

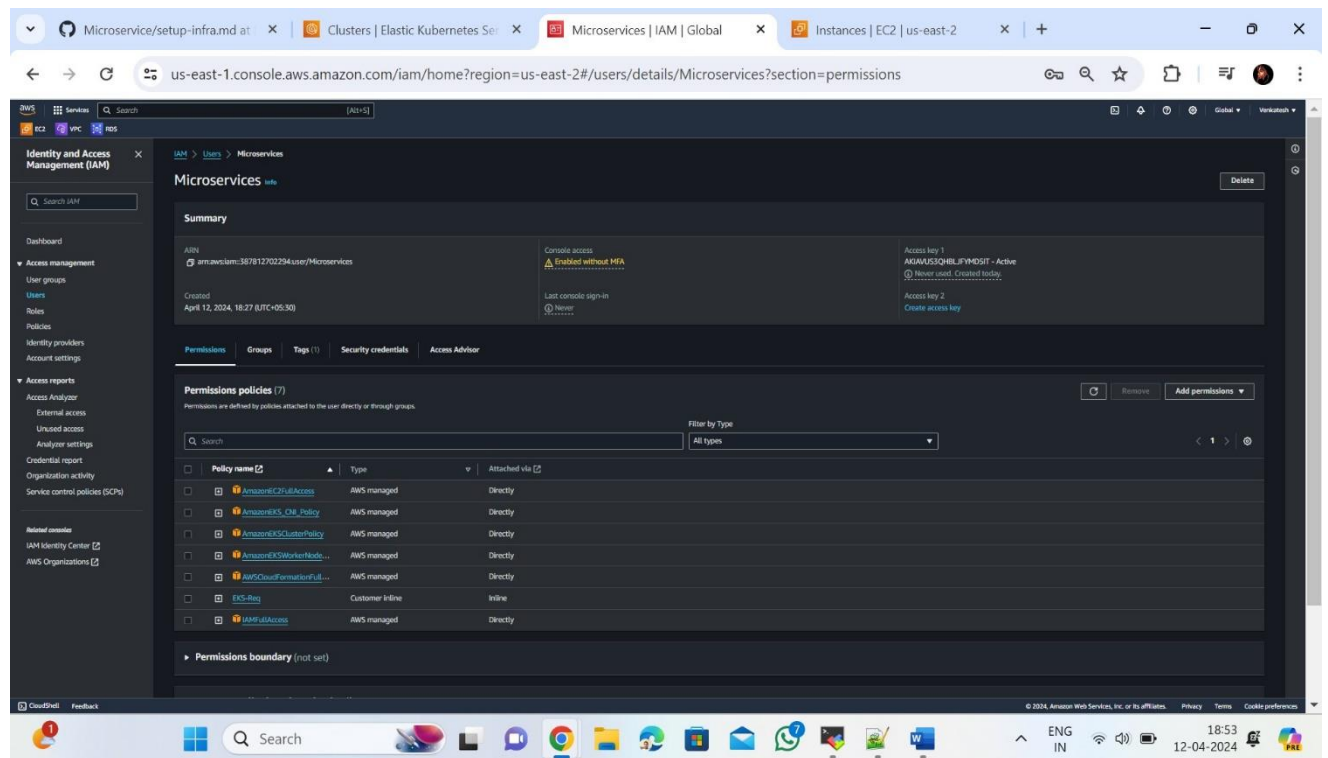
# 11-Microservices CI-CD Pipeline DEVOPS Project:

<https://github.com/Singareddy-Venkatesh/Microservice.git>

→ Create EC2 Instance with t2.Large

→ Go to IAM Create user and attach policy

- EKS-Req → go to add permission → Create inline Policy → Add JSON And Paste It



AmazonEC2FullAccess  
AmazonEKS\_CNI\_Policy  
AmazonEKSClusterPolicy  
AmazonEKSWorkerNodePolicy







AWSCloudFormationFullAccess

IAMFullAccess

One more policy we need to create with content as below


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "eks:*",
      "Resource": "*"
    }
  ]
}
```

Attach this policy to your user as well

<input type="checkbox"/>	Policy name <a href="#">↗</a>	Type	Attached via <a href="#">↗</a>
<input type="checkbox"/>	 <a href="#">AmazonEC2FullAccess</a>	AWS managed	Directly
<input type="checkbox"/>	 <a href="#">AmazonEKS_CNI_Policy</a>	AWS managed	Directly
<input type="checkbox"/>	 <a href="#">AmazonEKSClusterPolicy</a>	AWS managed	Directly
<input type="checkbox"/>	 <a href="#">AmazonEKSWorkerNodePolicy</a>	AWS managed	Directly
<input type="checkbox"/>	 <a href="#">AWSCloudFormationFullAccess</a>	AWS managed	Directly
<input type="checkbox"/>	 <a href="#">eksfullaccess</a>	Customer inline	Inline

eksfullaccess [Copy JSON](#) [Edit](#)

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "VisualEditor0",
6       "Effect": "Allow",
7       "Action": "eks:*",
8       "Resource": "*"
9     }
10  ]
11 }
```

