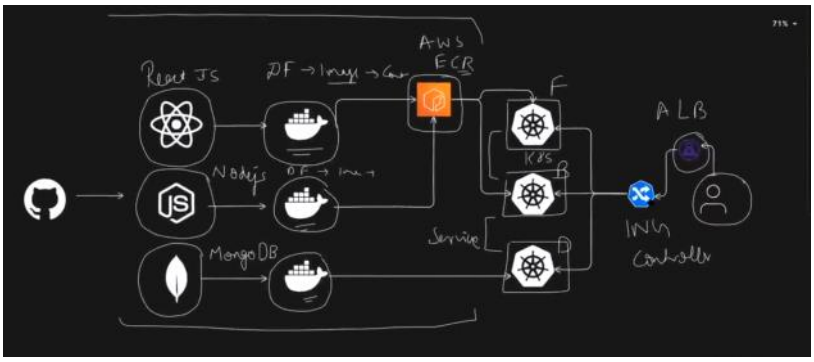
Three-tier Application Deployment on AWS EKS

03 December 2024 14:36

Tech Stack -
React JS - Frontend
Node JS - Backend
Mongo DB - Database
Docker
Kubernetes
AWS ECR



First will create container of all three.
React JS
Node JS
Docker file - Image - Container.
Mongo DB
For mongodb image can be used directly as its available online easily.

React JS and Node JS images will we stored in AWS ECR(Elastic Container Registry).

Now, Deploy on k8s.
Ways to Create cluster -
Minikube , Kubeadm ,EKS CTL , AKS , GKE.
In cluster 3 tier will be running frontend , backend , database.
And these tiers or kubernetes will communicate each other using Service.
And consider if I am a person of outside and I want to do routing like If I insert slash it should goes in frontend and if slash api then it should go to backend for this we will use ingress controller.
Now if the application is big more no. of people will use it for this we will use ALB load balancing.

First we will create an Workstation or we can say EC2 Instance on which we gonna work for the deployment.

3-tier-HQ

First plan - Bring Frontend React JS from github make its docker and push it to the ECR.

Now connect to instance and then git clone the application repository to the server.

Now we gonna make its docker file.

To get the information about the project.

cat package.json

Node Command - npm(Node Package Manager).

vim Dockerfile

(Base Image) From node 14: Stable version for node applications.

Working Directory - WORKDIR app

We need to copy and install all the packages from the file like package.json of the application to the working directory which is app.
COPY Package*.json ./
Two types are there
pacakge.json
packagelock.json
package*.json - * means Starting from package and ending till the json will be copied.

Run num install - To install all the packages.

Now the supporting files for the application are ready so we can bring the code and run it.

COPY . . - Copy everything to your current folder or container.

Now we will run.
Diff between run and CMD.

Run commands are for intermediate layers like if we wants to install packages and libraries.
and if I want to give the entry like want to run the command after the creation of the container that we will use CMD.

CMD ["npm" , "start"]

./ - Present WORKDIR is app therefore always it will be app.

- From node:14
- WORKDIR /app
- COPY package*.json ./
- RUN npm install
- COPY . .
- CMD ["npm" , "start"]

So, as of now we have written the docker file now we have to run the docker file so for it we gonna install the docker first.

- sudo apt-get update
- sudo apt-install docker.io

But now if I run docker ps commands it won't run as I don’t have any permissions for var/run/docker.sock.

To resolve this weather we can add our user to the docker group or we can give the permission of particular this socket to the user.
One thing is that if you are as root user than you can access directly.

- echo $user
- sudo chown $USER /var/run/docker.sock
- docker build -t 3-tier-hq/application-code/frontend .

So now I have got the docker image of the application.

- docker images

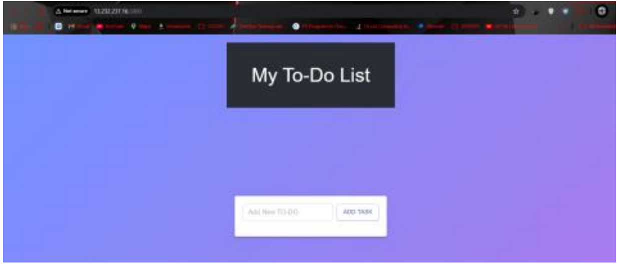
By default react JS frontend runs on 3000 port.

