A) Installing resource or services on aws using Terraform :

1- First install terraform from here : https://learn.hashicorp.com/terraform/getting-started/install.html

2- Create resource (to show how to create any resource on aws using terraform)
Using terraform to create infrastructure on AWS using code. In this code, we are trying to create a below items:

a- VPC
b- Subnet inside VPC
c- Internet gateway associated with VPC
d- Route Table inside VPC with a route that directs internet-bound traffic to the internet gateway
e- Route table association with our subnet to make it a public subnet
f- Security group inside VPC for the public instance, for allowing traffic from public network
g- Key pair used for SSH access
h- EC2 instance inside our public subnet with an associated security group and generated a key pair
i- A private subnet where we will provision our private EC2 instance
j- A Security group for the private instance, to allow traffic only from the public instance
k- A NAT Gateway in the public subnet, for allowing the private instance to have internet connection
l- An elastic ip for the NAT Gateway
m- A route table to link the NAT Gateway between the private subnet
n- Private EC2 instance, where we can run a database our a private application

Using terraform we are able to make a immutable infrastructure which can be destroyed and created using single command.

Commands used:

terraform init : Initialize a Terraform working directory
terraform plan : Generate and show an execution plan
terraform apply : Builds or changes infrastructure
terraform destroy : Destroy Terraform-managed infrastructure

- All files on folder name “Terrraform-Test”

B) Create cluster and worker using vagrant and ansible

1- First install vagrant from here : https://www.vagrantup.com/downloads

2- Config to work with aws use command on powershell

```
# vagrant plugin install vagrant-aws
# vagrant up --provider=aws
# vagrant box add dummy https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box
```

install kubectl on windows

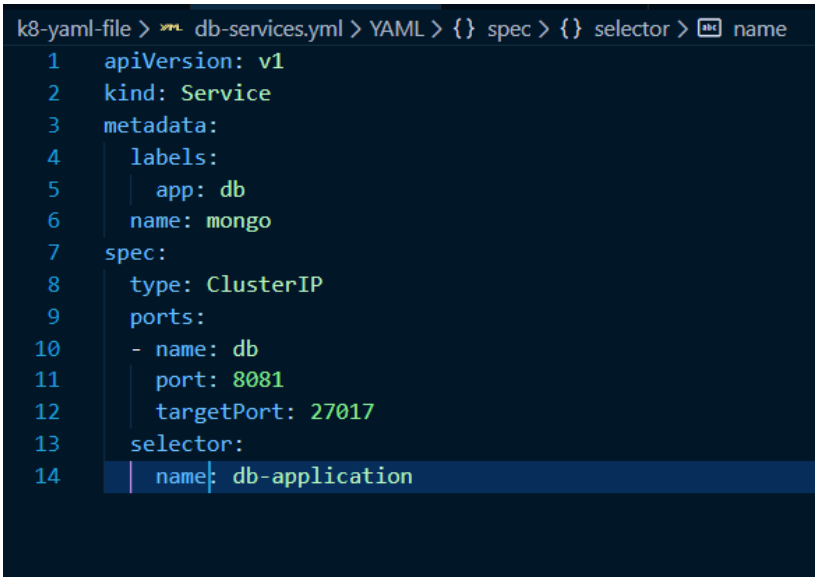
To work on the vagrant-aws plugin, clone this repository out, and use Bundler to get the dependencies:

```
# bundle
# bundle exec rake
# bundle exec vagrant up --provider=aws
```

- All file on folder name “Kubernetes-demo”

Setup Containers :

a- First create services for mongo db and create name and port for that to use on app before build application and create image



There is tow type of services one Cluster IP and Node Port (Cluster IP= this if I use can access inside cluster and can’t access outside cluster . Node Port = this to access services outside of cluster and use it for app and cluster IP for db)

This services for app and this is Node Port

k8-yaml-file > app-services.yml > YAML > {} spec > [ ] ports

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app-servive
spec:
  type: NodePort
  selector:
    app: myapp-deploy-lable
    tier: frontend
  ports:
      # By default and for convenience, the `targetPort` is set to the same value as the `port` field.
    - port: 8082
    # this port 8080 get form docker file
      targetPort: 8080
      # Optional field
      # By default and for convenience, the Kubernetes control plane will allocate a port from a range (default: 30000-32767)
      nodePort: 30007

```

b- Change db url On code

coding-task-devops > app > config > JS database.js > [@] <unknown> > mongoURI