<input type="checkbox"/>	 <a href="#">IAMFullAccess</a>	AWS managed	Directly
--------------------------	---	-------------	----------

Vi microservice.sh

# AWSCLI

```
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 configure
```

## 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
```

## EKSCTL

```
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
```

[chmod +x microservice.sh](#)

[./microservice.sh](#)

- ➔ [aws configure](#)
- ➔ [provide access key and secret access keys and zone.](#)
- ➔ [Vi cluster.sh](#)

## Create EKS CLUSTER

```
eksctl create cluster --name=EKS-1 \
                      --region=ap-south-1 \
                      --zones=ap-south-1a,ap-south-1b \
                      --without-nodegroup
```

```
eksctl utils associate-iam-oidc-provider \
    --region ap-south-1 \
    --cluster EKS-1 \
    --approve
```

```
eksctl create nodegroup --cluster=EKS-1 \
                       --region=ap-south-1 \
                       --name=node2 \
                       --node-type=t3.medium \
                       --nodes=3 \
                       --nodes-min=2 \
                       --nodes-max=4 \
                       --node-volume-size=20 \
                       --ssh-access \
                       --ssh-public-key=DevOps \
                       --managed \
                       --asg-access \
                       --external-dns-access \
```

```
--full-ecr-access \  
--appmesh-access \  
--alb-ingress-access
```

→ `chmod +x Cluster.sh`

→ `./cluster.sh`

→ Install Jenkins and Docker

→ `vi jenkins.sh`

```
sudo apt update -y  
apt install openjdk-17-jre-headless -y  
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \  
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key  
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 -y  
sudo apt-get install jenkins -y  
sudo systemctl enable jenkins  
sudo systemctl start jenkins  
sudo systemctl status jenkins
```