To delete all the unused images -

- docker images prune -a
- docker system prune -a
- docker rmi $(docker images -q)
- docker rmi -f $(docker images -q)

- docker run -d -p 3000:3000 three-tier-frontend:latest

Now change security from aws to open the 3000 port to run it.



- http://13.232.237.16:3000/

Now pause and terminate this conatiner.

- docker kill e418b3708888753fbf5edbc050f54d0761a56fb613b12f22bf7308ff1a20912d

Now as our frontend application is running now we need to push its docker image to the ECR.

Now there are some prerequiestes for this like AWS CLI.
Install it on the home directory.

- curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
- sudo apt install unzip
- unzip awscliv2.zip

To move the files in the bin folders so that if when we run aws commands infuture binaries can understand it.

- sudo ./aws/install -i /usr/local/aws-cli -b /usr/local/bin --update

Now we have installed the AWS CLI but now we have to configure it too so for that we will require an iam user for it with administrator access.

Give Administrator access to it and create access key from credentials and mark it for use on CLI.

- aws configure

To restore AWS CLI functionality:
# Create symlink to AWS CLI

- sudo ln -s /usr/local/aws-cli/v2/2.22.12/bin/aws /usr/local/bin/aws
- sudo ln -s /usr/local/aws-cli/v2/2.22.12/bin/aws_completer /usr/local/bin/aws_completer

To remove AWS CLI completely

```
# Uninstall AWS CLI
pip uninstall awscli
pip3 uninstall awscli

# Remove AWS CLI configuration files
rm -rf ~/.aws
rm -rf ~/.config/aws

# Remove AWS CLI executable
sudo rm /usr/local/bin/aws
sudo rm /usr/local/bin/aws_completer
```

Now we will create ECR where we can store images can make any type public or private.

Name as three-tier-fronend and create as public and then go on view push commands.

Run first command to login to the AWS ECR.

And the cd to the frontend directory and the build the image.

```
docker build -t three-tier-frontend .
```

Now tag the frontend image to the ECR attack.

```
docker tag three-tier-frontend:latest public.ecr.aws/d6d6f1j7/three-tier-frontend:latest
```

Now push the frontend.

```
docker push public.ecr.aws/d6d6f1j7/three-tier-frontend:latest
```

Inshort to push the image on ECR.

Simple prerequisites - AWSCLI , User with the permissions and push commands knowledge.

Now we will make image of backend and push it to the ECR.

In this command should be run to index.js cause it’s a node js application.

Dockerfile

```
From node:14

WORKDIR /app

COPY package*.json ./

Run npm install

COPY . .

CMD ["node","index.js"]
```

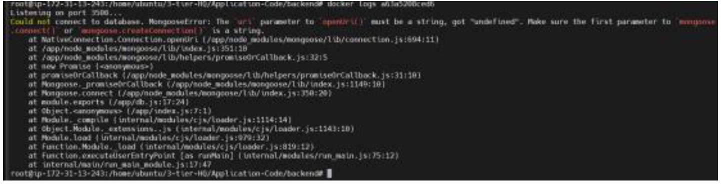
Now create ECR for backend and follow same process login using command and then build the image and tag and then push the image.

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/d6d6f1j7


docker build -t three-tier-backend .
docker tag three-tier-backend:latest public.ecr.aws/d6d6f1j7/three-tier-backend:latest
docker push public.ecr.aws/d6d6f1j7/three-tier-backend:latest

docker run -d -p 3500:3500 three-tier-backend:latest
```

Now backend won't be run until its not connected to database and we have not created the database yet which is Mongo DB.



