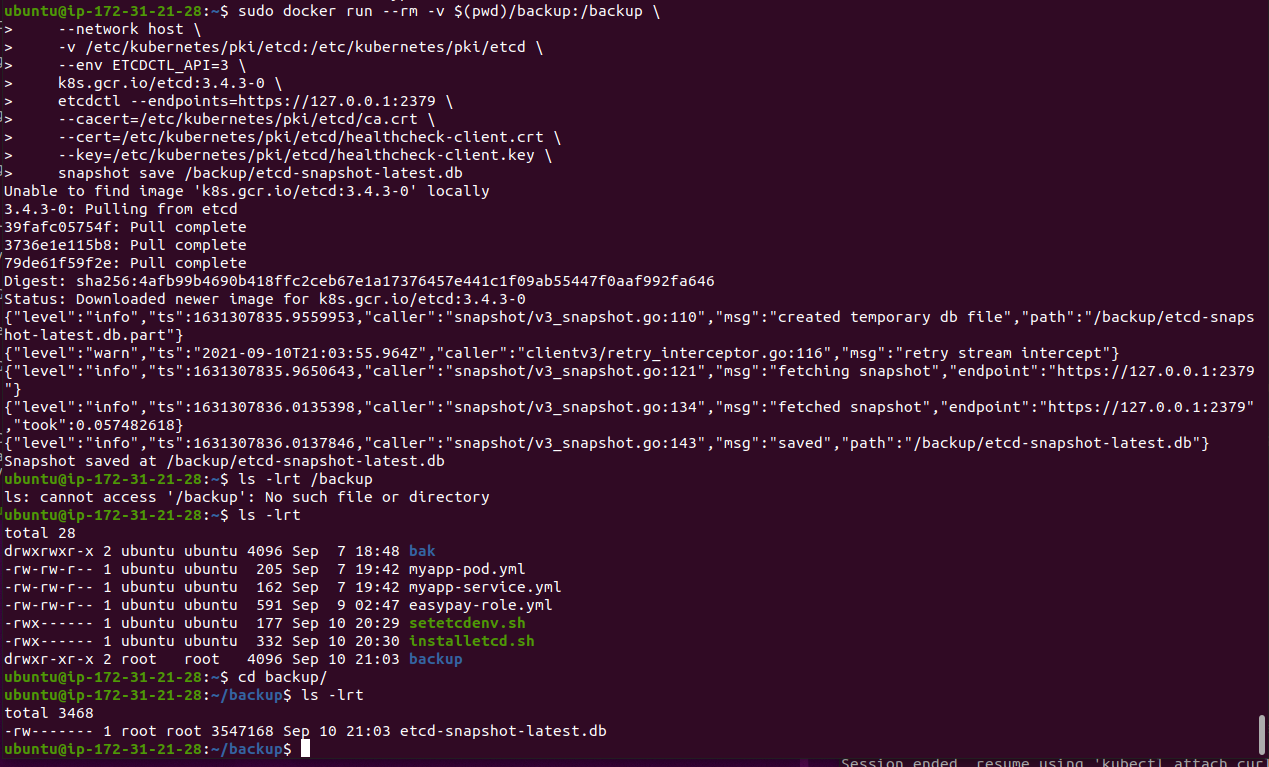
ubuntu@ip-172-31-21-28:~$ cd backup/

ubuntu@ip-172-31-21-28:~/backup$ ls -lrt

total 3468

-rw------- 1 root root 3547168 Sep 10 21:03 etcd-snapshot-latest.db

# Screen Capture



# 8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured

For this objective, modifying the “myapp” deployment from a single node to a replicated set requires updating the process. First create a “deployment” YML, create a new “service” yml, and then turn on autoscaling.

Step 1 – Create Replica set for testing

ubuntu@ip-172-31-21-28:~$ cat myapp-relicaset.yml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myapp-rs

labels:

replicaset: myapp-rs

type: front-end

app: myapp

spec:

template:

metadata:

name: myapp-pod

labels:

name: myapp-pod

app: myapp

type: myapp-front-end

spec:

containers:

- name: msg-api

image: tmarroquin/myapp:latest

ports:

- name: msg-api

containerPort: 8080

replicas: 1

selector:

matchLabels:

type: myapp-front-end

Step 2: Create update service definition for testing

ubuntu@ip-172-31-21-28:~$ vi myapp-service-v1.yml

apiVersion: v1

kind: Service

metadata:

name: myapp-service

labels:

name: myapp-service

app: myapp

type: myapp-front-end

spec:

ports:

- protocol: TCP

port: 8080

nodePort: 32136

selector:

type: myapp-front-end

type: NodePort

Step 3: Create the myapp deployment definition

ubuntu@ip-172-31-21-28:~$ vi myapp-service-v1.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

labels:

replicaset: myapp-rs

type: front-end

app: myapp

spec:

template:

metadata:

name: myapp-pod

labels:

name: myapp-pod

app: myapp

type: myapp-front-end

spec:

containers:

- name: msg-api

image: tmarroquin/myapp:latest

ports:

- name: msg-api

containerPort: 8080

replicas: 1

selector:

matchLabels:

type: myapp-front-end

---

apiVersion: v1

kind: Service

metadata:

name: myapp-service

labels:

name: myapp-service

app: myapp

type: myapp-front-end

spec:

ports:

- protocol: TCP

port: 8080

nodePort: 32136

selector:

type: myapp-front-end

type: NodePort

Step 4: Run the deployment and verify the application

ubuntu@ip-172-31-21-28:~$ kubectl create -f myapp-deployment.yml

deployment.apps/myapp-deployment created

service/myapp-service created

ubuntu@ip-172-31-21-28:~$ vi myapp-deployment.yml

ubuntu@ip-172-31-21-28:~$ kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 10d

myapp-service NodePort 10.102.106.237 <none> 8080:32136/TCP 37s

mysql-service ClusterIP 10.100.137.83 <none> 3306/TCP 7d17h

ubuntu@ip-172-31-21-28:~$ kubectl describe service myapp-service

Name: myapp-service

Namespace: default

Labels: app=myapp

name=myapp-service

type=myapp-front-end

Annotations: <none>

Selector: type=myapp-front-end

Type: NodePort

IP Family Policy: SingleStack

IP Families: IPv4

IP: 10.102.106.237

IPs: 10.102.106.237

Port: <unset> 8080/TCP

TargetPort: 8080/TCP

NodePort: <unset> 32136/TCP

Endpoints: 10.244.138.245:8080

Session Affinity: None

External Traffic Policy: Cluster

Events: <none>

ubuntu@ip-172-31-21-28:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

myapp-deployment-6f94db8f9-t9rb8 1/1 Running 0 60s

mysql 1/1 Running 0 7d17h

ubuntu@ip-172-31-21-28:~$ kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

myapp-deployment-6f94db8f9-t9rb8 1/1 Running 0 66s 10.244.138.245 ip-172-31-29-2 <none> <none>

mysql 1/1 Running 0 7d17h 10.244.139.132 ip-172-31-21-105 <none> <none>

ubuntu@ip-172-31-21-28:~$ curl http://172.31.29.2:32136/ping

hello there! I m up and running!

ubuntu@ip-172-31-21-28:~$

Step 5: Implement auto scaling and get the details

ubuntu@ip-172-31-21-28:~$ kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

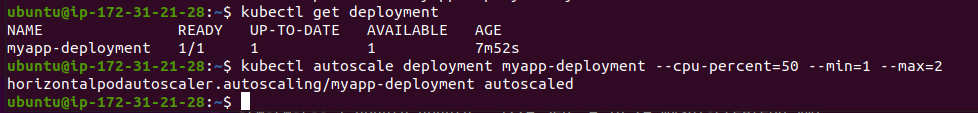
myapp-deployment 1/1 1 1 7m52s

ubuntu@ip-172-31-21-28:~$ kubectl autoscale deployment myapp-deployment --cpu-percent=50 --min=1 --max=2

horizontalpodautoscaler.autoscaling/myapp-deployment autoscaled

ubuntu@ip-172-31-21-28:~$

# Screen Capture



# Check the current status of the auto scaler

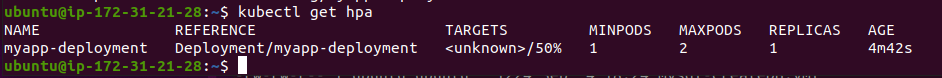
ubuntu@ip-172-31-21-28:~$ kubectl get hpa

NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE

myapp-deployment Deployment/myapp-deployment <unknown>/50% 1 2 1 4m42s

ubuntu@ip-172-31-21-28:~$

# Screen Capture



# 9. Project Repositories, Credits and Reference Examples

Project Repositories

**Simplilearn Devops Training course:** [**https://www.simplilearn.com**](https://www.simplilearn.com)

**Github Repository:** [**https://github.com/trinidadgithub/SimplilearnCapstoneProject.git**](https://github.com/trinidadgithub/SimplilearnCapstoneProject.git)

**Docker Hub:** [**https://hub.docker.com/repository/docker/tmarroquin/myapp**](https://hub.docker.com/repository/docker/tmarroquin/myapp)

Credits and Reference Examples

Create a Classic Load Balancer: <https://docs.aws.amazon.com/elasticloadbalancing/latest/cloassic/elb-getting-started.html>

Using ansible to create ec2 instances - <https://medium.datadriveninvestor.com/devops-using-ansible-to-provision-aws-ec2-instances-3d70a1cb155f>

Using ansible to create k8s cluster - <https://buildvirtual.net/deploy-a-kubernetes-cluster-using-ansible/>

Using ansible to create load balancer - <https://www.nickhammond.com/load-balancing-with-aws-and-ansible/>

Setting up node js application with default package.json - <https://docs.npmjs.com/creating-a-package-json-file>

How to run Kubernetes curl pod - <https://stackoverflow.com/questions/34601650/how-do-i-run-curl-command-from-within-a-kubernetes-pod>

Create user in Kubernetes and implement RBAC - <https://stackoverflow.com/questions/44948483/create-user-in-kubernetes-for-kubectl>

How to take Kubernetes backups - <https://elastisys.com/backup-kubernetes-how-and-why/>

Kubernetes Autoscaling walkthrough - <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>