```
chmod +x jenkins.sh
```

```
./jenkins.sh
```

→ `sudo apt install docker.io -y`

→ `chmod 666 /var/run/docker.sock`

→ Change docker Image [venkatesh09:latest](#) in the GitHub Account in Every Microservices.

→ Install Plugins (Docker, docker pipeline, Kubernetes, Kubernetes CLI, Multiple Scan Webhook Trigger.

→ In Jenkins New Item and Select Multi-branch pipeline

→ Manage Jenkins → Tools → docker setup and add credentials (docker username and Password)

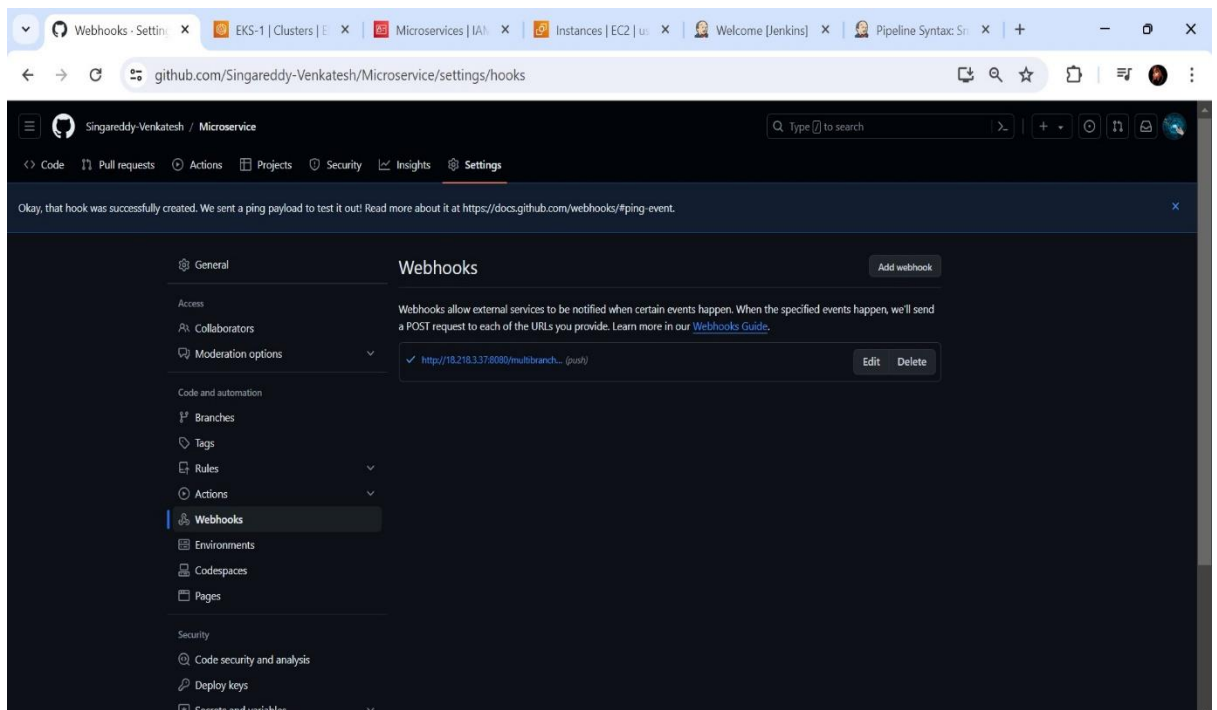
→ Kubernetes Credentials By using Secret Text

→ In Jenkins pipeline Go to Branch Source → select git

→ go to scan Multibranch pipeline triggers → select scan by webhook (Give Any name) venky

Ex. Jenkins URL/multibranch-webhook-trigger/invoke?  
Token=venky

<http://18.218.3.37:8080/multibranch-webhook-trigger/invoke?token=VenkateswaraReddy>



→Go to GitHub Account Settings select Webhook and paste in it→select Application/JSON→select just the push event→add Webhook

Here whenever code changes the pipeline is Automatically Triggered Itself →**CONTINUOUS INTEGRATION**

## **CONTINUOUS DEPLOYMENT: Create Role Based Service Account: -**

**Create Service Account, Role & Assign that role, and create a secret for Service Account and generate a Token**

### **Creating Service Account**

→**kubectl create namespace webapps**

Vi svc.yml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: webapps
```

→**kubectl apply -f svc.yml**

### **Create Role**

Vi role.yml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
  - apiGroups:
    - ""
    - apps
    - autoscaling
    - batch
    - extensions
```

```

    - policy
    - rbac.authorization.k8s.io
resources:
  - pods
  - componentstatuses
  - configmaps
  - daemonsets
  - deployments
  - events
  - endpoints
  - horizontalpodautoscalers
  - ingress
  - jobs
  - limitranges
  - namespaces
  - nodes
  - pods
  - persistentvolumes
  - persistentvolumeclaims
  - resourcequotas
  - replicaset
  - replicationcontrollers
  - serviceaccounts
  - services
verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]

```

→ **kubectl apply -f role.yml**

## Bind the role to service account

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: webapps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- namespace: webapps
  kind: ServiceAccount
  name: jenkins

```

→ **kubectl apply -f bind.yml**

## Generate token using service account in the namespace

[Create Token](#)

Vi secret.yml

```
apiVersion: v1
```

```
kind: Secret
```

```
kubernetes.io/service-account.name: jenkins
```

→ `kubectl apply -f secret.yml -n webapps`

→ `kubectl describe secret mysecretname -n webapps`

## →Kubernetes Credentials By using Secret Text

[illegible]

Take a sample pipeline and create Kubernetes pipeline by using pipeline syntax:

Create New file in GitHub in **main** branch with the name

## Jenkinsfile

```
pipeline {
```

agent any



```

stages {
  stage('Deploy To Kubernetes') {
    steps {
      withKubeCredentials(kubectxCredentials:
[[caCertificate: '', clusterName: 'EKS-1', contextName: '',
credentialsId: 'k8-token', namespace: 'webapps', serverUrl:
'https://4B67A39F29010B03576C25C3C3EE080A.gr7.us-east-
2.eks.amazonaws.com']])) {
        sh "kubectl apply -f deployment-service.yml -n
webapps"
      }
    }
  }

  stage('verify Deployment') {
    steps {
      withKubeCredentials(kubectxCredentials:
[[caCertificate: '', clusterName: 'EKS-1', contextName: '',
credentialsId: 'k8-token', namespace: 'webapps', serverUrl:
'https://4B67A39F29010B03576C25C3C3EE080A.gr7.us-east-
2.eks.amazonaws.com']])) {
        sh "kubectl get svc -n webapps"
      }
    }
  }
}