```javascript
module.exports = {
    mongoURI: 'mongodb://mongo:8081'
};

```

c- Create env on pods file and call env and value for variable

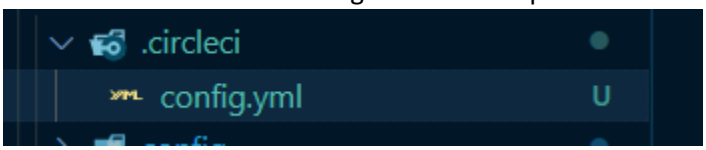
```yaml
spec:
  containers:
  - name: APP-F-Container
    image: mmamoon/demo-test:1.0
    ports:
      - containerPort: 8080
    env:
      - name: mongoURI
        value: mongodb://mongo:8081
        #also can i ues configMap to put all variable in file and call here from file
```

d- Create dockerfile for build app

coding-task-devops > app > Dockerfile > ...

```
1   FROM node:10-alpine
2
3   WORKDIR /usr/src/app
4
5   COPY package*.json ./
6
7   RUN npm install
8
9   ENV mongoURI mongodb://mongo:8081
10
11  COPY . .
12
13  EXPOSE 8080
14
15  CMD [ "npm", "start" ]
```

e- Also use circle CI to build image and add script like this



In this file yaml I write code to explain what code should run when build image like this

```
#https://circleci.com/docs/2.0/deployment-
examples/index.html  >> this for example to deploy app i search here
#https://circleci.com/docs/2.0/configuration-reference/ also this page help me
#https://circleci.com/docs/2.0/databases/ this to build custome db is very easy
```

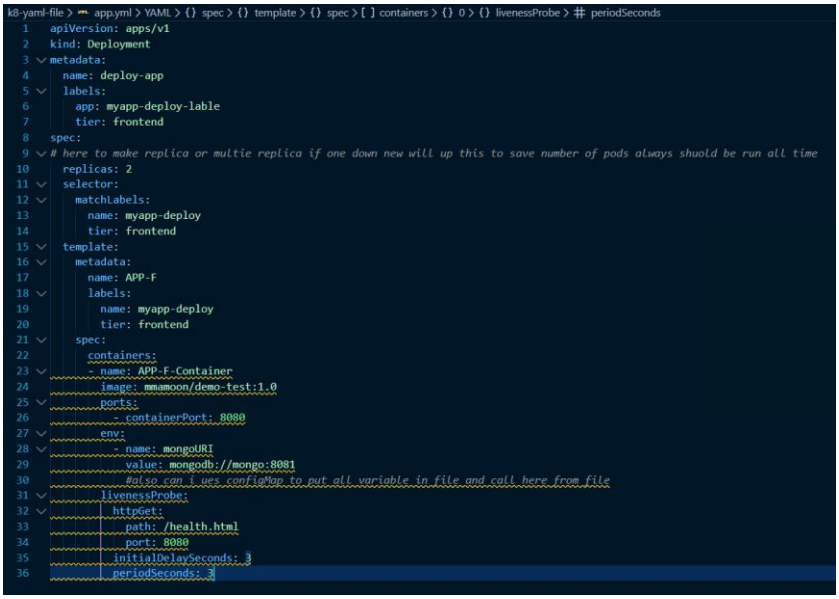
```yaml
steps:
  - checkout
  - setup_remote_docker
  - restore_cache:
      keys:
      - v1-dependencies-{{ checksum "package.json" }}
      - v1-dependencies-
  - run:
      name: Install node packages
      command: |
        npm install