```
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Application-Code/backend# docker logs a63a5208cedb
Listening on port 3500...
Could not connect to database. MongooseError: The `uri` parameter to `openUri()` must be a string, got "undefined". Make sure the first parameter to `mongoose
.connect()` or `mongoose.createConnection()` is a string.
    at NativeConnection.Connection.openUri (/app/node_modules/mongoose/lib/connection.js:694:11)
    at /app/node_modules/mongoose/lib/index.js:351:10
    at /app/node_modules/mongoose/lib/helpers/promiseOrCallback.js:32:5
    at new Promise (<anonymous>)
    at promiseOrCallback (/app/node_modules/mongoose/lib/helpers/promiseOrCallback.js:31:10)
    at Mongoose._promiseOrCallback (/app/node_modules/mongoose/lib/index.js:1149:10)
    at Mongoose.connect (/app/node_modules/mongoose/lib/index.js:350:20)
    at module.exports (/app/db.js:17:24)
    at Object.<anonymous> (/app/index.js:7:1)
    at Module._compile (internal/modules/cjs/loader.js:1114:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1143:10)
    at Module.load (internal/modules/cjs/loader.js:979:32)
    at Function.Module._load (internal/modules/cjs/loader.js:819:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:75:12)
    at internal/main/run_main_module.js:17:47
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Application-Code/backend#
```

Now we will run the Mongo DB directly with the kubernetes.

Now we will make AWS EKS so for creating a cluster on AWS EKS we have to install a tool name as eksctl and to control these clusters the kubectl needs to be installed.
And always install these things on the home directory other wise their files will go also on github if we don't remove those files.

KUBECTL -

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
```

```
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
```

We transfer files in bin so that we don’t have to use ./ everytime cause by this the files comes into the environmental path.

--silent - means it won't show log and will download in background.

AWS EKS -

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```

# Uninstall kubectl

```
sudo apt-get remove kubectl
sudo apt-get purge kubectl
rm -rf ~/.kube
```

# Remove eksctl

```
sudo rm /usr/local/bin/eksctl
```

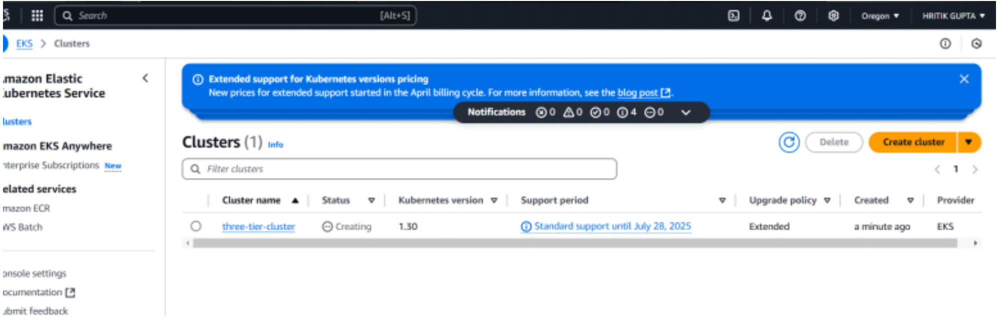
# Remove any eksctl-related files

```
rm -rf ~/.eksctl
```

Now command to setup EKS Cluster -

```
eksctl create cluster --name three-tier-cluster --region us-west-2 --node-type t2.medium --nodes-min 2 --nodes-max 2
kubectl get nodes
```

Now it will take time and whatever the stack is in creating it's creating by cloud formation.



Extended support for Kubernetes versions pricing
New prices for extended support started in the April billing cycle. For more information, see the blog post.

Clusters (1) Info

	Cluster name	Status	Kubernetes version	Support period	Upgrade policy	Created	Provider
○	three-tier-cluster	⊖ Creating	1.30	ⓘ Standard support until July 28, 2025	Extended	a minute ago	EKS

In us-west-2 its getting created.

2 Nodes are running in this eks cluster.

Now we have to bind kubectl with our eks cluster.

Now in general the cluster will come using.

```
kubectl get nodes
```

but if we want to bring particular nodes of the cluster than we can set the context.

```
aws eks update-kubeconfig --region us-west-2 --name three-tier-cluster
```

Now EKS is also ready now we need to create manifest of kubernetes.

In general we take the yaml file of nginix manifest from google and make the yaml manifesst to run nginx but if we want to run the particular like backend of three tier application than we can use the image of it directly to run it.

and the path will be of image will be the ECR path of image.