```

```
}  
  
}  
  
}
```

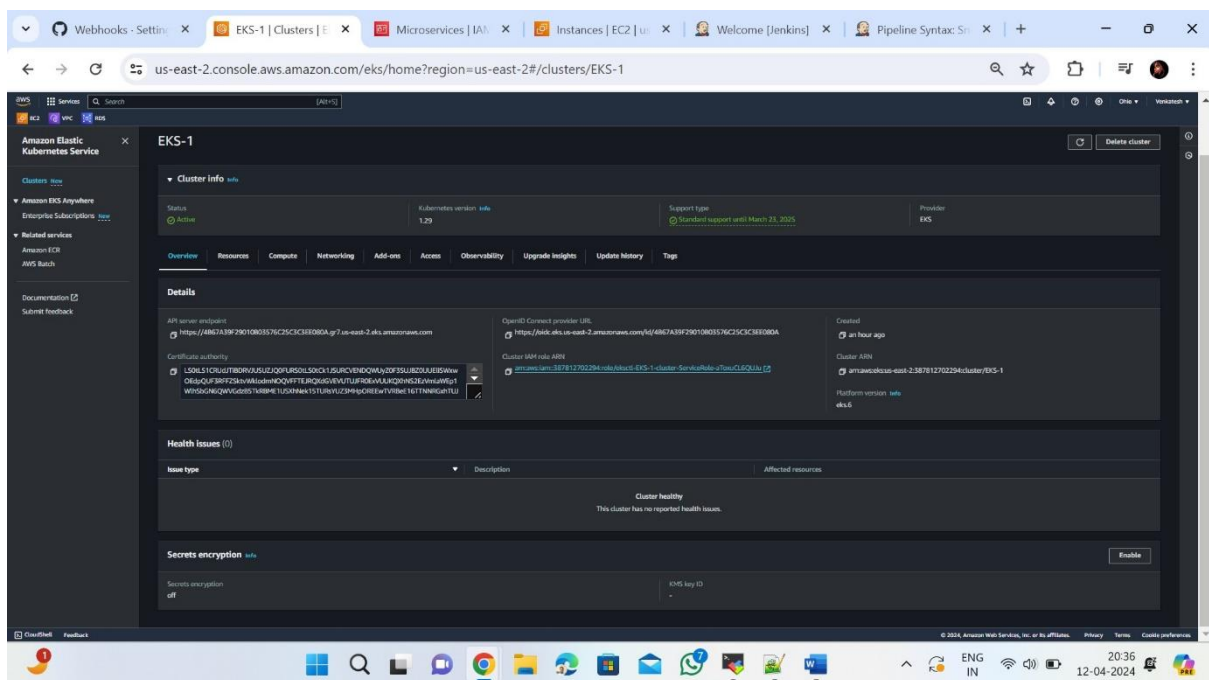
## Plugin: withKubecredentials: Kubernetes CLI

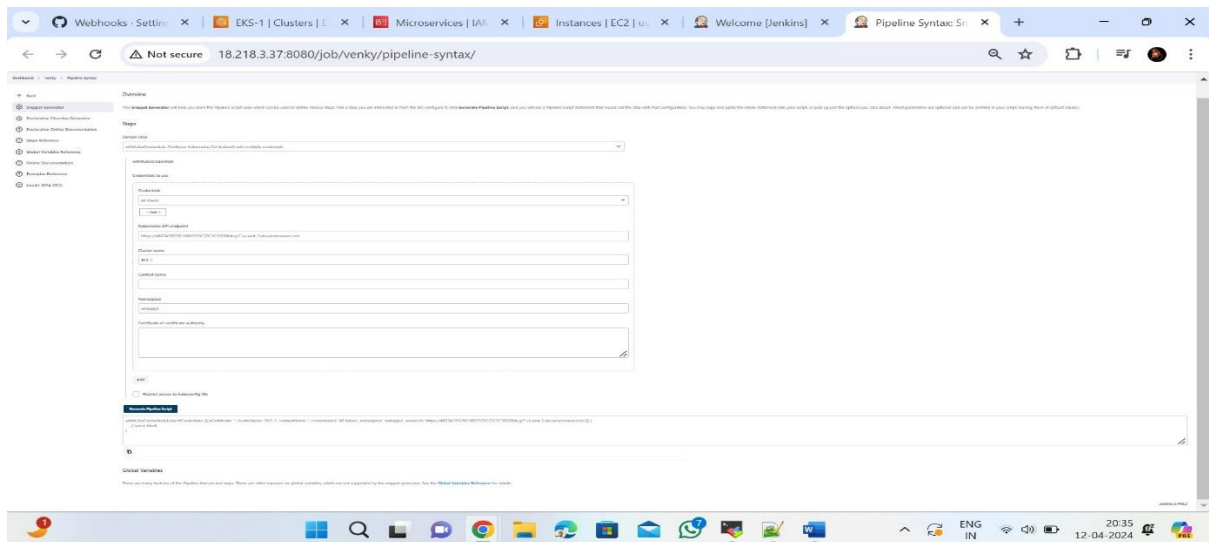
Kubernetes API Endpoint (You will Get in the AWS Cluster Info)

