End To End Deployment of React-App on AWS EKS with Jenkins

Create and ssh to EC2 instance.

ADD ports- 80, 8080, 3000 in Inbounds rules to ec2 security group.

1. install java

```
sudo apt update
sudo apt-get install default-jdk -y
java --version
```

2. install jenkins on server

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update
sudo apt-get install jenkins
```

3. Start jenkins

```
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
```

Make sure port 8080 and 80 are added in inbounds rules security group.
To access jenkins page: “ec2-public-ip:8080”

4. install AWS CLI for working with AWS services as EKS, ECR

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

5. install eksctl - command line tool for working with EKS clusters that automates many individual tasks.

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

6. install kubectl - command line tool for working with Kubernetes clusters

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
sudo touch /etc/apt/sources.list.d/kubernetes.list
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl

7. create AWS EKS cluster with eksctl cli
But jenkins-ec2 instance requires certain privileges to create the cluster.
2 ways:
1. create access key and secreat key
2. create IAM role with admin access policy
I will use 2nd.

8. Create iam role with AdministrationAccess and attach with ec2.

9. Create AWS EKS cluster using eksctl:

eksctl create cluster --name my-cluster --region us-east-1 --nodegroup-name my-nodes
--node-type t3.small --managed --nodes 2

Once Cluster is created verify it-
kubectl get nodes
kubectl get pods
kubectl get ns
kubectl get svc

10. Create AWS ECR repo

11. Install Docker
sudo apt install docker.io -y
sudo usermod -aG docker $USER

12. Install Jenkins Plugins
Docker, Dockerpipeline, Kubernetes CLI

13. Verify If EKS Cluster is up and running
eksctl get cluster --name my-eks-cluster --region us-east-1

kubectl command works with "/home/ubuntu/.kube/config"

14. Setup Connection Between Jenkins and Kubernetes using "/home/ubuntu/.kube/config" file.

cat /home/ubuntu/.kube/config

Copy the content and save it in a text file and upload it in jenkins.
Path is : manage jenkins-> manage Credentials

Restart Docker and Jenkins to make sure all the changes are reflected.

sudo systemctl stop docker
sudo systemctl start docker
sudo systemctl daemon-reload
sudo systemctl status docker

sudo usermod -a -G docker jenkins
sudo service jenkins restart

EKS Cluster and ECR Repo has been created.
All the required packages and plug-ins has been installed.

Final Step: **Now Lets Build the jenkins pipeline for deployment.**

Build Pipeline: There will be 4 stages.

1. Checkout - clone git repo
2. Build docker
3. Push image to ECR
4. Deploy microservices into AKS Cluster

PipeLine Jenikns:

```
pipeline {
        agent any
        environment{
        registry = "532806123370.dkr.ecr.us-east-1.amazonaws.com/react-app"
        }
```

```
        stages {
        stage('Checkout from version Control') {
        steps {
                checkout scmGit(branches: [[name: '*/gh-pages-branch']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/shashiiitp19/my-React-Docker-K8S-app.git']])
        }
        }