  - run:
      name: Start app
      command: |
        npm start &
  - run:
      name: Run tests
      command: |
        npm test
  - restore_cache:
      keys:
      - v1-dependencies-{{ checksum "package.json" }}
      - v1-dependencies-
  - save_cache:
      paths:
        - node_modules
      key: v1-dependencies-{{ checksum "package.json" }}
```

f- Use these command to build and push image on docker hub:
docker build -t demo-test .
This command to build app on current directory
#docker tag demo-test mmamoon/demo-test:1.0
This command to tag image and version it every change on code should be build and make tag "Mmamoon/" this my repo on docker hub if use aws should put url for repo on aws

#docker push mmamoon/demo-test:latest
This command to push image on docker hub

((
First I test image create single pod to see if work or not I use this command
kubectl run test --image=mmamoon/demo-test --restart=Never
And then create port-forward to see pod on browser using this command
#kubectl port-forward pods/test 80
))

g- Create File yaml for app and use replica set and services for app if want external access



This to create pods should use this command

kubectl create -f “path of file”

- I add env variable on file to mention url for db

```
  env:
    - name: mongoURI
      value: mongodb://mongo:8081
      #also can i ues configMap to put all variable in file and call here from file
```

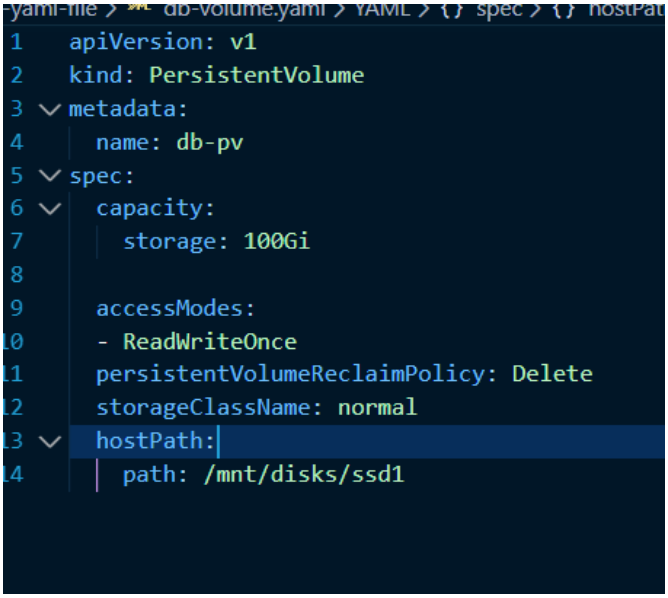
- Also I use liveness to check health if replica not work to restart pods

```
  livenessProbe:
    httpGet:
      path: /health.html
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
```

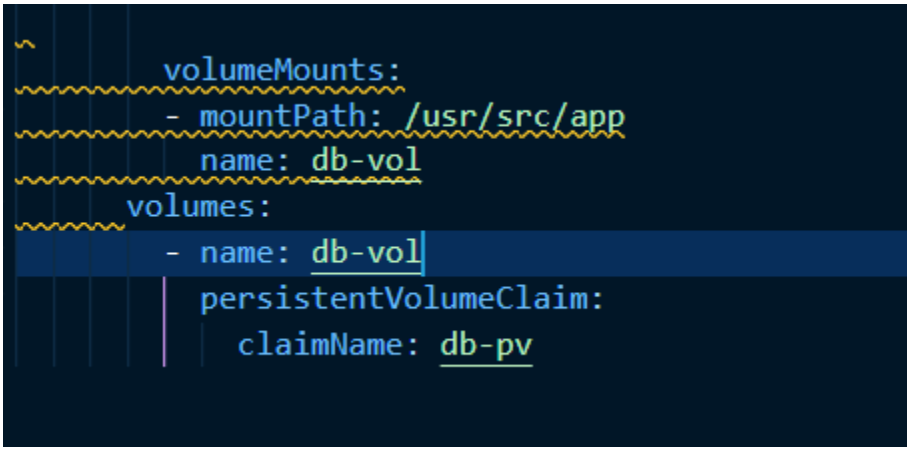
- Also create volume for app and db

```
      volumes:
        - name: test-volume
    # This AWS EBS volume must already exist.
          awsElasticBlockStore:
            volumeID: vol-0edd6d691eb848827
            fsType: ext4
```

- Also I create Volume persistent Claim for DB



And the add to db pod file like that



Also this file use deployment because if I have pods and want to deploy new version or roll back to old version if I want to roll back to specific version first should see history and then select version number use these commands

kubectl rollout history deployment/deploy-app

#kubectl rollout undo deployment/deploy-app --to-revision 2