Credentials: Select Token

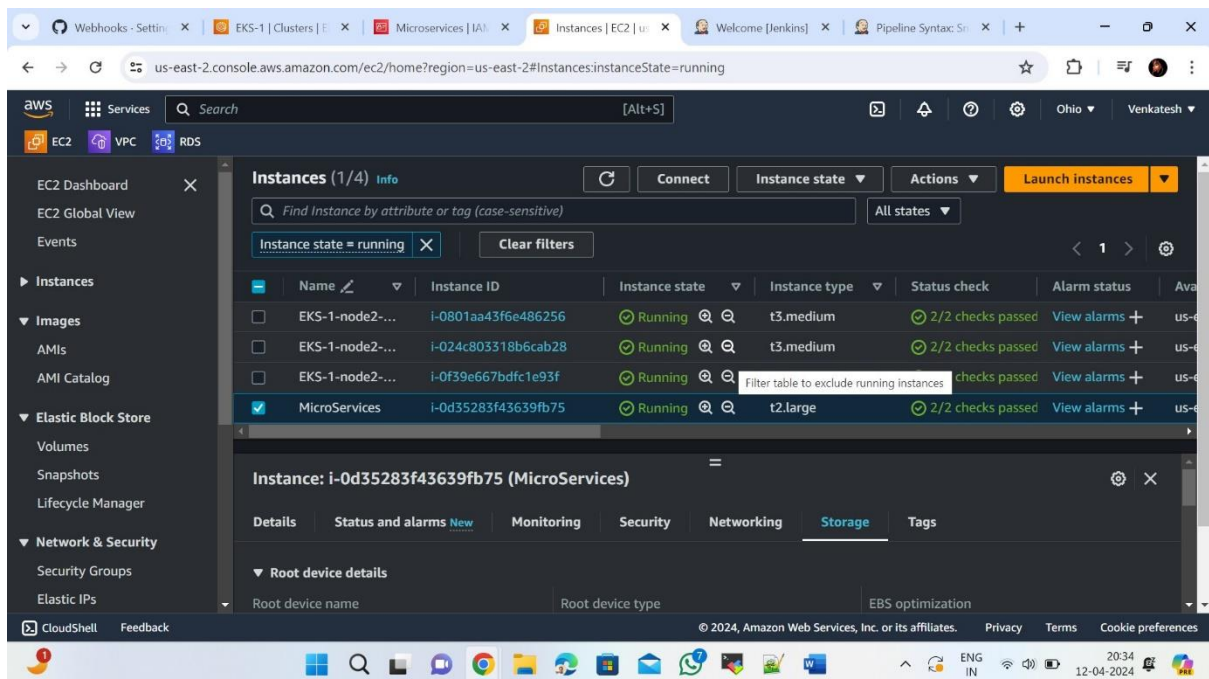
Cluster name: EKS-1 (See Your Created Cluster)