        stage('Build Docker') {
        steps {
                script{
                docker.build registry
                }
        }
        }
        stage('push image to ECR') {
        steps {
                sh "aws ecr get-login-password --region us-east-1 | docker login --username
AWS --password-stdin 532806123370.dkr.ecr.us-east-1.amazonaws.com"
                sh "docker push
532806123370.dkr.ecr.us-east-1.amazonaws.com/react-app:latest"
        }
        }
        stage('K8S Deploy') {
        steps {
        script {
                sh ('aws eks update-kubeconfig --name my-eks-cluster --region us-east-1')
                sh "kubectl apply -f deploy-k8s-eks.yaml"
                 }
        }
        }
        }
}
```

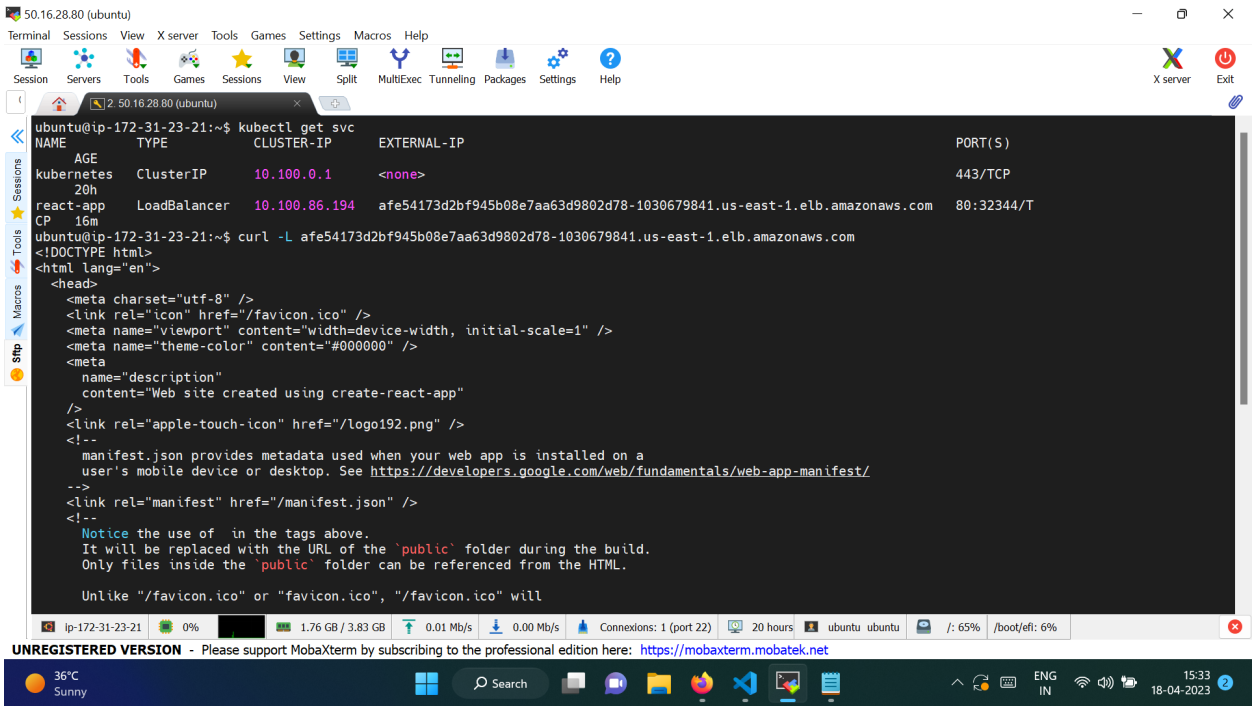
If deployment is successful then we can see all the details-

kubectl get deployment
kubectl get pods
kubectl get svc

Verify if React App is Running -

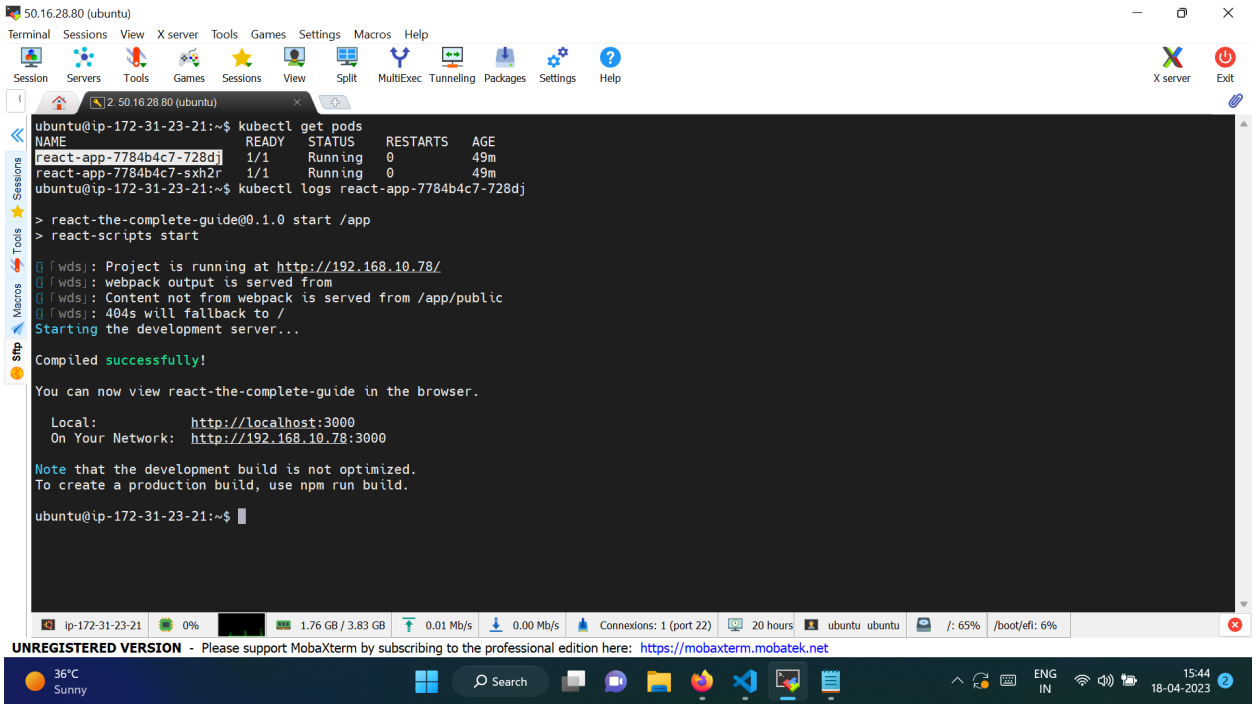
Use LoadBalancer URL:

1. kubectl get svc