In mongo db yaml deploy file we create conatiner.

and for deploy user password will require for this we have to create an kubernetes secret file for this we can use template of kubernetes secret to create it and use those credentails in kubernetes yaml file using file name and key and these crendentials we can encrypt using | base64.

and Service we will create so that other applications and deployment can access the Mongo DB.
These are only for internal access.
Services are used so that internal applications can commuincate to each other.
which give access and name to the port on which mongodb is running port is running.

Like to connect from backend to database the url will be MongoDB://mongodb-svc

Now we will create or paste the three files of the database.

deploy.yaml - To run the containers of MongoDB.

secrets.yaml - To run and get the username and passwords for MongoDB.

service.yaml - To access the deployment of Mongo DB to the other remain applications.

Now we have to create namespace of workshop and apply or run these yaml files inside the cluster.

```
kubectl create namespace workshop
```

★ deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: workshop
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mon
        image: mongo:4.4.6
        command:
            - "numactl"
            - "--interleave=all"
            - "mongod"
            - "--wiredTigerCacheSizeGB"
            - "0.1"
            - "--bind_ip"
            - "0.0.0.0"
        ports:
        - containerPort: 27017
        env:
          - name: MONGO_INITDB_ROOT_USERNAME
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: username
          - name: MONGO_INITDB_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: password
```

```
kubectl apply -f deployment.yaml
```

To see the any deployment inside an namespace.

```
kubectl get deployment -n workshop
```

```
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
mongodb   0/1     1            0           58s
```

As we can see its not ready yet as we have not passed the secrets yet.

```
apiVersion: v1
kind: Secret
metadata:
  namespace: workshop
  name: mongo-sec
type: Opaque
data:
  password: c2Fuc2thcmd1cHRhCg== #sanskargupta
  username: YWRtaW4K #admin
```

```
kubectl apply -f secrets.yaml
```

Now Mongo DB is running we can check using -

```
kubectl get deployment -n workshop
```

```
kubectl get pods -n workshop
```

Now docker conatiner of Mongo DB and kubernetes deployment has been created and pod is running.

To check service of this Mongo DB in the workshop namespace.

```
kubectl get service -n workshop
```

Now we need to create the service.

```
apiVersion: v1
kind: Service
metadata:
  namespace: workshop
  name: mongodb-svc
spec:
  selector:
    app: mongodb
  ports:
  - name: mongodb-svc
    protocol: TCP
    port: 27017
    targetPort: 27017
```

```
kubectl apply -f service.yaml
```

Now the service name as Mongo DB has been create inside the ClusterIP.
and ClusterIP is written because if we do not written type of service then it takes by default as Cluster IP.

Now the Database of mongo db is running on kubernetes.

Now we will run the backend by creating its yaml files.

First Change the backend image from ECR to the Backend deployment.yaml file.

deployment.yaml

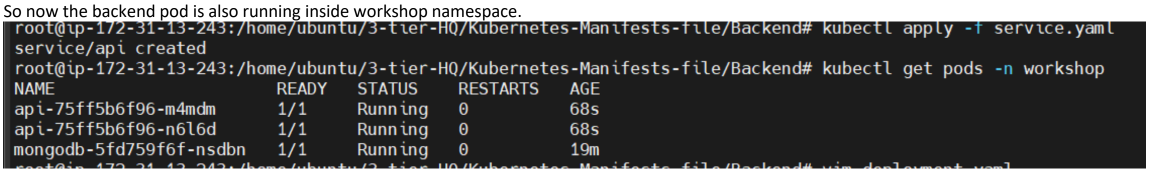
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api
  namespace: three-tier
  labels:
    role: api
    env: demo
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 25%
  selector:
    matchLabels:
      role: api
  template:
    metadata:
      labels:
        role: api
    spec:
      imagePullSecrets:
      - name: ecr-registry-secret
      containers:
      - name: api
        image: public.ecr.aws/d6d6f1j7/three-tier-backend:latest
        imagePullPolicy: Always
        env:
          - name: MONGO_CONN_STR
            value: mongodb://mongodb-svc:27017/todo?directConnection=true
          - name: MONGO_USERNAME
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: username
          - name: MONGO_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: password
        ports:
        - containerPort: 3500
        livenessProbe:
          httpGet:
            path: /ok
            port: 3500
          initialDelaySeconds: 2
          periodSeconds: 5
        readinessProbe:
          httpGet:
            path: /ok
            port: 3500
          initialDelaySeconds: 5
          periodSeconds: 5
          successThreshold: 1
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: api
  namespace: three-tier
spec:
  ports:
  - port: 3500
    protocol: TCP
  type: ClusterIP
  selector:
    role: api
```

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

So now the backend pod is also running inside workshop namespace.



```
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Kubernetes-Manifests-file/Backend# kubectl apply -f service.yaml
service/api created
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Kubernetes-Manifests-file/Backend# kubectl get pods -n workshop
NAME                       READY   STATUS    RESTARTS   AGE
api-75ff5b6f96-m4mdm       1/1     Running   0          68s
api-75ff5b6f96-n6l6d       1/1     Running   0          68s
mongodb-5fd759f6f-nsdbn    1/1     Running   0          19m
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Kubernetes-Manifests-file/Backend# vim deployment.yaml
```

Now its connected to database.

```
kubectl logs api-75ff5b6f96-m4mdm -n workshop
```

So now our database and logical tier has been completed.

Now we need to work on presentation tier.
We can make a service of it and can give it to user which can be run by user but that's not a good idea cause internal deployments should talk internal only for outside access we have to attach a Load Balancer ALB and if we want to do routing internally than we can attach ingress controller.

Now work on Frontend Deployment.

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: frontend
 namespace: workshop
 labels:
   role: frontend
   env: demo
spec:
 replicas: 1
 strategy:
   type: RollingUpdate
   rollingUpdate:
     maxSurge: 1
     maxUnavailable: 25%
 selector:
   matchLabels:
     role: frontend
 template:
   metadata:
     labels:
       role: frontend
   spec:
     containers:
     - name: frontend
       image: public.ecr.aws/d6d6f1j7/three-tier-frontend:latest
       imagePullPolicy: Always
       env:
         - name: REACT_APP_BACKEND_URL
           value: "http://challange.trainwithshubham.com/api/tasks"
       ports:
       - containerPort: 3500
```

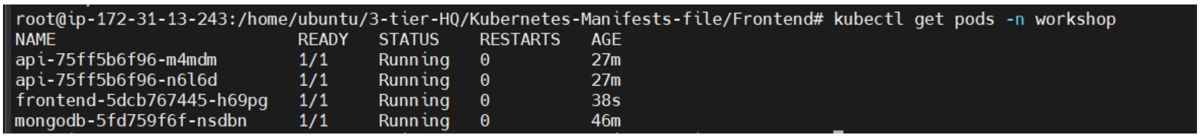
service.yaml

```
apiVersion: v1
kind: Service
metadata:
 name: frontend
 namespace: workshop
spec:
 ports:
 - port: 3500
   protocol: TCP
 type: ClusterIP
 selector:
   role: frontend
```

```
kubectl apply -f deployment.yaml
```

```
kubectl apply -f service.yaml
```

Now, the frontend pods is also running all the three applications and pods are running now frontend , backend and databse.



```
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Kubernetes-Manifests-file/Frontend# kubectl get pods -n workshop
NAME                        READY   STATUS    RESTARTS   AGE
api-75ff5b6f96-m4mdm        1/1     Running   0          27m
api-75ff5b6f96-n6l6d        1/1     Running   0          27m
frontend-5dcb767445-h69pg   1/1     Running   0          38s
mongodb-5fd759f6f-nsdbn     1/1     Running   0          46m
```

Now who I can access the application using frontend for this we have to use ingress controller which can help in this routing.

So to install the ingress controller we will use helm.

Helm is basically a package manager kind which package the manifests of kubernetes.

The thing is muliple yaml files needs to be written to run the load balancer or ingress controller therefore helm is there in which package all the necessary yaml files are already written which just need needs to be installed directly using helm.

ALB will run using eksctl cause it does't related to anything kubernetes.

EKSCTL will tell cluster to add one loadbalancer to it.

So first we will install an IAM policy of AWS Load Balancer. { It tells how to transmit outside traffic to the cluster.}

curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam_policy.json

```
aws iam create-role --role-name AmazonEKSLoadBalancerControllerRole --assume-role-policy-document '{
  "Version": "2012-10-17",
  "Statement": [
   {
    "Effect": "Allow",
    "Principal": {
     "Federated": "arn:aws:iam::992382408215:oidc-provider/oidc.eks.us-west-2.amazonaws.com/id/YOUR_CLUSTER_OIDC_PROVIDER_ID"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
     "StringEquals": {
      "oidc.eks.us-west-2.amazonaws.com/id/https://oidc.eks.us-west-2.amazonaws.com/id/A8ABC095D702D2A351545B9C95D2C35F:sub": "system:serviceaccount:kube-system:aws-load-balancer-controller"
     }
    }
   }
  ]
}'
```

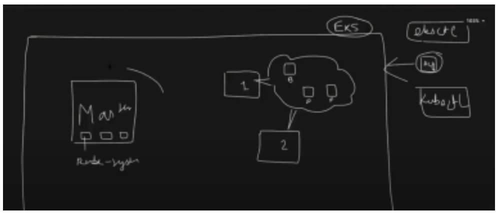
Now we creating an IAM policy by which eks and loadbalancer can get the connectivity.

aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json

Now we will install utils which will help in attachment of policies to the eks cluster.

eksctl utils associate-iam-oidc-provider --region=us-west-2 --cluster=three-tier-cluster --approve

It will create a service account which will help in communication of services to eachother.



namespace is of kubernetes services which is kube system which can impact kubernetes architecture.

eksctl create iamserviceaccount --cluster=three-tier-cluster --namespace=kube-system --name=aws-load-balancer-controller --role-name AmazonEKSLoadBalancerControllerRole --attach-policy-arn=arn:aws:iam::992382408215:policy/AWSLoadBalancerControllerIAMPolicy --approve --region=us-west-2

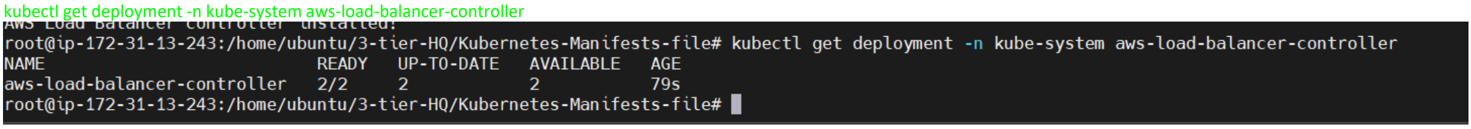
Now the loadbalancer and service account has been created.
Now we have to install the loadbalancer in our kubernetes cluster.

We will install helm inside the cluster now.

sudo snap install helm --classic
helm repo add eks https://aws.github.io/eks-charts
helm repo update eks
helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=three-tier-cluster --set serviceAccount.create=false --set serviceAccount.name=aws-

load-balancer-controller

kubectl get deployment -n kube-system aws-load-balancer-controller



```
AWS Load Balancer controller installed!
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Kubernetes-Manifests-file# kubectl get deployment -n kube-system aws-load-balancer-controller
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           79s
root@ip-172-31-13-243:/home/ubuntu/3-tier-HQ/Kubernetes-Manifests-file# 
```

kubectl apply -f full_stack_lb.yaml

Now the ALB is ready too now we have to use ingress controller for routing so that on load balancer user can run and use the specific services.

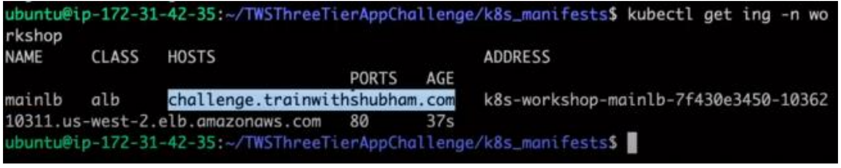
Now we have to create full stack load balancer ingress.yaml file.

vim full_stack_lb.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mainlb
  namespace: workshop
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}]'
spec:
  ingressClassName: alb
  rules:
    - host: challange.trainwithshubham.com
      http:
        paths:
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: api
                port:
                  number: 8080
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 3500
```

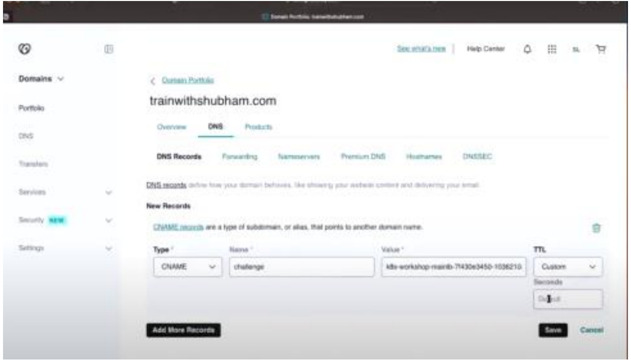
kubectl apply -f full_stack_lb.yaml

So, the load balancer will connect to the domain url and then route it to the kubernetes cluster.
Now we just have to buy the domain and assign the address and subdomian in the domain which will be linked to the aws load balancer and then we can access the application using url.



```
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ kubectl get ing -n wo
rkshop
NAME     CLASS   HOSTS                                ADDRESS
                             PORTS   AGE
mainlb   alb     challenge.trainwithshubham.com       k8s-workshop-mainlb-7f430e3450-10362
10311.us-west-2.elb.amazonaws.com    80      37s
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ 
```

kubectl get ing -n workshop



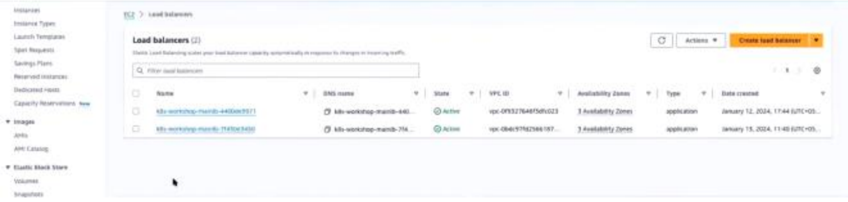
trainwithshubham.com

DNS records define how your domain behaves, like showing your website content and delivering your email.

New Records

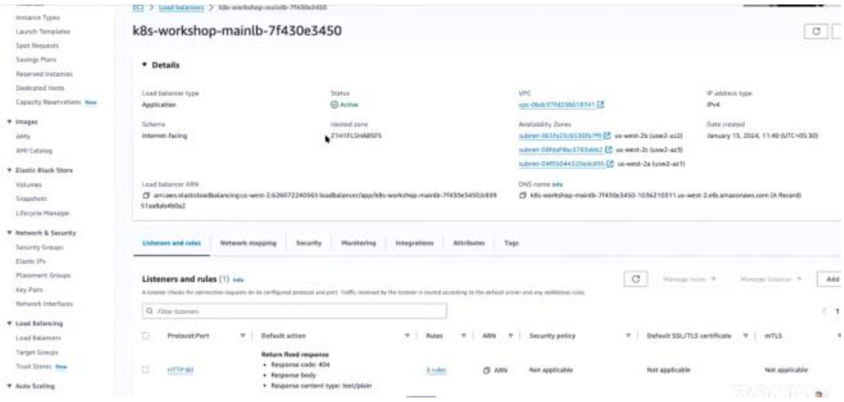
CNAME records are a type of subdomain, or alias, that points to another domain name.

Type *	Name *	Value *	TTL
CNAME	challenge	k8s-workshop-mainlb-7f430e3450-1036210	Custom



Load balancers (2)

Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
k8s-workshop-mainlb-[illegible]	k8s-workshop-mainlb-440...	Active	vpc-0f[illegible]	3 Availability Zones	application	January 12, 2024, 17:44 (UTC+05...
k8s-workshop-mainlb-7f430e3450	k8s-workshop-mainlb-7f4...	Active	vpc-0b[illegible]	3 Availability Zones	application	January 13, 2024, 11:40 (UTC+05...



EC2 > Load balancers > k8s-workshop-mainlb-7f430e3450

k8s-workshop-mainlb-7f430e3450

▼ Details

Load balancer type: Application

Status: Active

VPC: vpc-0bdc97fd258618741

IP address type: IPv4

Scheme: Internet-facing

Hosted zone: Z1H1FLSHABSF5

Availability Zones: subnet-065fa23c6530fb7f9 us-west-2b (usw2-az2)
subnet-08fdaf4bc3783abb2 us-west-2c (usw2-az3)
subnet-04f05044325edc895 us-west-2a (usw2-az1)

Date created: January 13, 2024, 11:40 (UTC+05:30)

Load balancer ARN: arn:aws:elasticloadbalancing:us-west-2:626072240565:loadbalancer/app/k8s-workshop-mainlb-7f430e3450/c839 51aa8ab4b0a2

DNS name Info: k8s-workshop-mainlb-7f430e3450-1036210311.us-west-2.elb.amazonaws.com (A Record)

Listeners and rules (1) Info

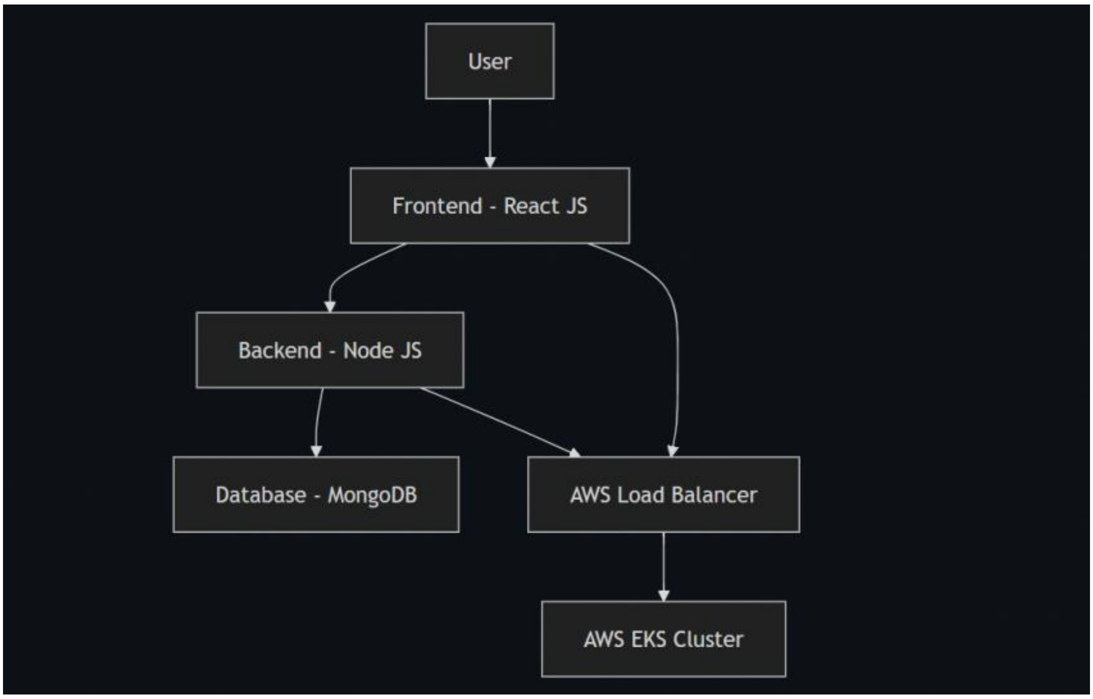
A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

□	Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate	mTLS
□	HTTP:80	Return fixed response • Response code: 404 • Response body • Response content type: text/plain	3 rules	ARN	Not applicable	Not applicable	Not applicable

Now we can enter into mongo db conatiner and can check the database table using mongo.

To delete the cluster

```
eksctl delete cluster --name three-tier-cluster --region us-west-2
```



User
Frontend - React JS
Backend - Node JS
Database - MongoDB
AWS Load Balancer
AWS EKS Cluster