Namespace: webapps

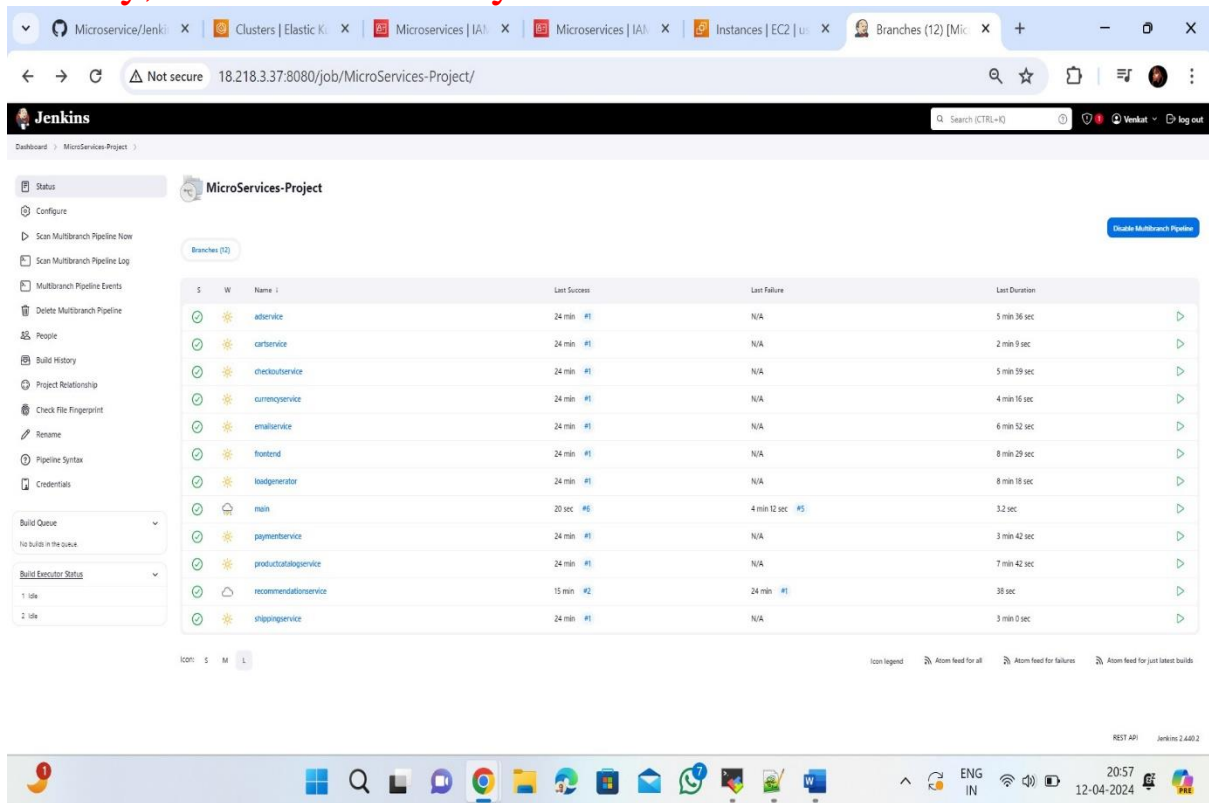




Here It Will Creates Worker Nodes Automatically:



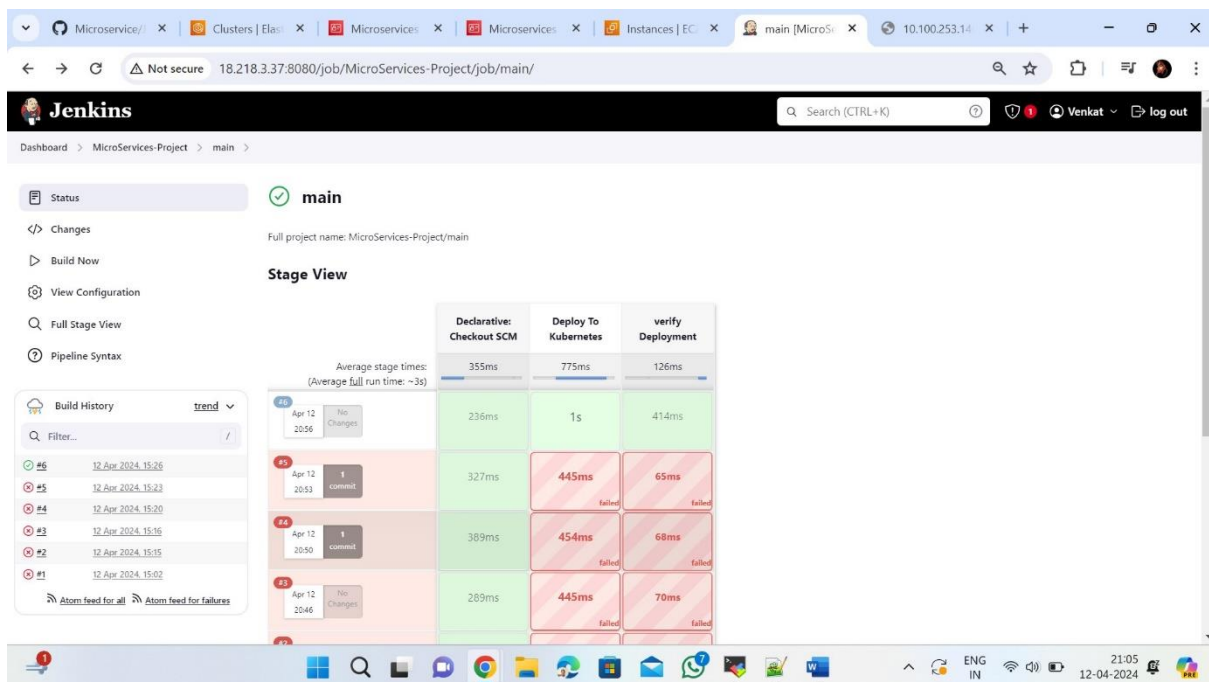
## Finally, Here Successfully Creates Our Microservices:



The screenshot shows the Jenkins Dashboard for the 'MicroServices-Project'. The left sidebar contains navigation options like Status, Configure, and Build Queue. The main area displays a table of 12 branches, each with a status icon, name, last success/failure time, and duration. The branches are: adservice, carservice, checkoutservice, currencyservice, emailservice, frontend, loadgenerator, main, paymentservice, productcatalogservice, recommendationsservice, and shippingservice. The 'main' branch is highlighted with a green status icon.

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀	adservice	24 min #1	N/A	5 min 36 sec
✓	☀	carservice	24 min #1	N/A	2 min 9 sec
✓	☀	checkoutservice	24 min #1	N/A	5 min 59 sec
✓	☀	currencyservice	24 min #1	N/A	4 min 16 sec
✓	☀	emailservice	24 min #1	N/A	6 min 52 sec
✓	☀	frontend	24 min #1	N/A	8 min 29 sec
✓	☀	loadgenerator	24 min #1	N/A	8 min 18 sec
✓	☀	main	20 sec #6	4 min 12 sec #5	32 sec
✓	☀	paymentservice	24 min #1	N/A	3 min 42 sec
✓	☀	productcatalogservice	24 min #1	N/A	7 min 42 sec
✓	☀	recommendationservice	15 min #2	24 min #1	38 sec
✓	☀	shippingservice	24 min #1	N/A	3 min 0 sec