2. curl -L afe54173d2bf945b08e7aa63d9802d78-1030679841.us-east-1.elb.amazonaws.com

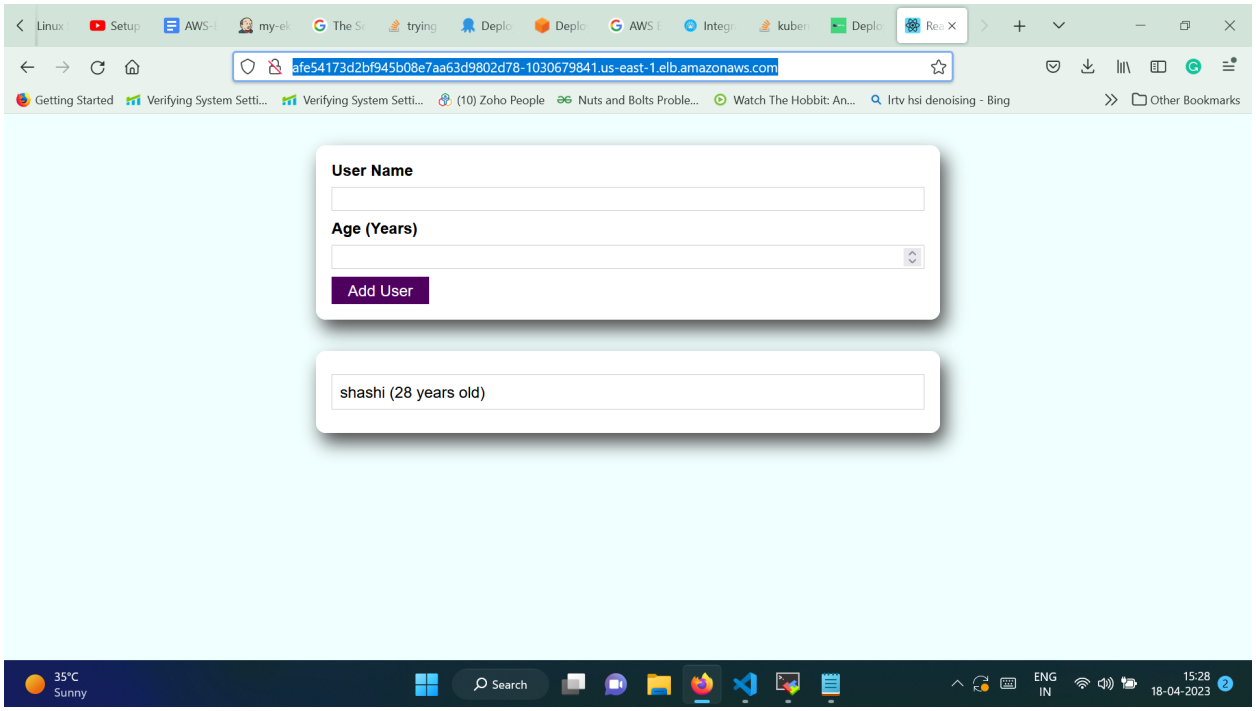


Or else to verify if app is running or not-

1. kubectl get pods
2. kubectl logs react-app-7784b4c7-728dj



USE LoadBalancer's URL in browser to verify the React App is Running.



Once deployment is done, DELETE all the resources:

Delete EKS Cluster & Node Groups

Step-01: Delete Node Group

List EKS Clusters
eksctl get clusters

Capture Node Group name
eksctl get nodegroup --cluster=<clusterName>
eksctl get nodegroup --cluster=eksdemo1

Delete Node Group
eksctl delete nodegroup --cluster=<clusterName> --name=<nodegroupName>
eksctl delete nodegroup --cluster=eksdemo1 --name=eksdemo1-ng-public1

Step-02: Delete Cluster

Delete Cluster
eksctl delete cluster <clusterName>
eksctl delete cluster eksdemo1

kubectl delete all --all

Example-

pod "react-app-7784b4c7-728dj" deleted
pod "react-app-7784b4c7-sxh2r" deleted
service "kubernetes" deleted
service "react-app" deleted
deployment.apps "react-app" deleted
replicaset.apps "react-app-7784b4c7" deleted