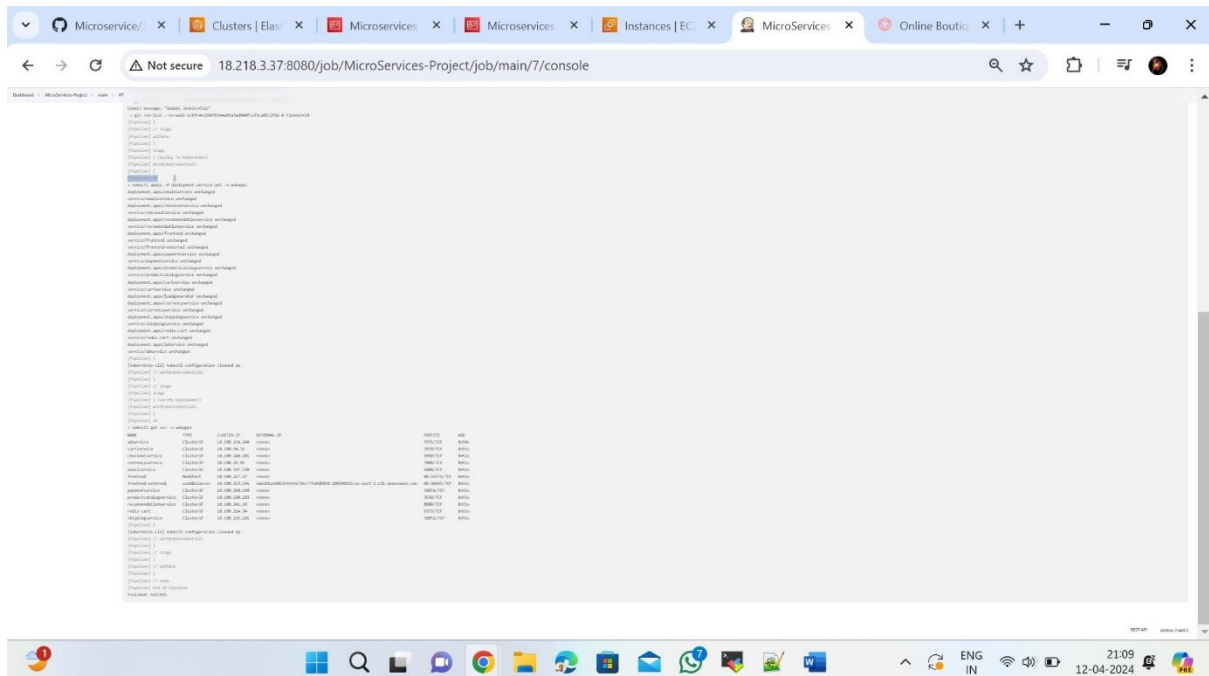
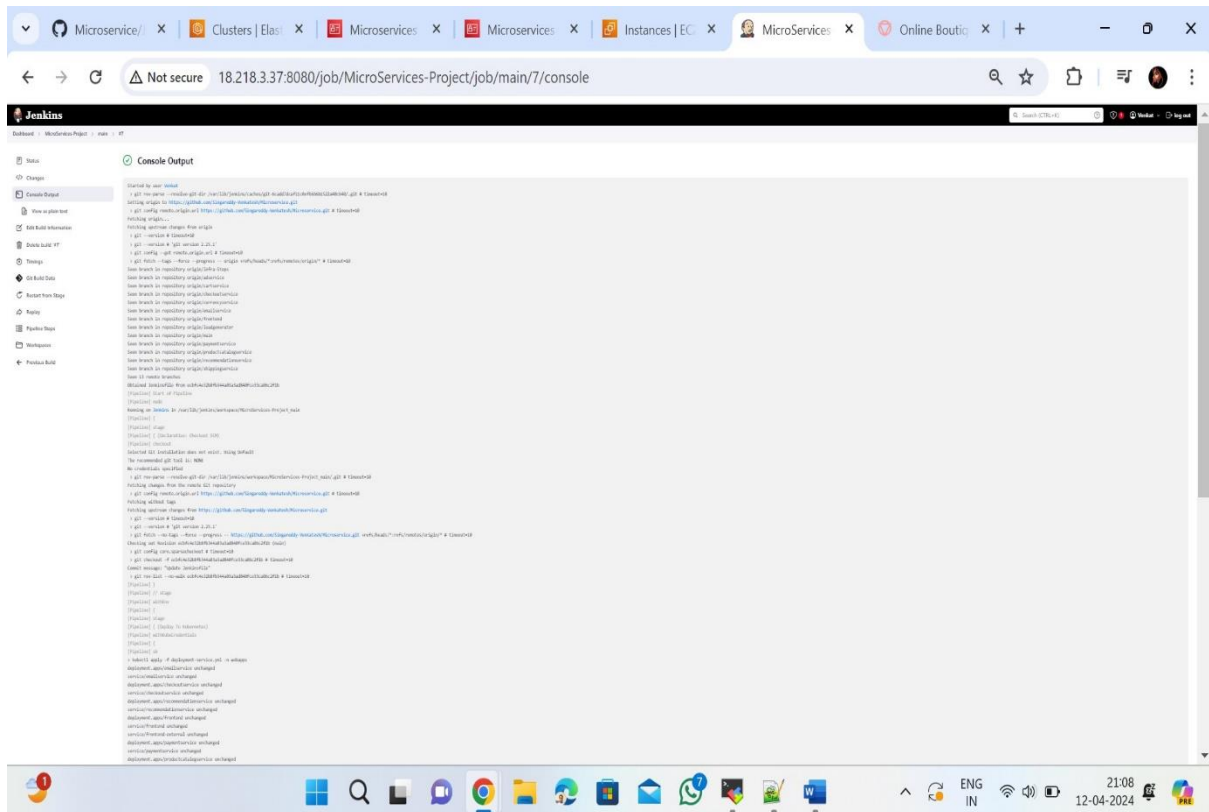
## Kubernetes Deployment:



The screenshot shows the Jenkins Stage View for the 'main' branch. The left sidebar contains navigation options like Status, Changes, Build Now, View Configuration, Full Stage View, and Pipeline Syntax. The main area displays a table of stage metrics for the 'main' branch. The stages are: Declarative: Checkout SCM, Deploy To Kubernetes, and verify Deployment. The table shows the average stage times and the duration of each stage for the last build.

	Declarative: Checkout SCM	Deploy To Kubernetes	verify Deployment
Average stage times: (Average full run time: ~3s)	355ms	775ms	126ms
#6 Apr 12 20:56	236ms	1s	414ms
#5 Apr 12 20:53	327ms	445ms	65ms
#4 Apr 12 20:50	389ms	454ms	68ms
#3 Apr 12 20:46	289ms	445ms	70ms

## Console Output:



Here You will Get Load balancer URL In Console Output from that you can access in the browser.

# Here Our Final Output:

