**Hotel-Side Hospital**

**Devops Capstone Project 3**

**Description**

Hotel-Side Hospital, a globally renowned hospital chain headquartered in Australia, is aiming to streamline its operation by setting up an infrastructure within the hotel premises. However, in order to maintain seamless functioning and scalability, they require fully managed virtual machines (VMs) on the Amazon Web Services (AWS) platform. The organization seeks an automated provisioned infrastructure solution that can enable them to effortlessly create new Amazon Elastic Kubernetes Service (EKS) clusters, whenever required, and promptly delete them when they are no longer needed. This will optimize resource allocation and enhance operational efficiency.

**Task (Activities)**

1. Validate if Terraform is installed in the virtual machine

2. Install AWS CLI

3. Navigate to AWS IAM service, and get AWS Access key and Secret Key to connect AWS with

the AWS CLI

4. Export the AWS Access Key, Secret Key, and Security Token to configure AWS CLI

connectivity with AWS Cloud

5. Create terraform scripts to create a new VM using autoscaling which includes the following

files: autoscaling.tf, VPC.tf, internetgateway.tf, subnets.tf (public subnet), routetable.tf,

Route\_table\_association\_with\_public\_subnets.tf.

6. Execute terraform scripts

7. Connect to an instance and install the stress utility (The stress files are provided along with

the problem statement document.)

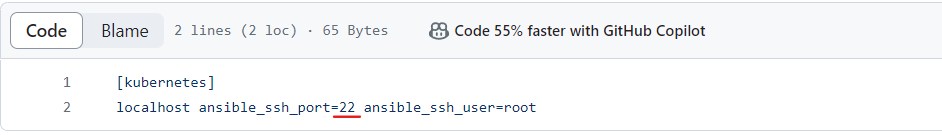
8. Validate if autoscaling is working by putting load on autoscaling group

**Step 1 : Update the Inventory.Yaml and Jenkinsfile**

1. Update the Jenkinsfile with the following code and commit changes

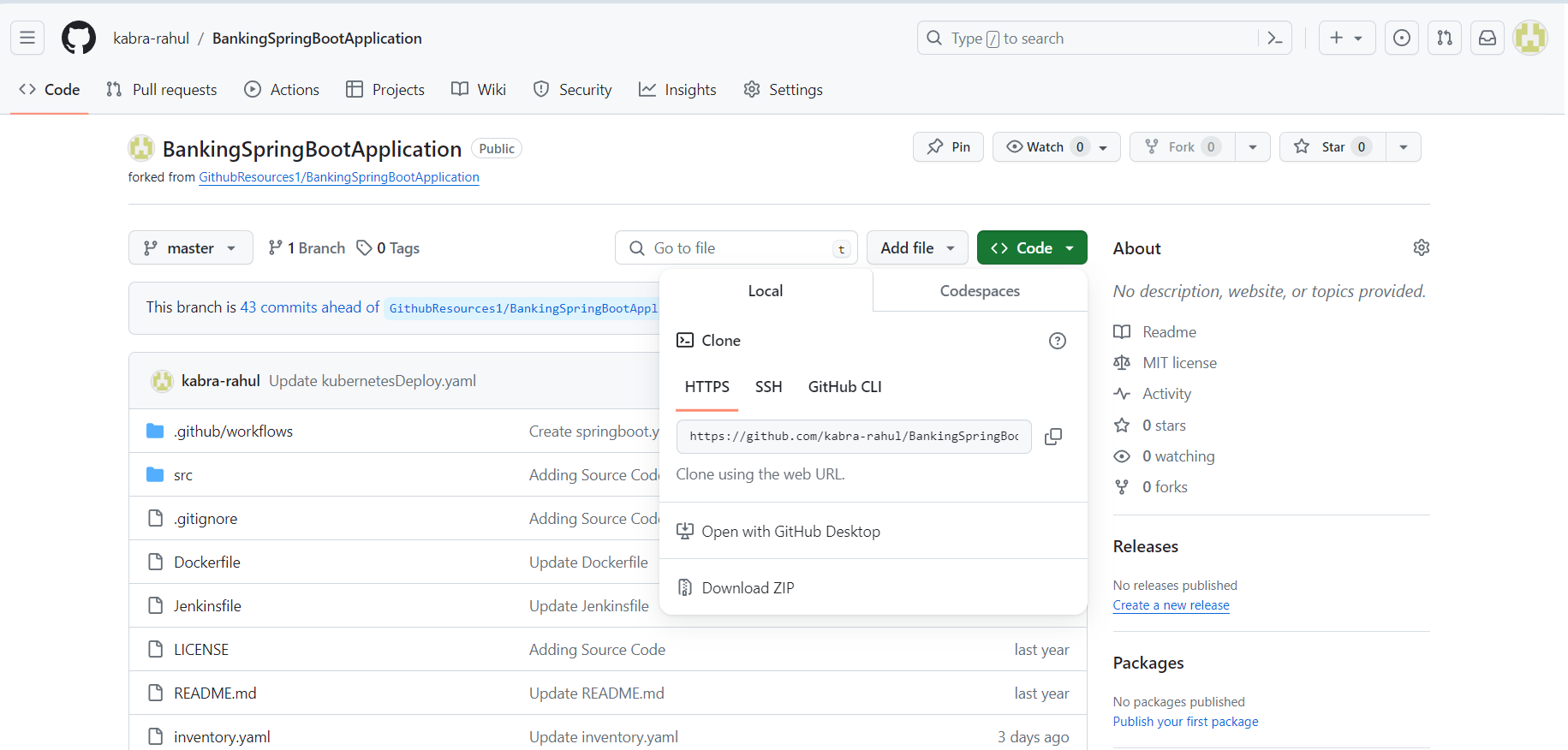


1. Update the inventory with with following code and commit the changes



1. Copy the project link

[**https://github.com/kabra-rahul/BankingSpringBootApplication.git**](https://github.com/kabra-rahul/BankingSpringBootApplication.git)



**Step 2 : Create an EC2 Instance and IAM user**

2.1 Launch and EC2 instance with the following configuration

**Name – devops-server**

**Number of Instance -1**

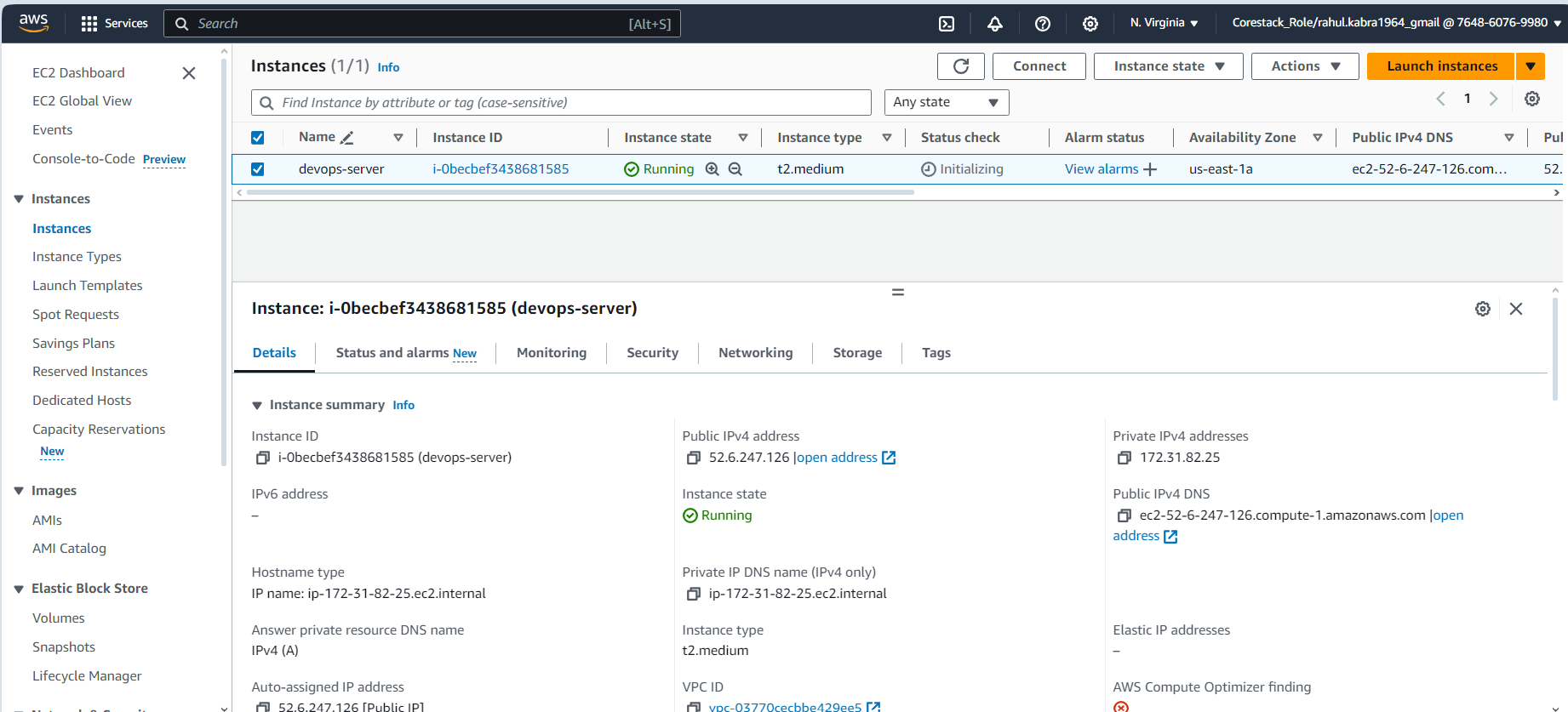
**AMI - Amazon Linux 2 Kernel 5.10 AMI 2.0.20240223.0 x86\_64 HVM gp2**

**Instance Type – t2.medium**

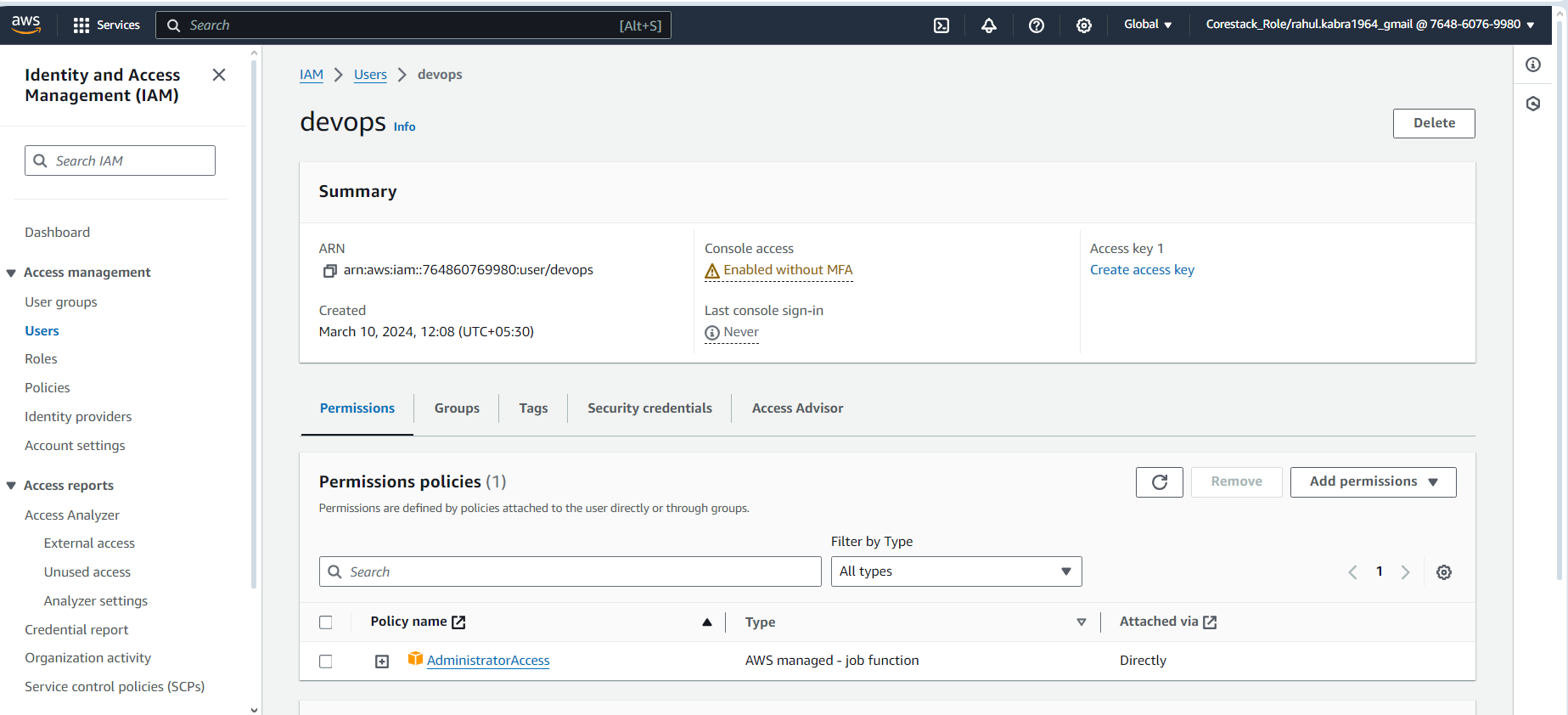
**Key pair – devops.pem**

**Select Create Security Group –> Select Allow SSH Traffic and HTTP traffic from**

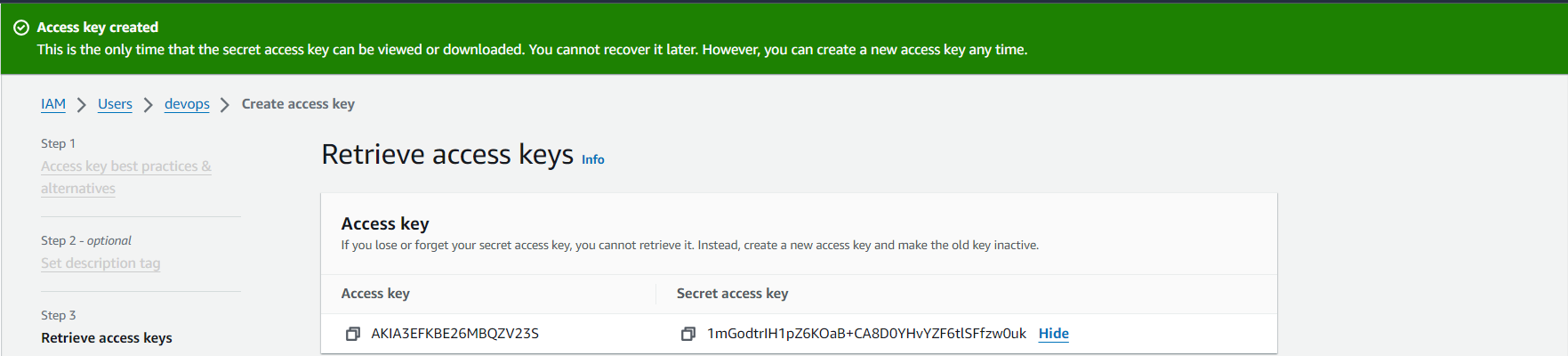
**Add Inbound Security Rule -> All Traffic (0.0.0.0/0)**

****

2.2 Create a new IAM user devops with Administrator role

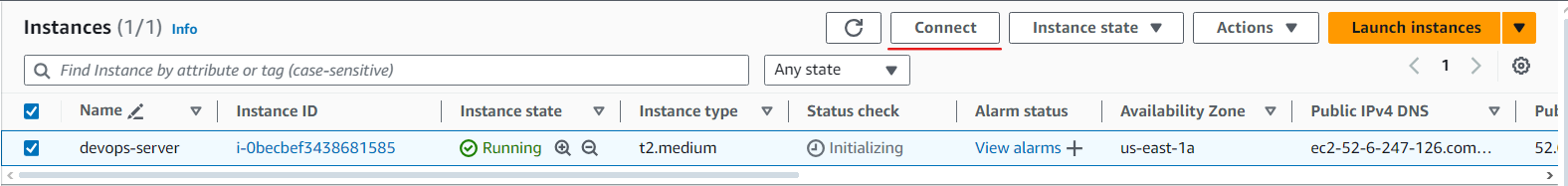


2.3 Create the access key and secret access key and save them

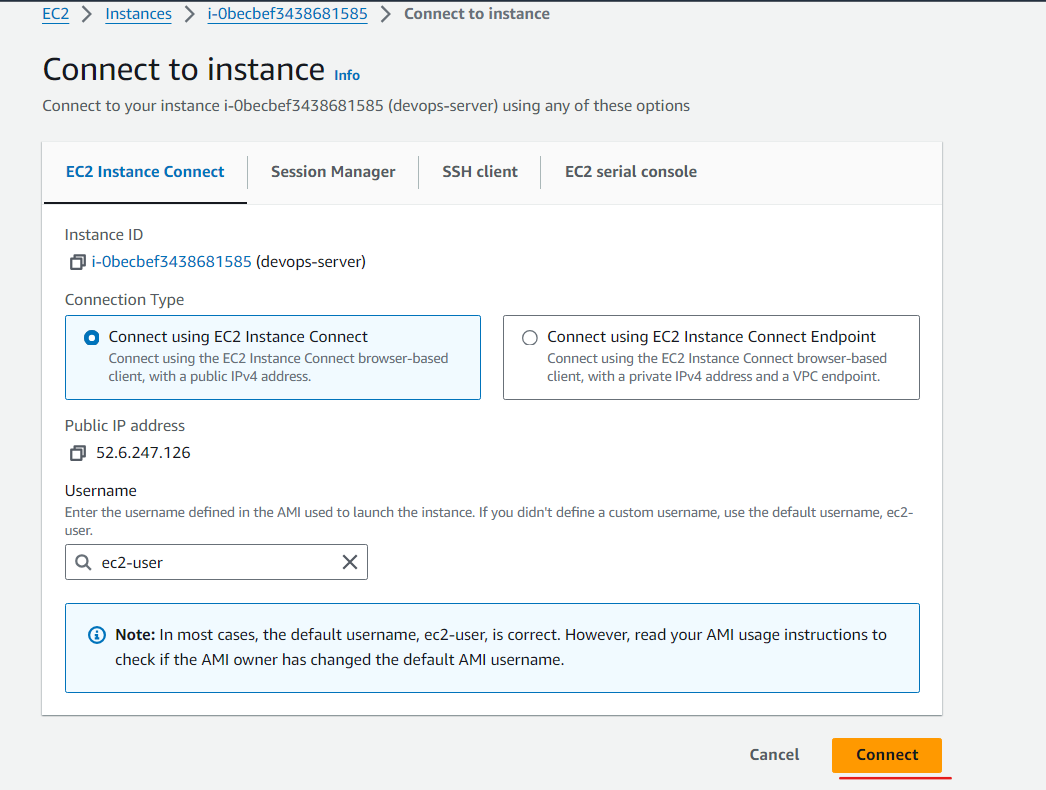


**Step 3 : Connecting to EC2 Machine and Installing package**

3.1 Go to EC2 instance home and select devops-server and click on **Connect**



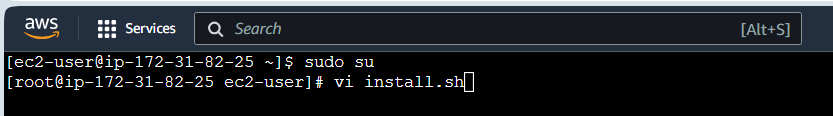
3.2 Click on **Connect**



3.3 Run the below commands in the terminal and create a file install.sh

**sudo su**

**vi install.sh**

****

3.4 Type the below command in the install.sh

**#!/bin/bash**

**#Installing Java and Jenkins**

**sudo yum update –y**

**sudo amazon-linux-extras install java-openjdk11 -y**

**sudo wget -O /etc/yum.repos.d/jenkins.repo \https://pkg.jenkins.io/redhat-stable/jenkins.repo**

**sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key**

**sudo yum upgrade**

**sudo yum install jenkins -y**

**sudo systemctl enable jenkins**

**sudo systemctl start jenkins**

**#Installing NodeJs**

**curl -fsSL https://rpm.nodesource.com/setup\_16.x | sudo bash -**

**sudo yum install -y nodejs**

**#Installing Maven**

**sudo yum install -y maven**

**#Installing Docker**

**sudo yum install -y docker**

**sudo systemctl start docker**

**sudo systemctl enable docker**

**#Installing python and Ansible**

**sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm -y**

**sudo yum update -y**

**sudo yum install git python python-level python-pip openssl ansible -y**

**#Installing Terraform**

**sudo yum install -y yum-utils shadow-utils**

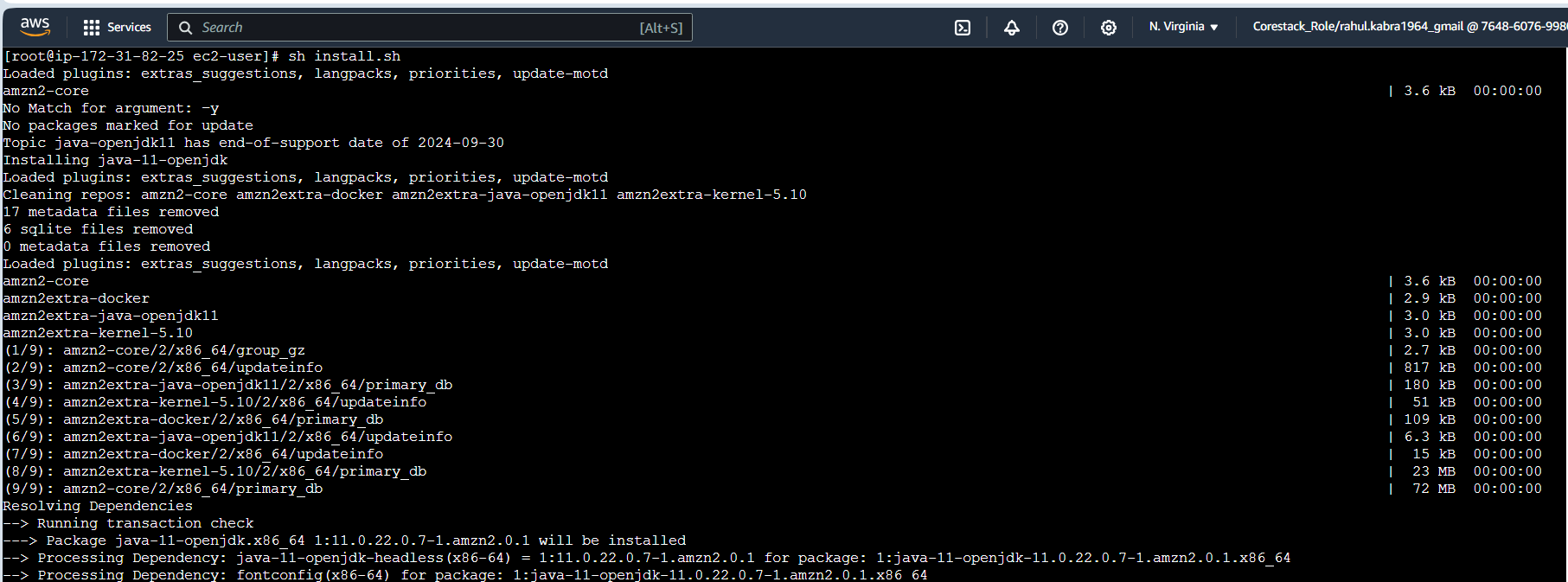
**sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo**

**sudo yum -y install terraform**



3.5 Run below command to execute

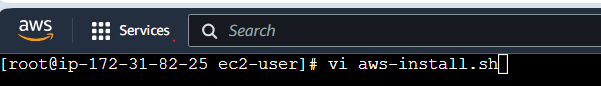
**sh install.sh**

****

****

3.6 Create file install aws-install.sh

**vi aws-install.sh**

****

3.7 Type the below command in the aws-install.sh

**#!/bin/bash**

**# Check if the required parameters are provided**

**if [ $# -lt 3 ]; then**

**echo "Usage: $0 <access\_key> <secret\_key> <region>"**

**exit 1**

**fi**

**#Install asw cli**

**sudo yum install awscli**

**curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"**

**sudo yum install unzip**

**unzip awscliv2.zip**

**sudo ./aws/install --update**

**# Access and use the runtime parameters**

**access\_key=$1**

**secret\_key=$2**

**region=$3**

**# Set AWS credentials and default region**

**aws configure set aws\_access\_key\_id $access\_key**

**aws configure set aws\_secret\_access\_key $secret\_key**

**aws configure set default.region $region**

**#Install Kubectl**

**curl -LO "https://dl.k8s.io/release/v1.23.6/bin/linux/amd64/kubectl"**

**sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl**

**chmod +x kubectl**

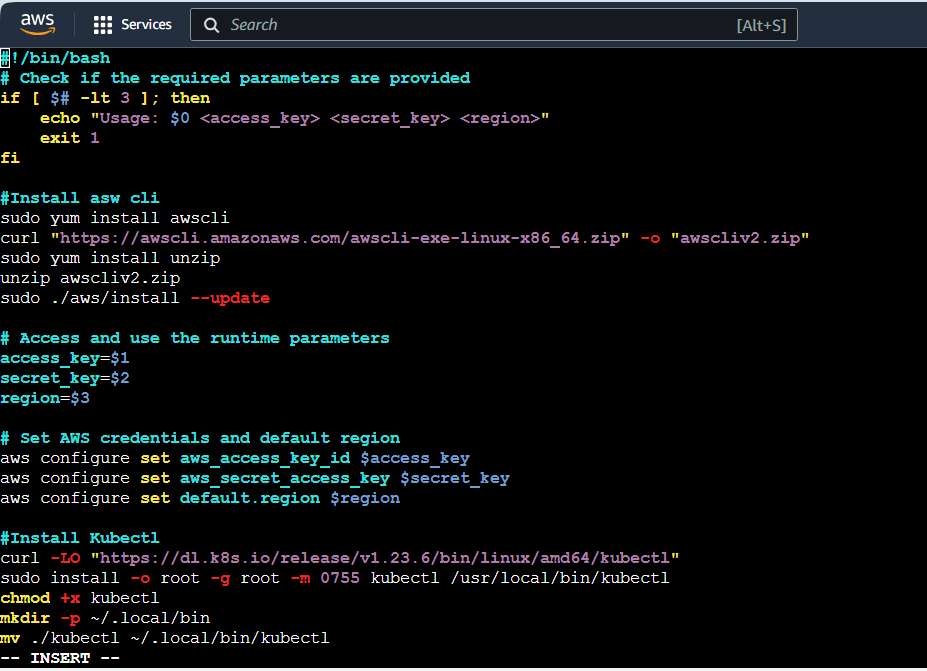
**mkdir -p ~/.local/bin**

**mv ./kubectl ~/.local/bin/kubectl**

**#Set env variable**

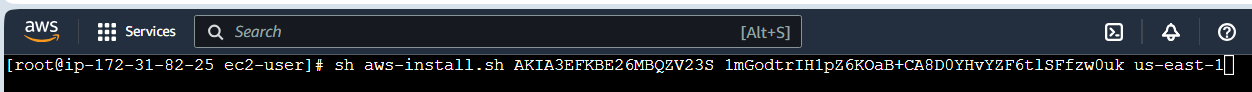
**echo PATH="$PATH:~/.local/bin/"**

**echo "export PATH=$PATH:~/.local/bin/" >> ~/.bashrc**

****

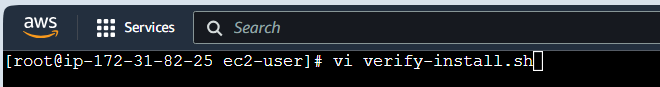
3.8 Run below command to execute

**sh aws-install.sh <access-key> <secret-access-key> us-east-1**

****

3.9 Create file install verify-install.sh

**vi verify-install.sh**

****

3.10 Type the below command in the verify-install.sh , to verify if installed correctly

**#!/bin/bash**

**#Verify installations**

**echo “Java version : “ && java -version**

**echo “======================================”**

**echo “Jenkins Status : “ && sudo systemctl status jenkins**

**echo “======================================”**

**echo “Docker version : “ && docker--version**

**echo “======================================”**

**echo “Maven version : “ && mvn --version**

**echo “======================================”**

**echo “Ansible version : “ && ansible --version**

**echo “======================================”**

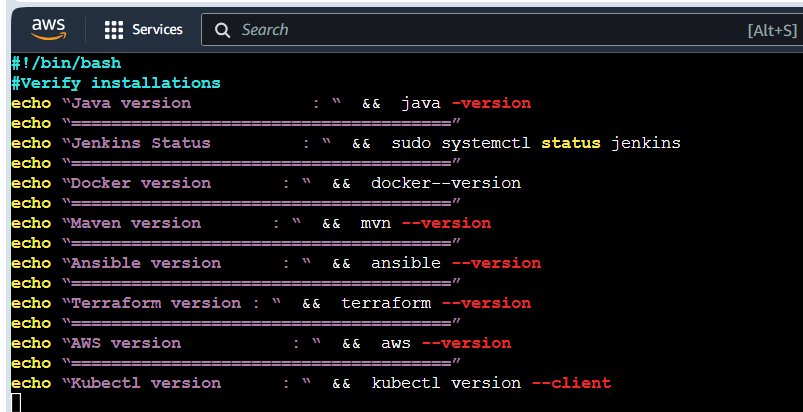
**echo “Terraform version : “ && terraform --version**

**echo “======================================”**

**echo “AWS version : “ && aws --version**

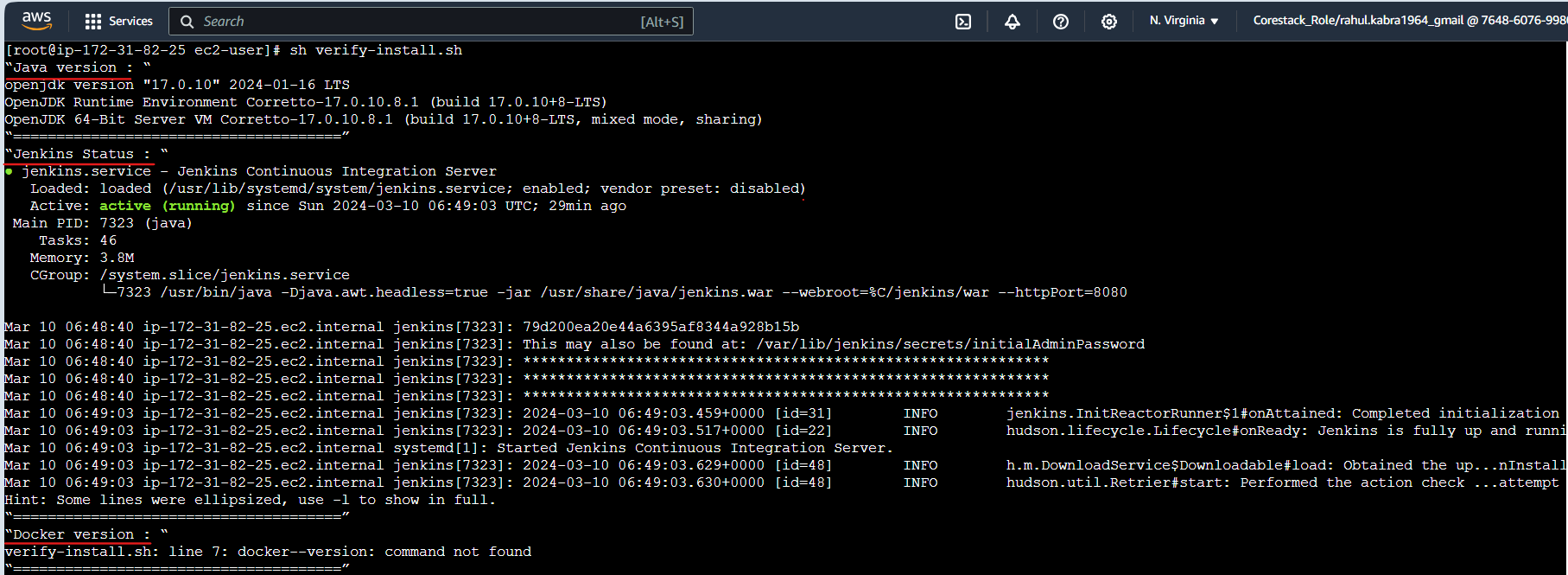
**echo “======================================”**

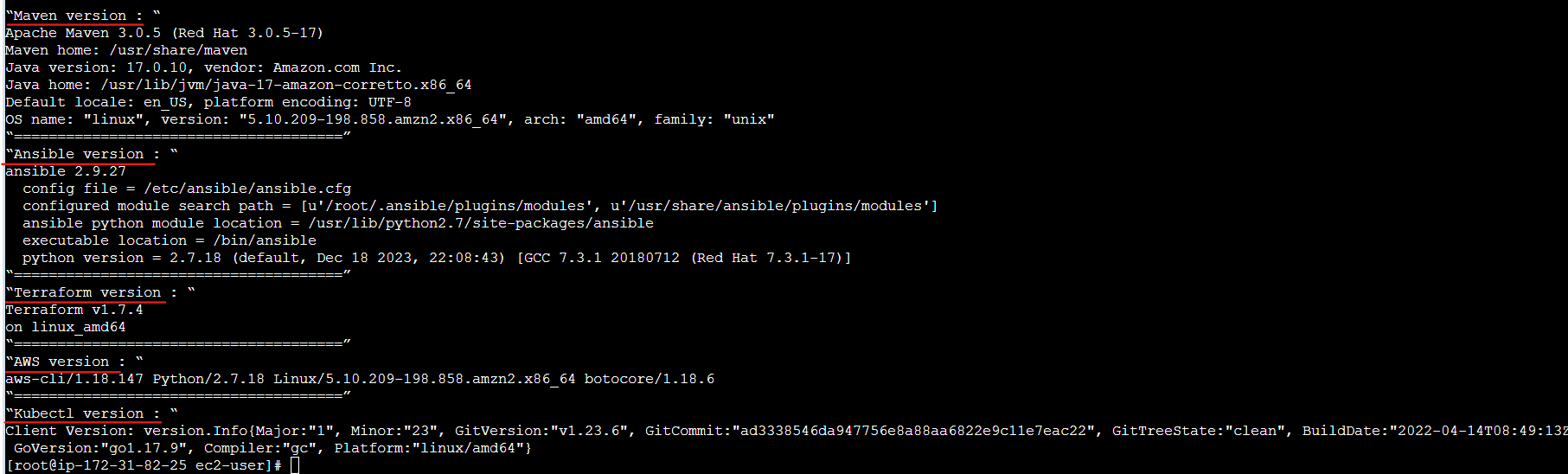
**echo “Kubectl version : “ && kubectl version --client**

****

3.11 Run below command to execute

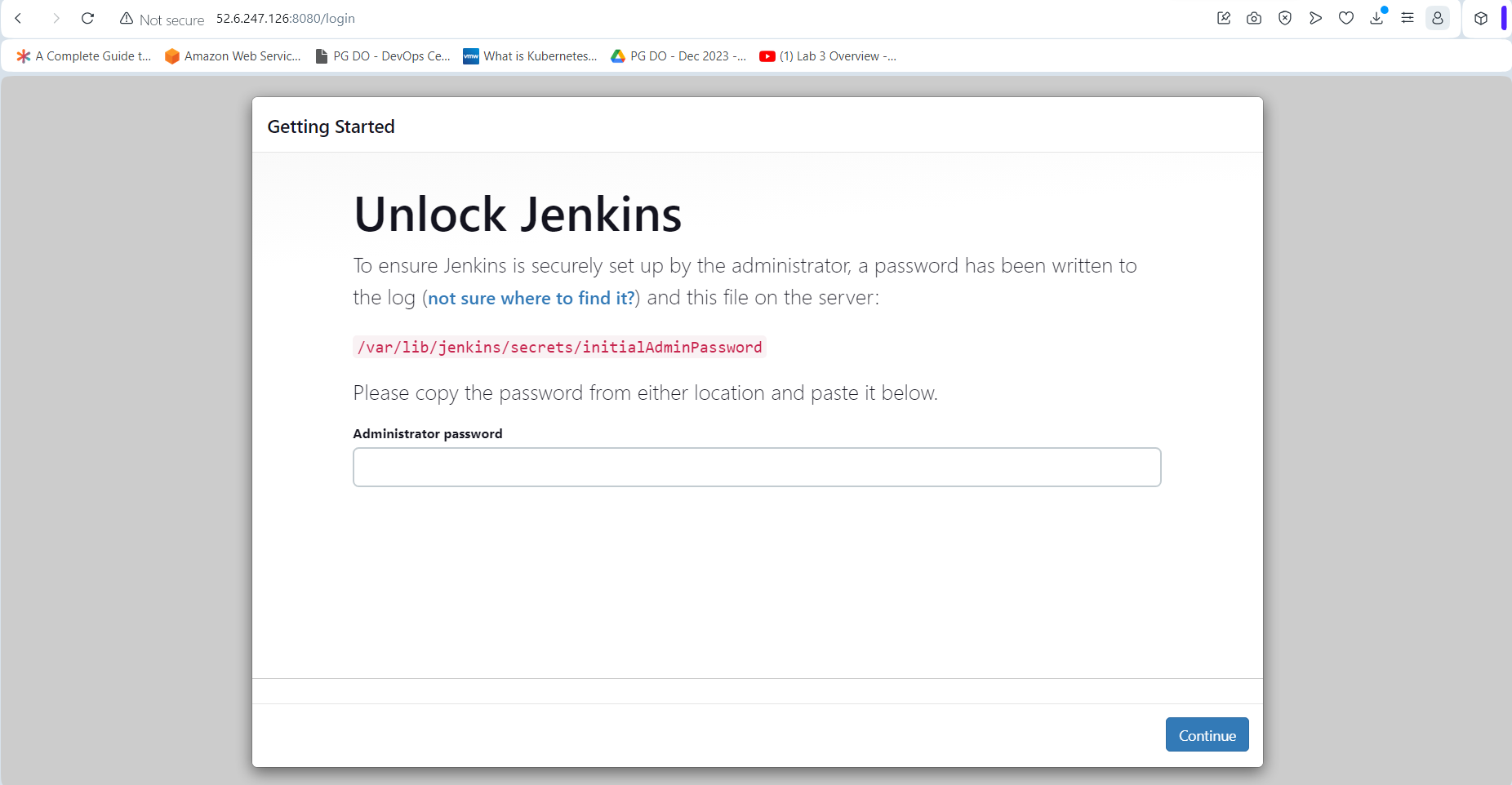
**sh verify-install.sh**

****

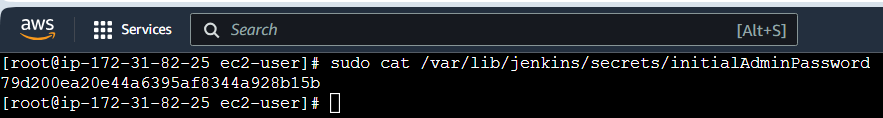
****

**Step 4 : Accessing Jenkins**

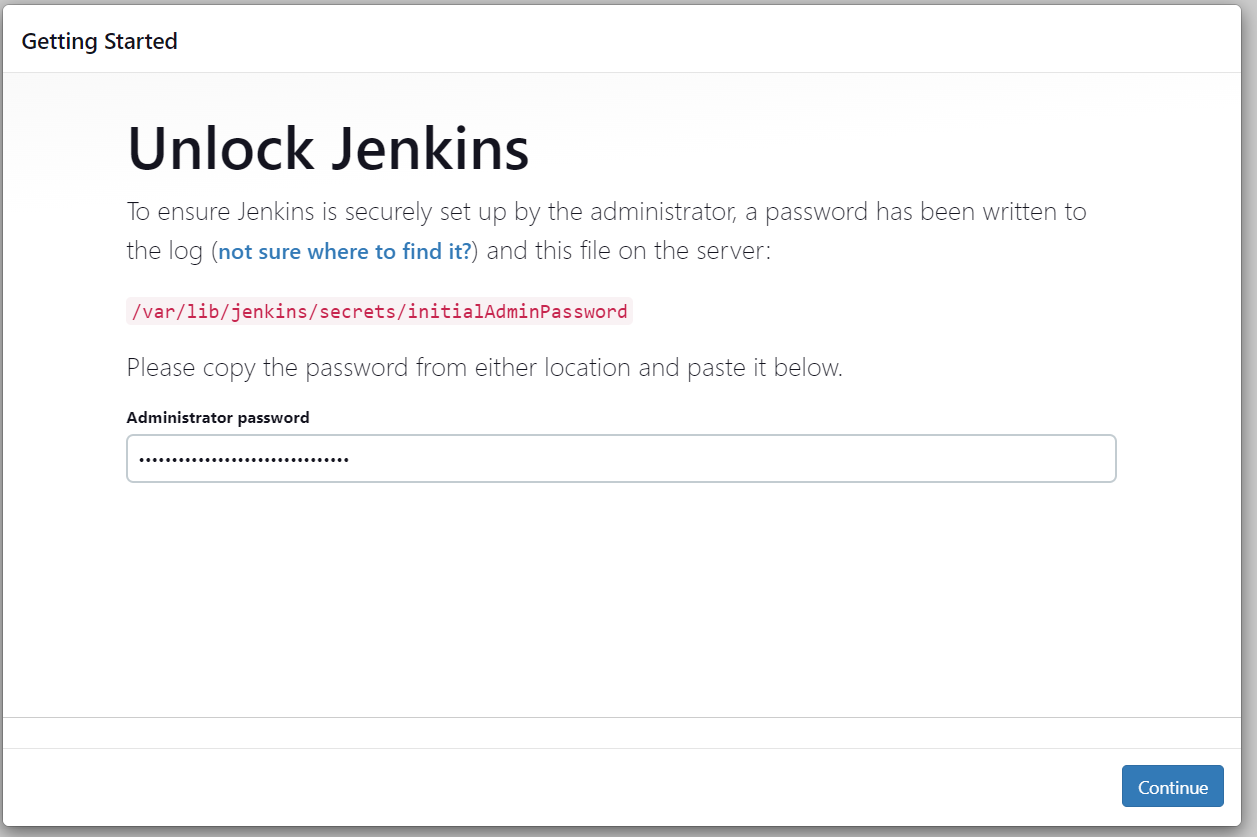
* 1. Open  [**http://52.6.247.126:8080**](http://3.88.187.77:8080/)**/** in the browser



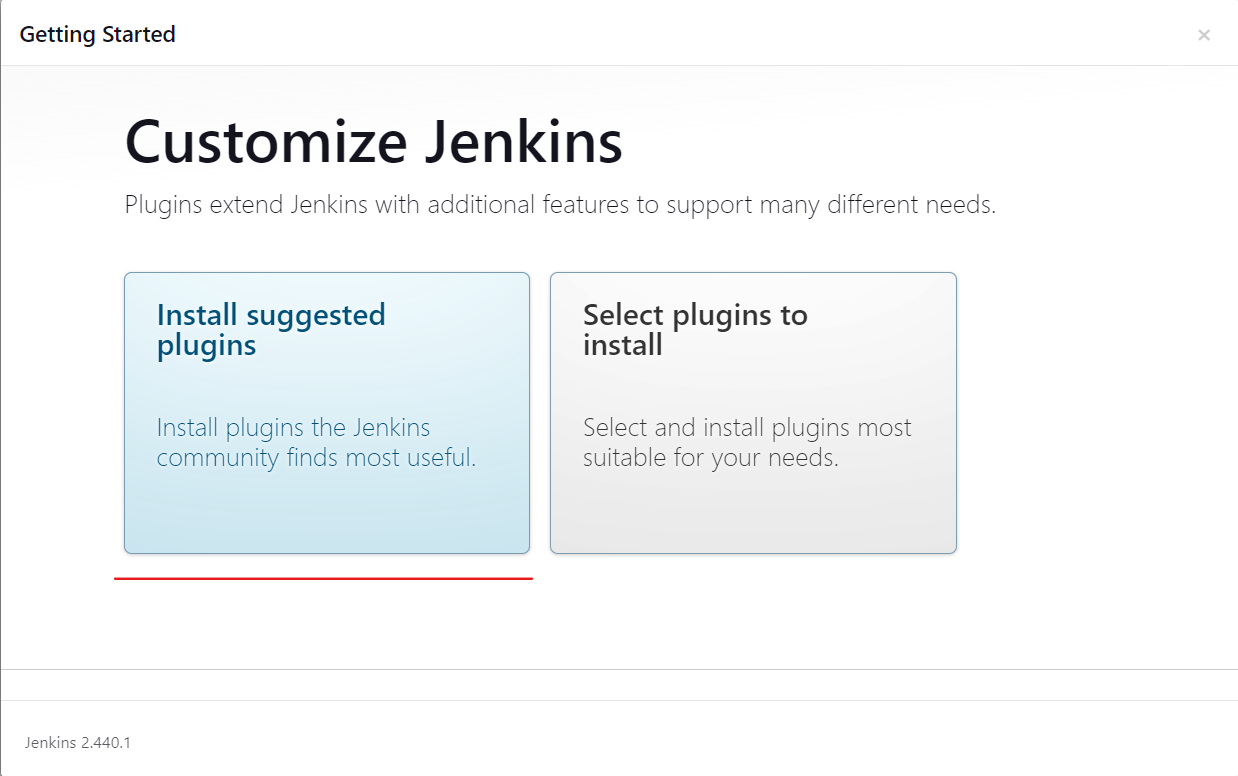
* 1. In your terminal run the following command: ***sudo cat /var/lib/jenkins/secrets/initialAdminPassword***

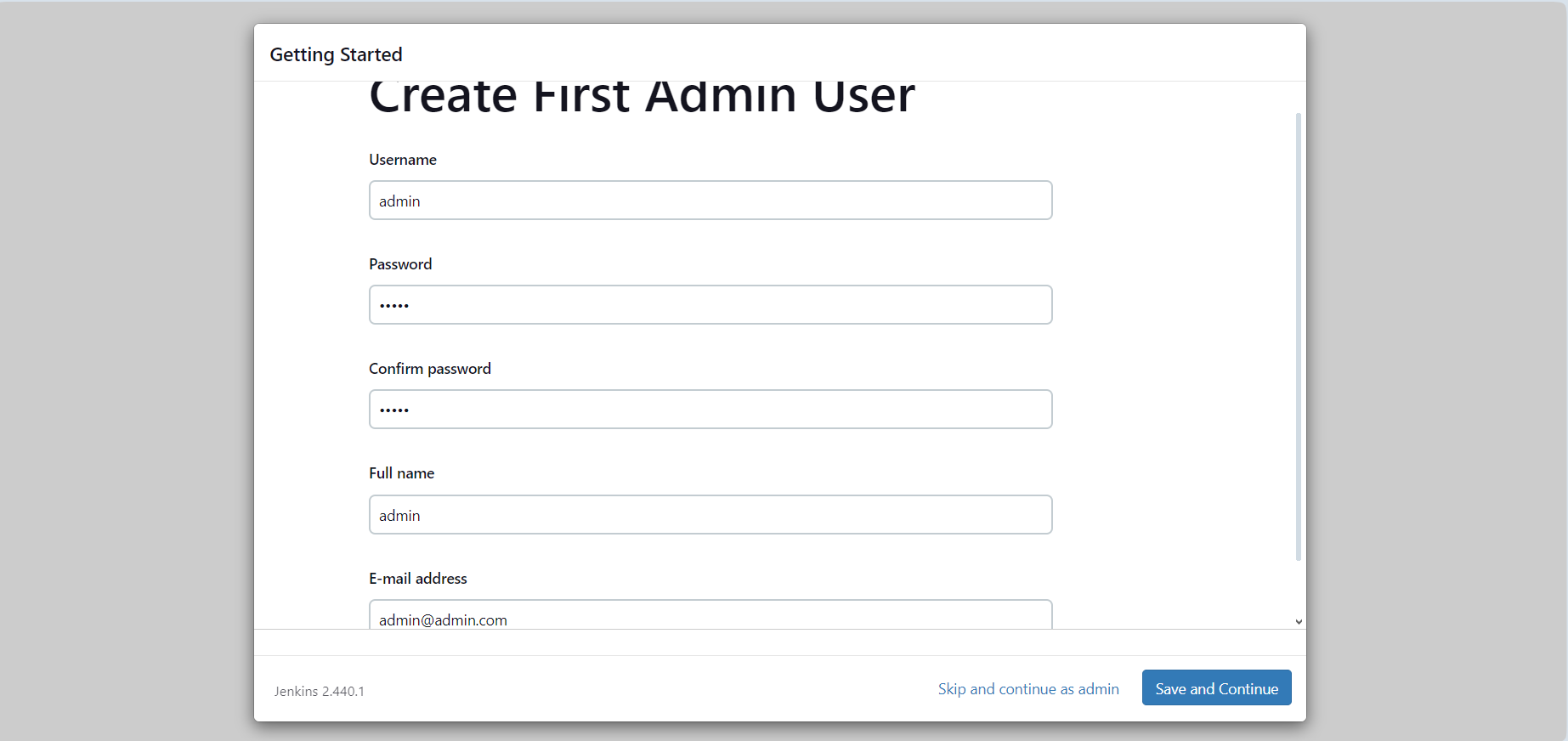


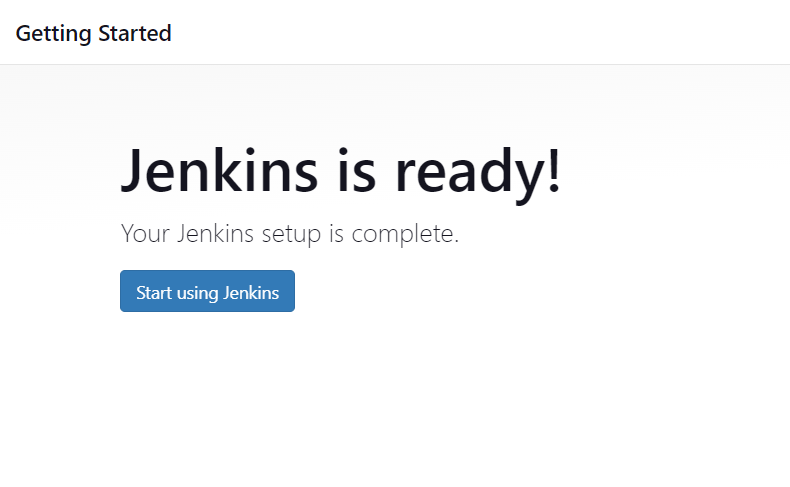
* 1. Copy this password and paste it your Jenkins page in the browser and click continue



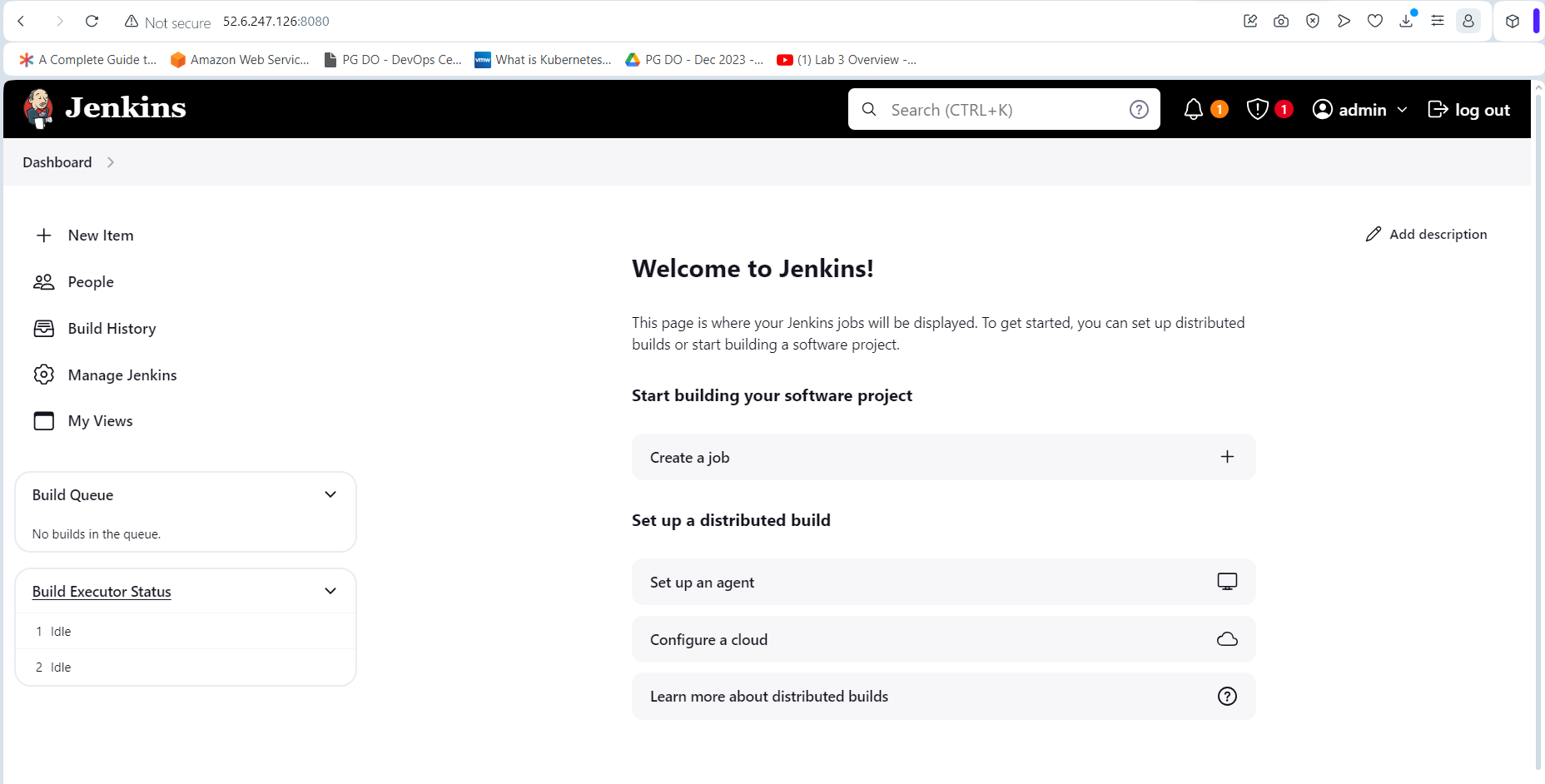
* 1. Now, click on I**nstall the suggested plugins**



* 1. You can create an admin user by filling below details
  2. In the Instance configuration page, click on the **Start using Jenkins** button.

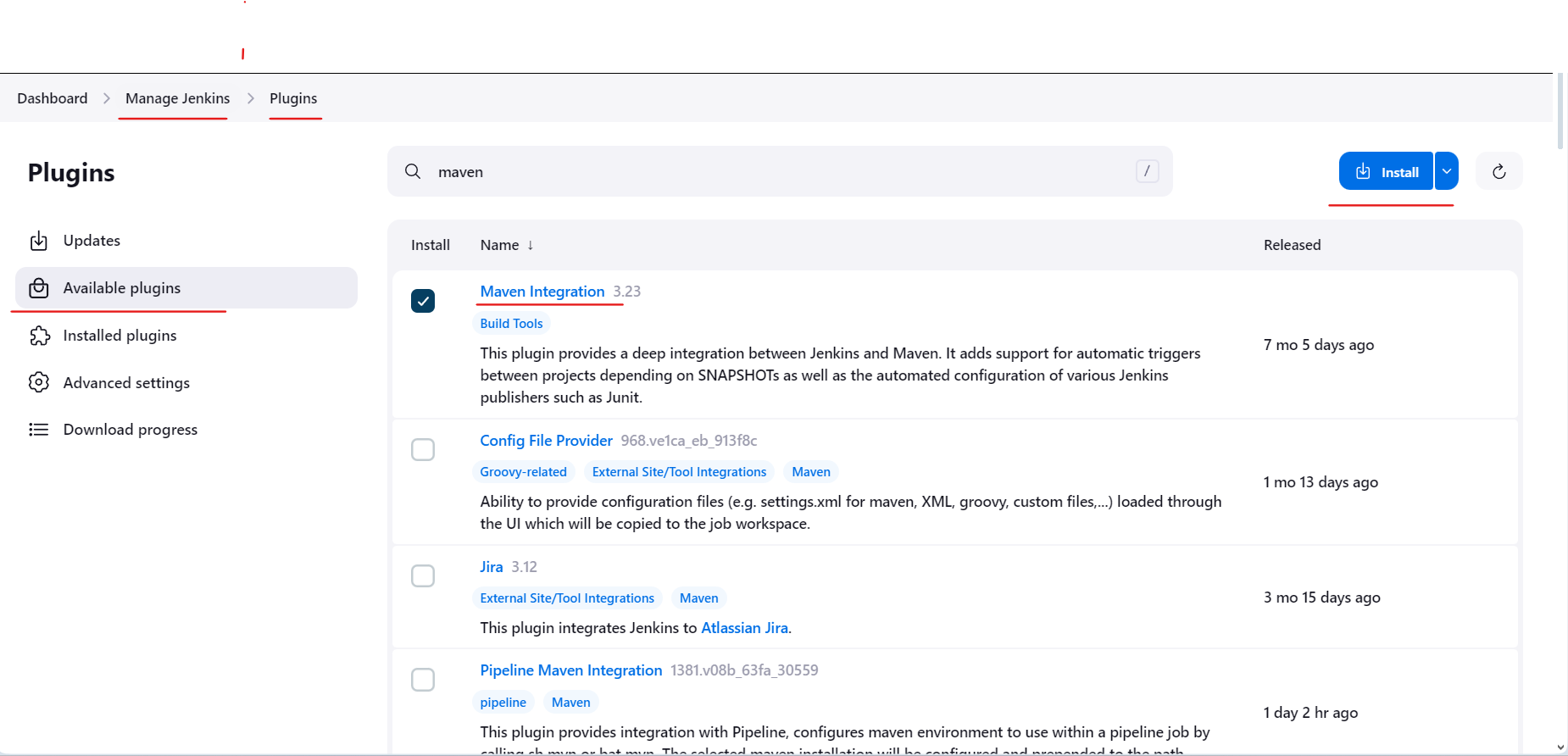


* 1. Now, you can work with Jenkins as shown in the screenshot below



**Step 5 : Installing Plugins and configuring Jenkins**

1. Go to Manage Jenkins -> Plugins -> Available Plugins -> Maven and then click install



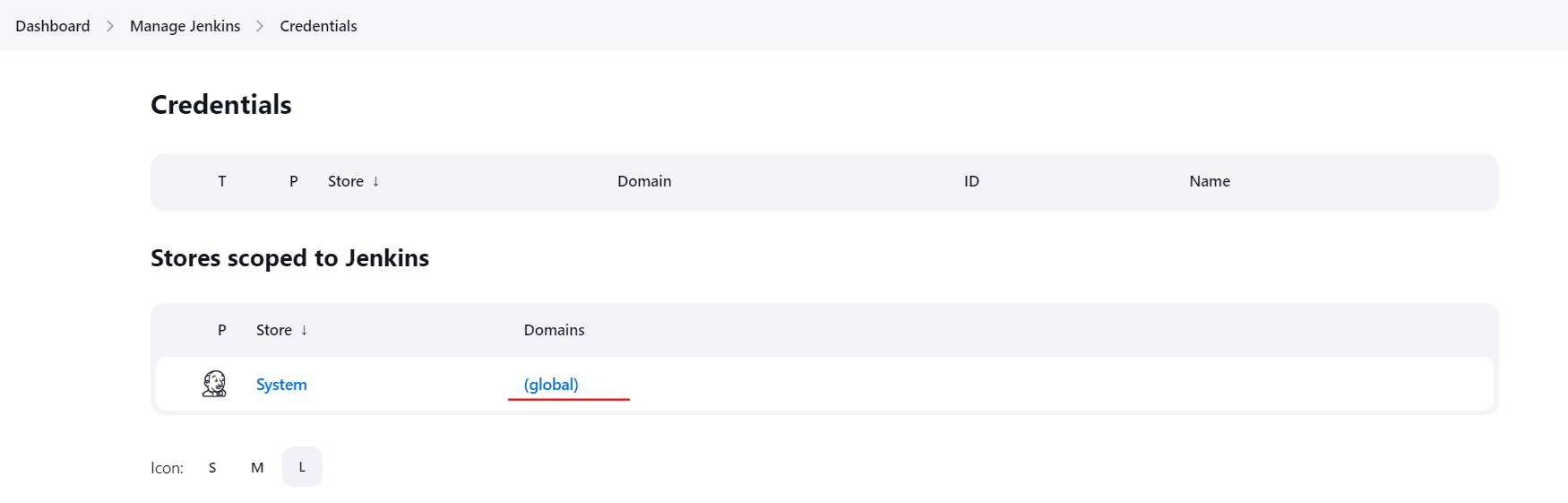
1. Similarly install the following plugins

**Ansible**

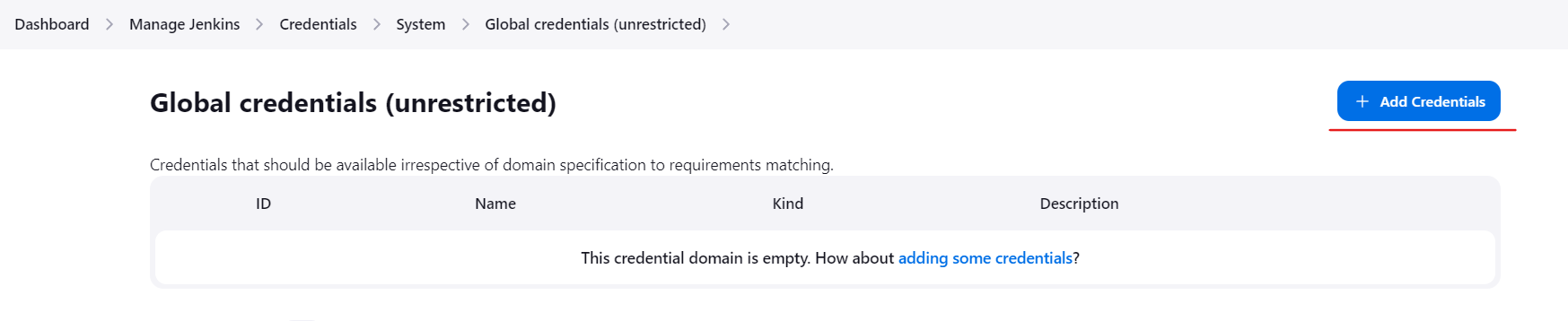
**Docker Pipeline / Docker**

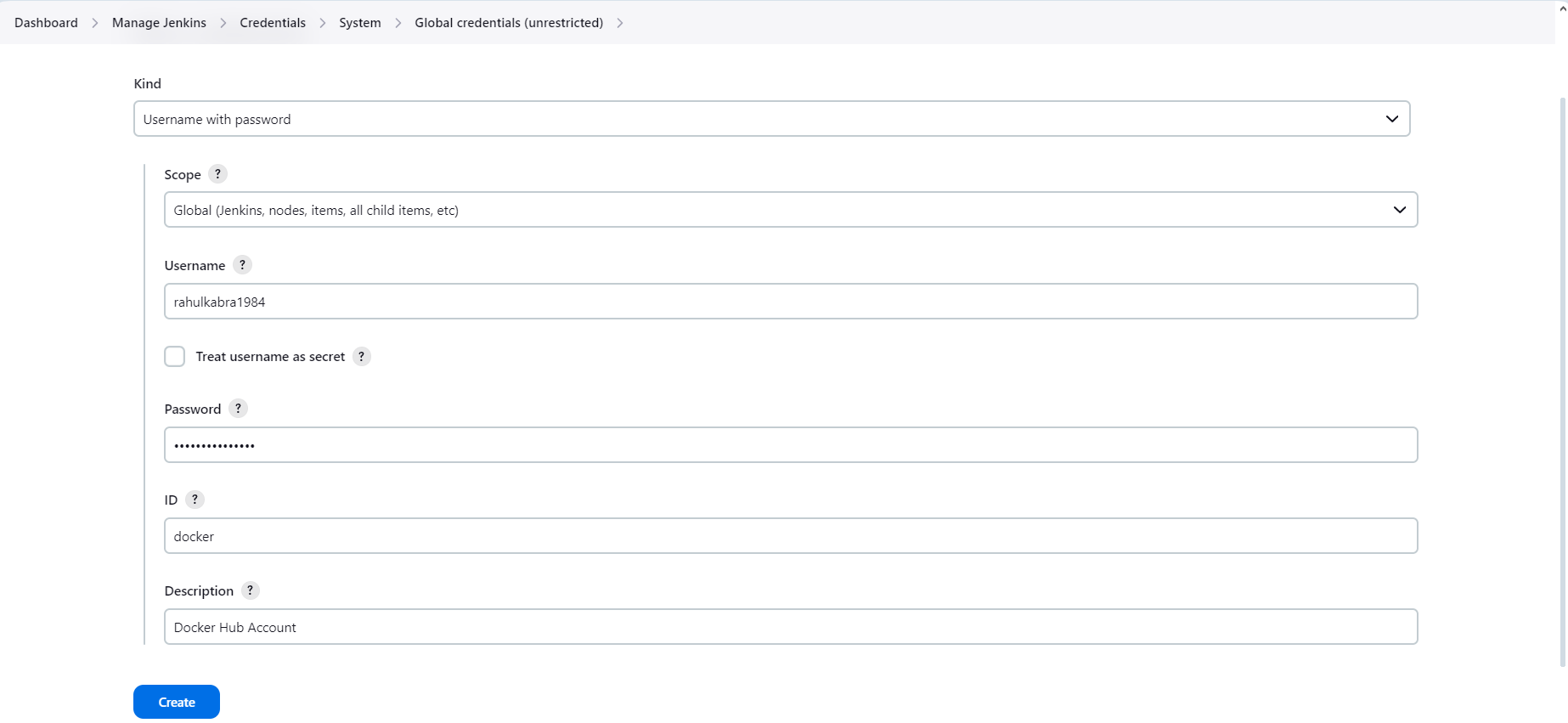
**HTML Publisher**

1. Now navigate to Manage Jenkins 🡪 Credentials and then select global domain.



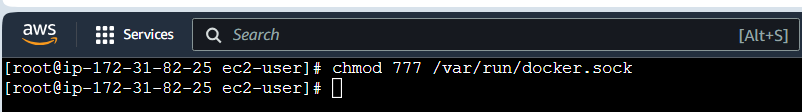
1. Click on new Credential by clicking on Add Credentials to create a new Docker hub credentials as per below details: (credential id should match the one you mentioned in jenkinsfile





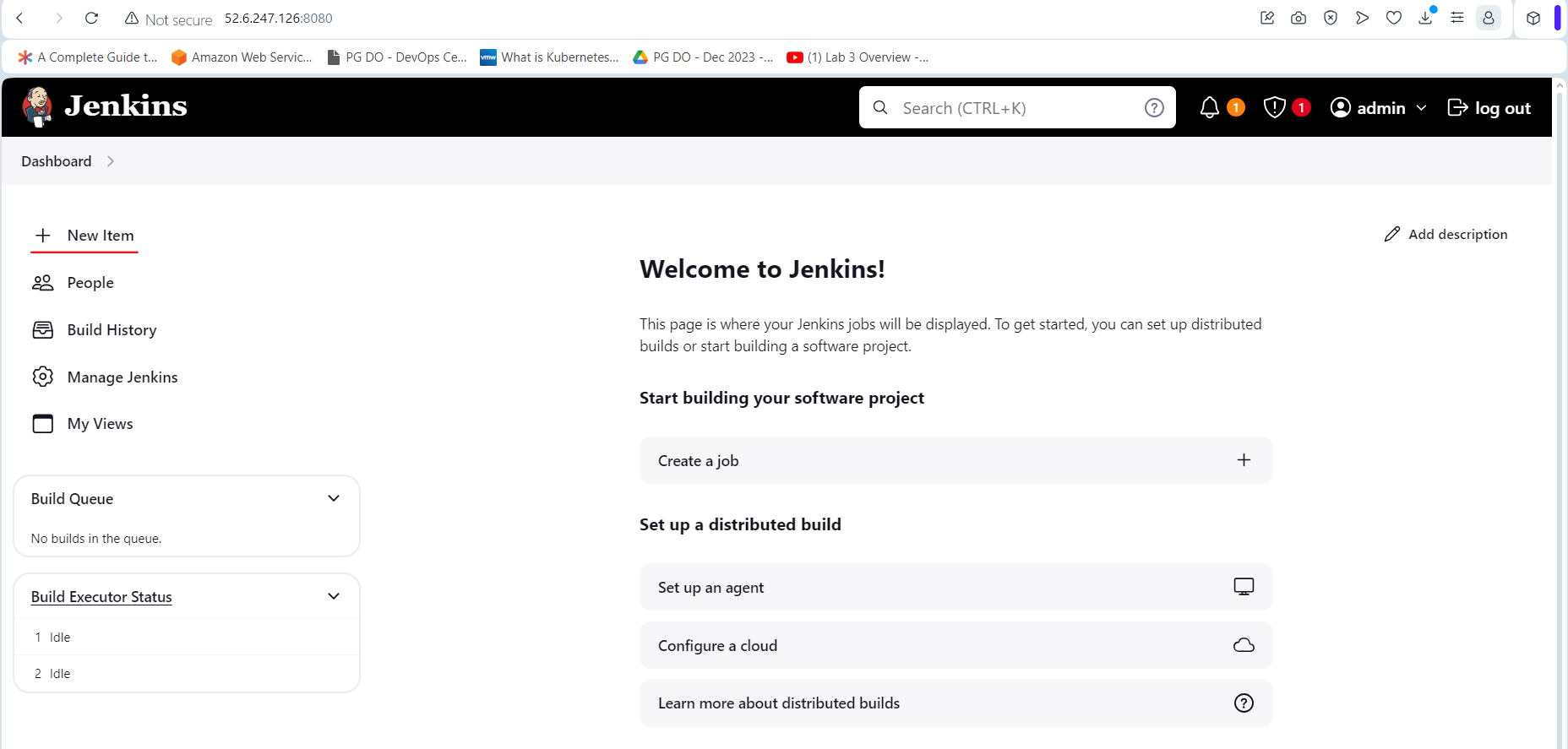
1. Next provide full access to Docker Sock file using below command:

**chmod 777 /var/run/docker.sock**

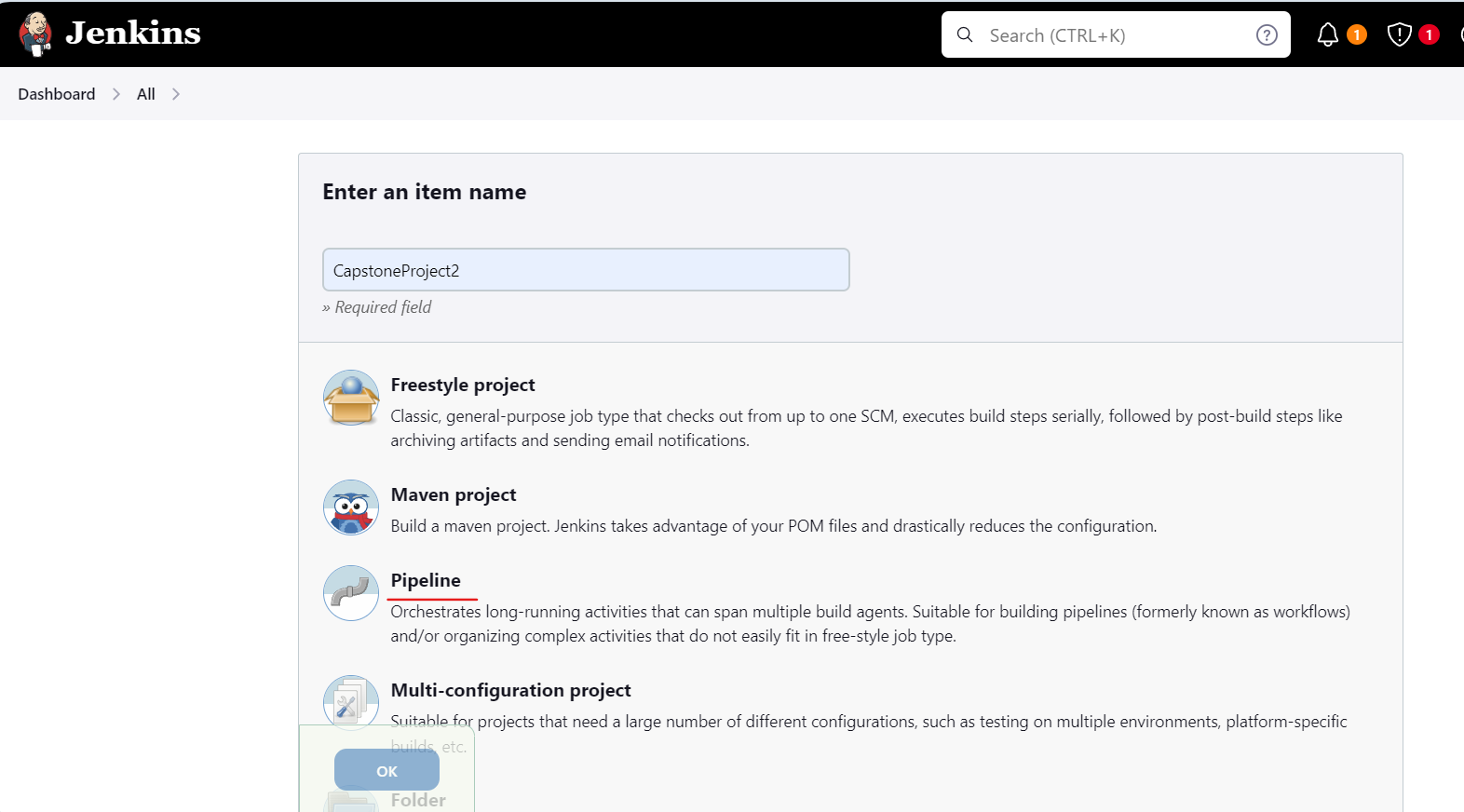
****

**Step 6 : Create Jenkins Pipeline**

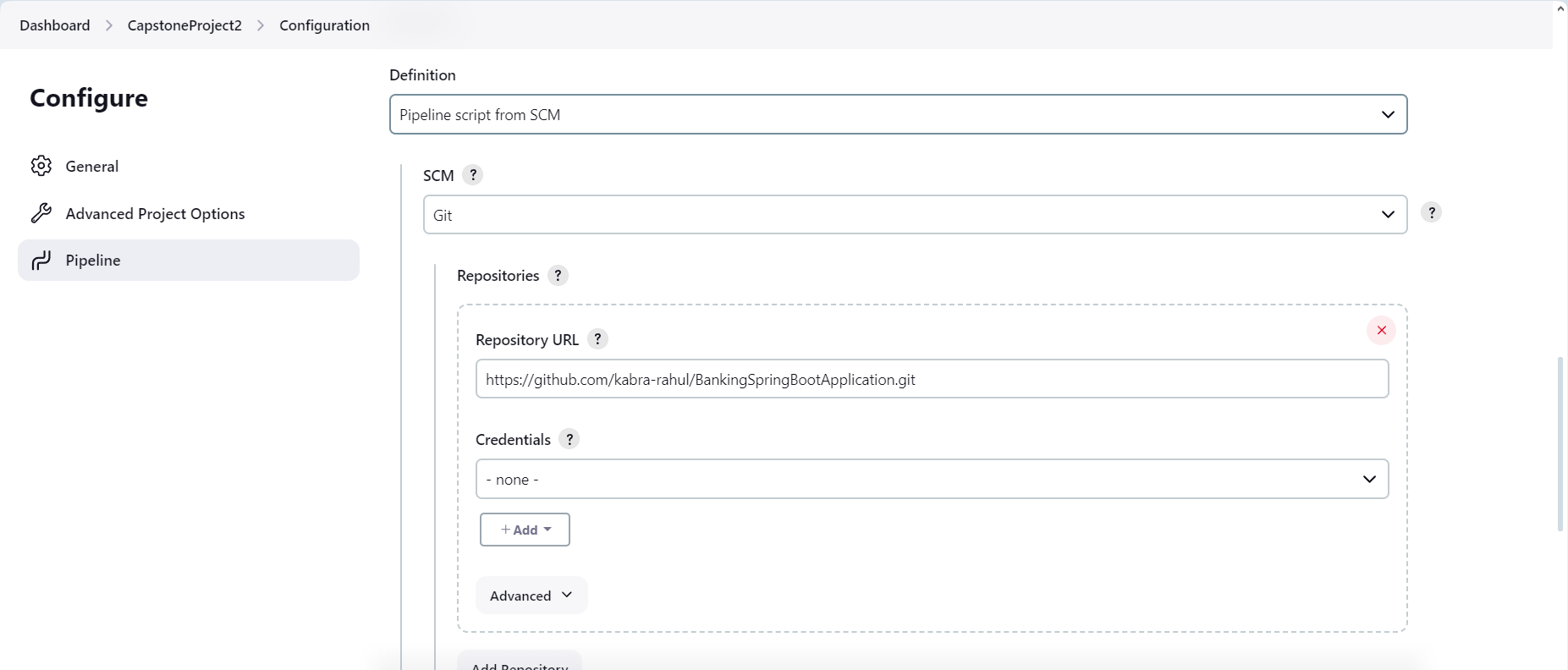
1. Access Jenkins application and click on New Item to create a new Jenkins job:

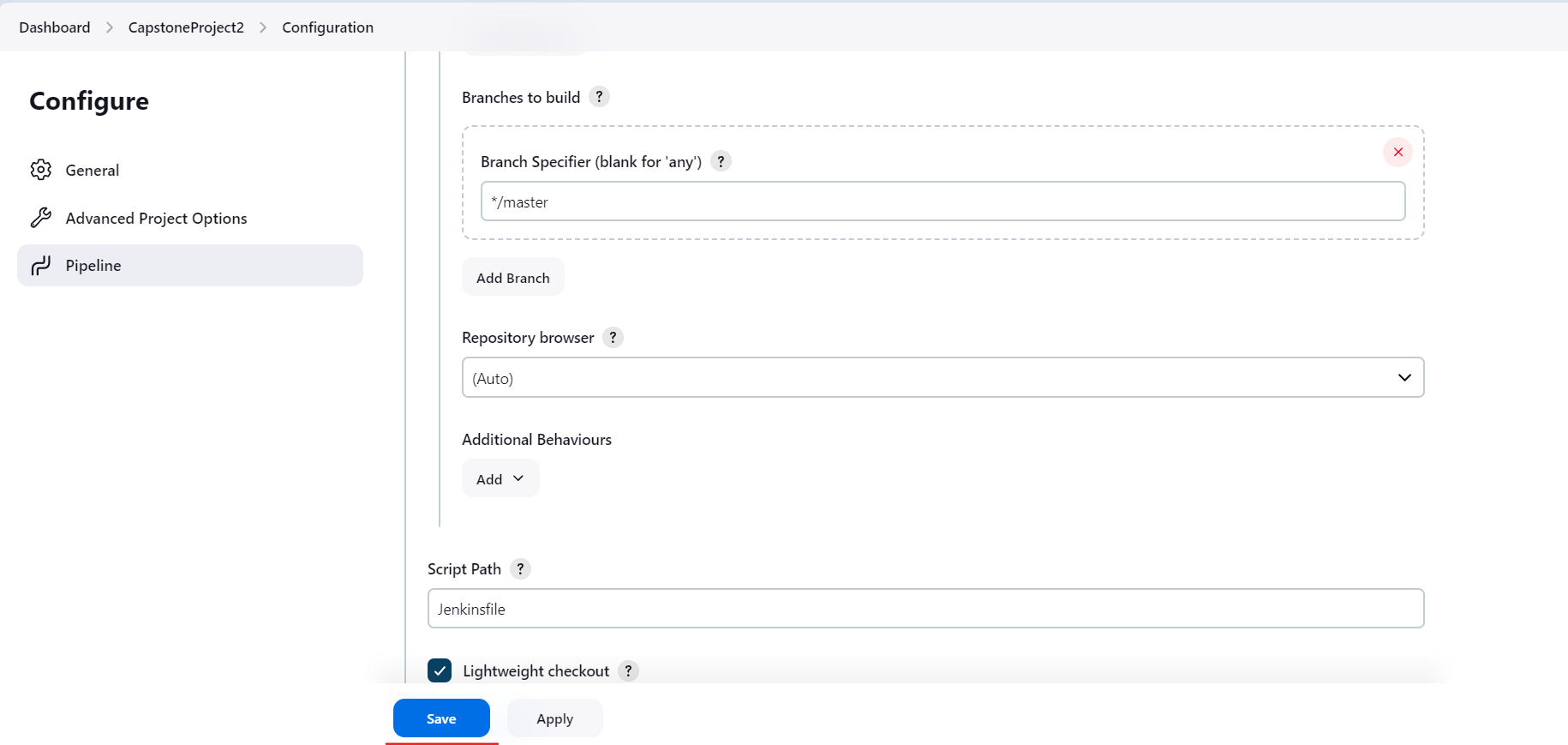


1. Select desired Jenkins pipeline job type and fill in job name



1. Once Clicked on Ok you will be navigated to Jenkins job configuration page where we can provide Jenkins job details such as parameters, Jenkins pipeline configuration etc.

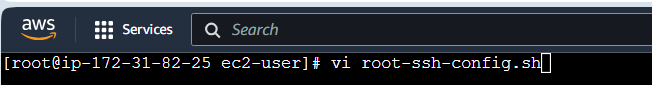




**Step 7 Configuring SSH Connection in EC2 Instance**

7.1 In the EC2 instance create file install verify-install.sh

**vi root-ssh-config.sh**

****

7.2 Type the below command in the verify-install.sh

**#!/bin/bash**

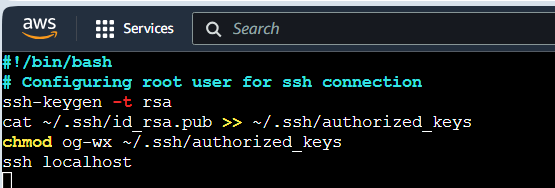
**# Configuring root user for ssh connection**

**ssh-keygen -t rsa**

**cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys**

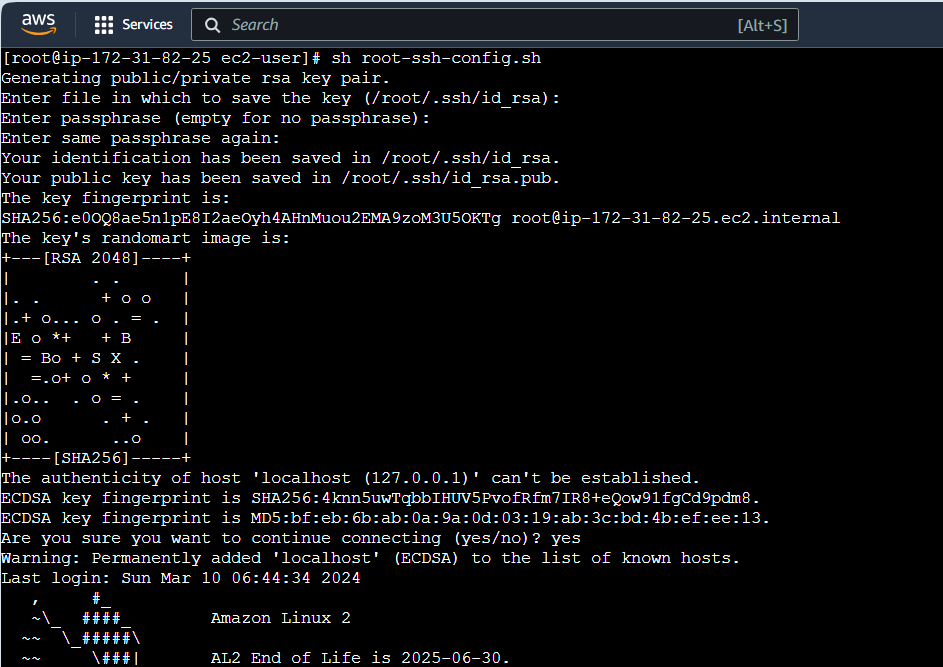
**chmod og-wx ~/.ssh/authorized\_keys**

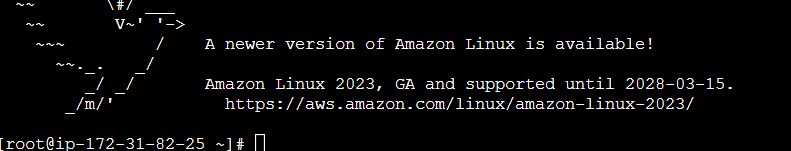
**ssh localhost**

****

7.3 Run below command to execute

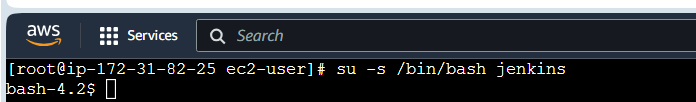
**sh root-ssh-config.sh**

****

****

7.4 Login as Jenkins user by executing the below command

**su -s /bin/bash jenkins**

****

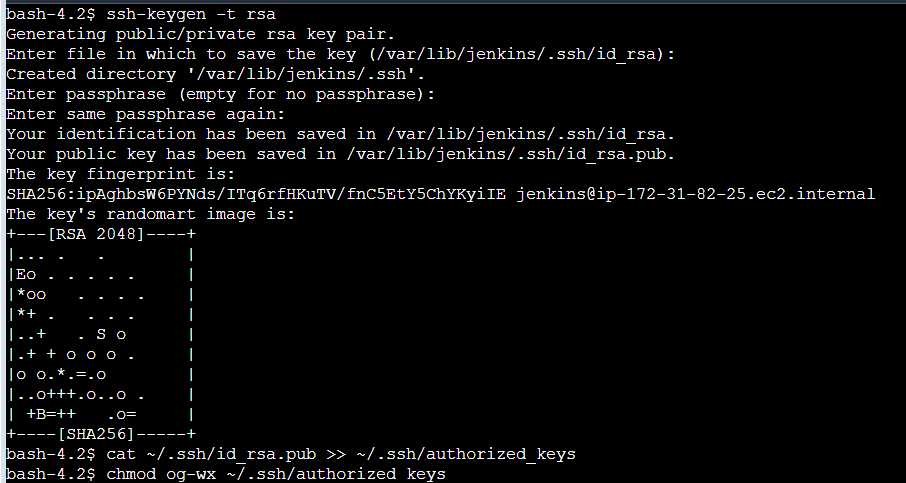
7.5 Execute the below commands in the bash shell

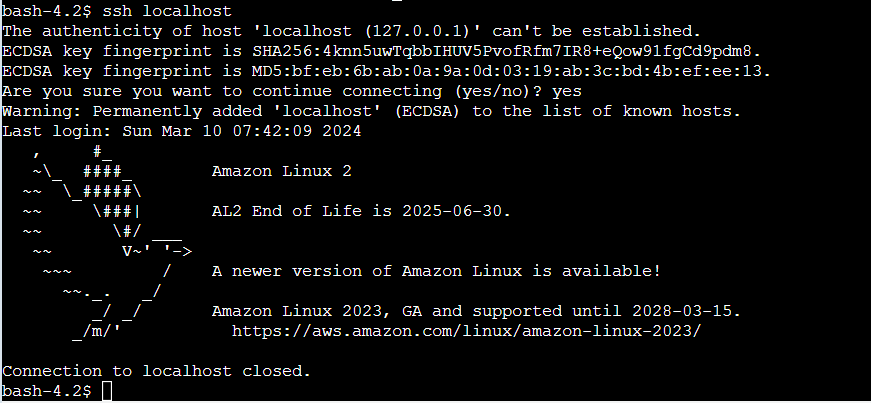
**ssh-keygen -t rsa**

**cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys**

**chmod og-wx ~/.ssh/authorized\_keys**

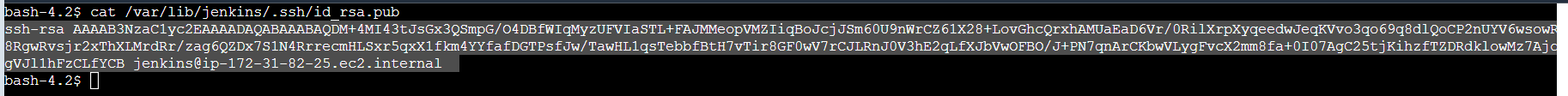
**ssh localhost**

****

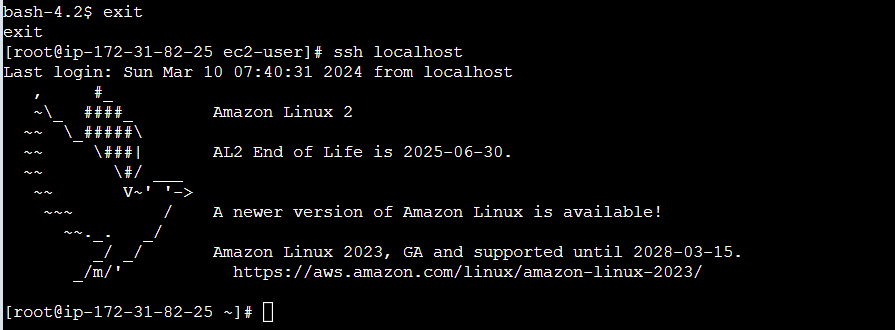
****

7.6 Copy key from id\_rsa.pub

**cat /var/lib/jenkins/.ssh/id\_rsa.pub**

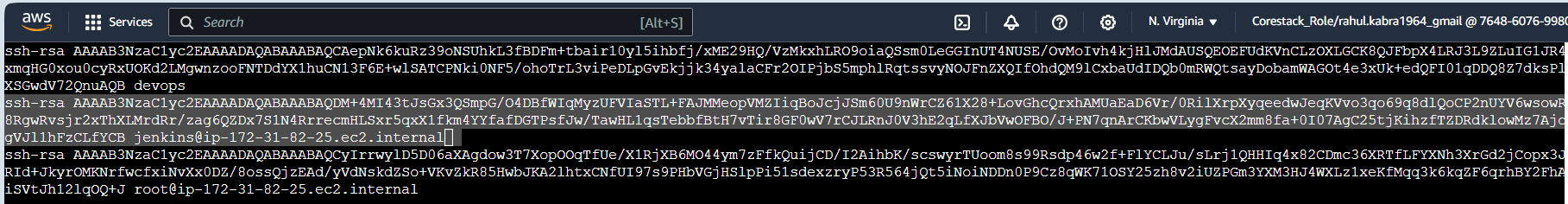


7.7 Exit from bash back to root user and ssh to localhost



7.8 Paste the copied key into authorized\_keys file using below command

**vi .ssh/authorized\_keys**

****

**Step 8 Creating EKS using Terraform**

8.1 Create a file main.tf and paste the below code in it

**provider "aws" {**

**region = "us-east-1"**

**access\_key = "AKIA3EFKBE26MBQZV23S"**

**secret\_key = "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"**

**}**

**# Creating VPC**

**resource "aws\_vpc" "eks\_vpc" {**

**cidr\_block = "10.0.0.0/16"**

**tags = {**

**Name = "dev-vpc"**

**}**

**}**

**# Creating Internet Gateway and attach it to VPC**

**resource "aws\_internet\_gateway" "eks\_internet\_gateway" {**

**vpc\_id = aws\_vpc.eks\_vpc.id**

**tags = {**

**Name = "dev-igw"**

**}**

**}**

**# Creating Public Subnet**

**resource "aws\_subnet" "public\_subnet\_1a" {**

**vpc\_id = aws\_vpc.eks\_vpc.id**

**cidr\_block = "10.0.1.0/24"**

**availability\_zone = "us-east-1a"**

**map\_public\_ip\_on\_launch = true**

**tags = {**

**Name = "public\_subnet\_1a"**

**}**

**}**

**resource "aws\_subnet" "public\_subnet\_1b" {**

**vpc\_id = aws\_vpc.eks\_vpc.id**

**cidr\_block = "10.0.2.0/24"**

**availability\_zone = "us-east-1b"**

**map\_public\_ip\_on\_launch = true**

**tags = {**

**Name = "public\_subnet\_1b"**

**}**

**}**

**# Creating Route Table**

**resource "aws\_route\_table" "public\_route\_table" {**

**vpc\_id = aws\_vpc.eks\_vpc.id**

**route {**

**cidr\_block = "0.0.0.0/0"**

**gateway\_id = aws\_internet\_gateway.eks\_internet\_gateway.id**

**}**

**tags = {**

**Name = "public\_route\_table"**

**}**

**}**

**# Associating Public Subnet to route table**

**resource "aws\_route\_table\_association" "public\_subnet\_1a\_route\_table\_association" {**

**subnet\_id = aws\_subnet.public\_subnet\_1a.id**

**route\_table\_id = aws\_route\_table.public\_route\_table.id**

**}**

**resource "aws\_route\_table\_association" "public\_subnet\_1b\_route\_table\_association" {**

**subnet\_id = aws\_subnet.public\_subnet\_1b.id**

**route\_table\_id = aws\_route\_table.public\_route\_table.id**

**}**

**# Create Security Group for the EKS**

**resource "aws\_security\_group" "eks\_security\_group" {**

**name = "SH security group"**

**vpc\_id = aws\_vpc.eks\_vpc.id**

**ingress {**

**description = "SSH access"**

**from\_port = 22**

**to\_port = 22**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**ingress {**

**description = "HTTP access"**

**from\_port = 80**

**to\_port = 80**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**egress {**

**description = "outbound access"**

**from\_port = 0**

**to\_port = 0**

**protocol = -1**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**tags = {**

**Name = "dev-EKS-security-group"**

**}**

**}**

**# Creating IAM role for Master Node**

**resource "aws\_iam\_role" "master" {**

**name = "EKS-Master"**

**assume\_role\_policy = jsonencode({**

**"Version" : "2012-10-17",**

**"Statement" : [**

**{**

**"Effect" : "Allow",**

**"Principal" : {**

**"Service" : "eks.amazonaws.com"**

**},**

**"Action" : "sts:AssumeRole"**

**}**

**]**

**})**

**}**

**# Attaching Policy to IAM role**

**resource "aws\_iam\_role\_policy\_attachment" "eks\_cluster\_policy" {**

**policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"**

**role = aws\_iam\_role.master.name**

**}**

**# Creating IAM role for Worker Node**

**resource "aws\_iam\_role" "worker" {**

**name = "ed-eks-worker"**

**assume\_role\_policy = jsonencode({**

**"Version" : "2012-10-17",**

**"Statement" : [**

**{**

**"Effect" : "Allow",**

**"Principal" : {**

**"Service" : "ec2.amazonaws.com"**

**},**

**"Action" : "sts:AssumeRole"**

**}**

**]**

**})**

**}**

**# Attaching Policy to IAM role**

**resource "aws\_iam\_role\_policy\_attachment" "eks\_worker\_node\_policy" {**

**policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"**

**role = aws\_iam\_role.worker.name**

**}**

**# Attaching Policy to IAM role**

**resource "aws\_iam\_role\_policy\_attachment" "eks\_cni\_policy" {**

**policy\_arn = "arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy"**

**role = aws\_iam\_role.worker.name**

**}**

**# Attaching Policy to IAM role**

**resource "aws\_iam\_role\_policy\_attachment" "ec2\_container\_registry\_read\_only" {**

**policy\_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"**

**role = aws\_iam\_role.worker.name**

**}**

**# Creating EKS Cluster**

**resource "aws\_eks\_cluster" "eks" {**

**name = "kubernetescluster"**

**role\_arn = aws\_iam\_role.master.arn**

**version = "1.28"**

**vpc\_config {**

**endpoint\_private\_access = true**

**endpoint\_public\_access = true**

**subnet\_ids = [aws\_subnet.public\_subnet\_1a.id, aws\_subnet.public\_subnet\_1b.id]**

**}**

**depends\_on = [**

**aws\_iam\_role\_policy\_attachment.eks\_cluster\_policy**

**]**

**}**

**# Creating Worker Node Group**

**resource "aws\_eks\_node\_group" "node-grp" {**

**cluster\_name = aws\_eks\_cluster.eks.name**

**node\_group\_name = "worker-nodes"**

**node\_role\_arn = aws\_iam\_role.worker.arn**

**subnet\_ids = [aws\_subnet.public\_subnet\_1a.id,aws\_subnet.public\_subnet\_1b.id]**

**capacity\_type = "ON\_DEMAND"**

**disk\_size = 20**

**instance\_types = ["t2.medium"]**

**ami\_type = "AL2\_x86\_64"**

**version = "1.28"**

**labels = {**

**role = "worker\_nodes"**

**}**

**scaling\_config {**

**desired\_size = 1**

**max\_size = 3**

**min\_size = 1**

**}**

**depends\_on = [**

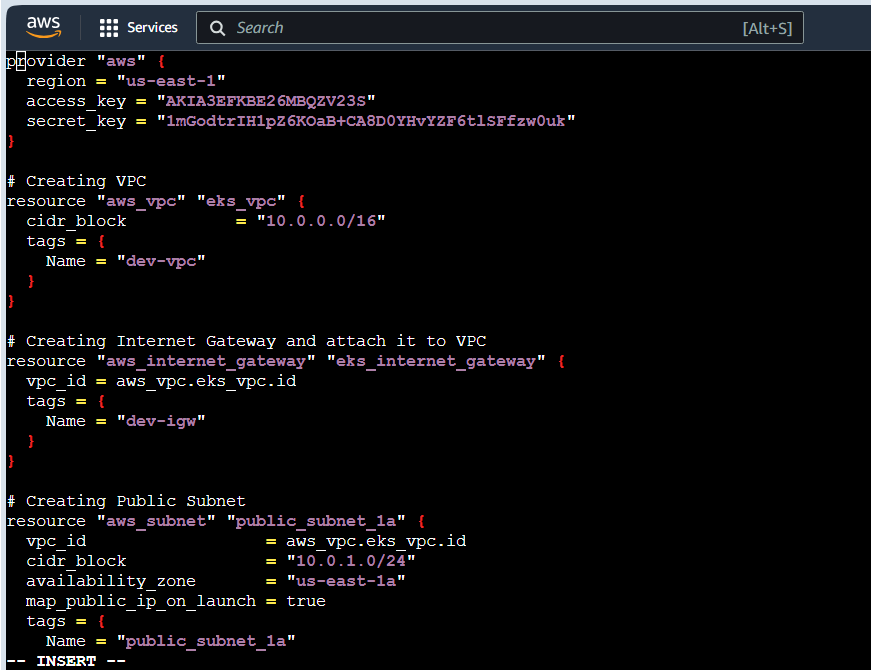
**aws\_iam\_role\_policy\_attachment.eks\_worker\_node\_policy,**

**aws\_iam\_role\_policy\_attachment.eks\_cni\_policy,**

**aws\_iam\_role\_policy\_attachment.ec2\_container\_registry\_read\_only**

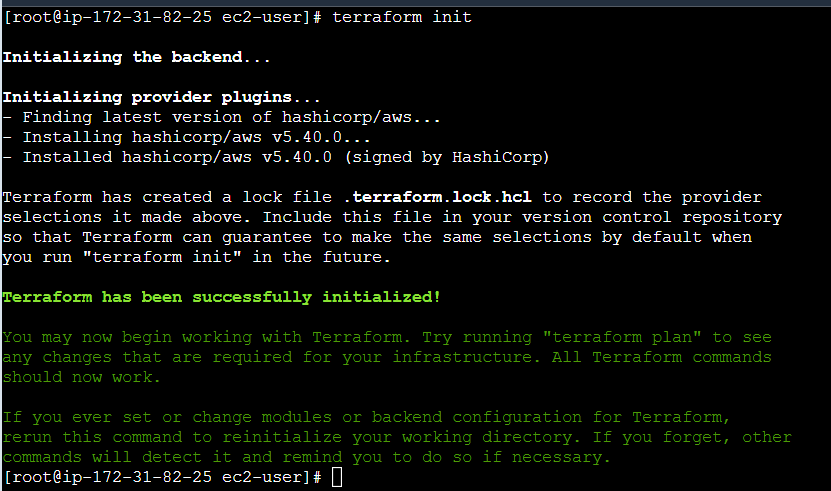
**]**

**}**

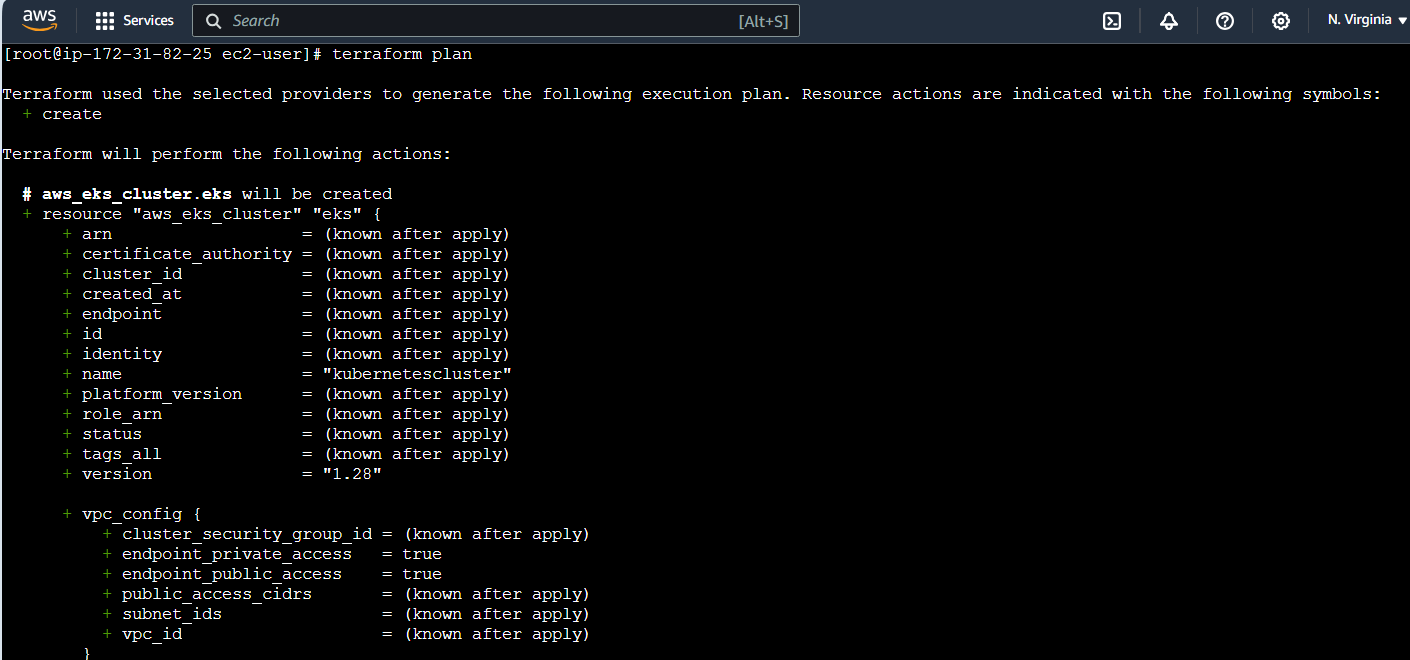
****

8.2 Now execute the main.tf file using below command

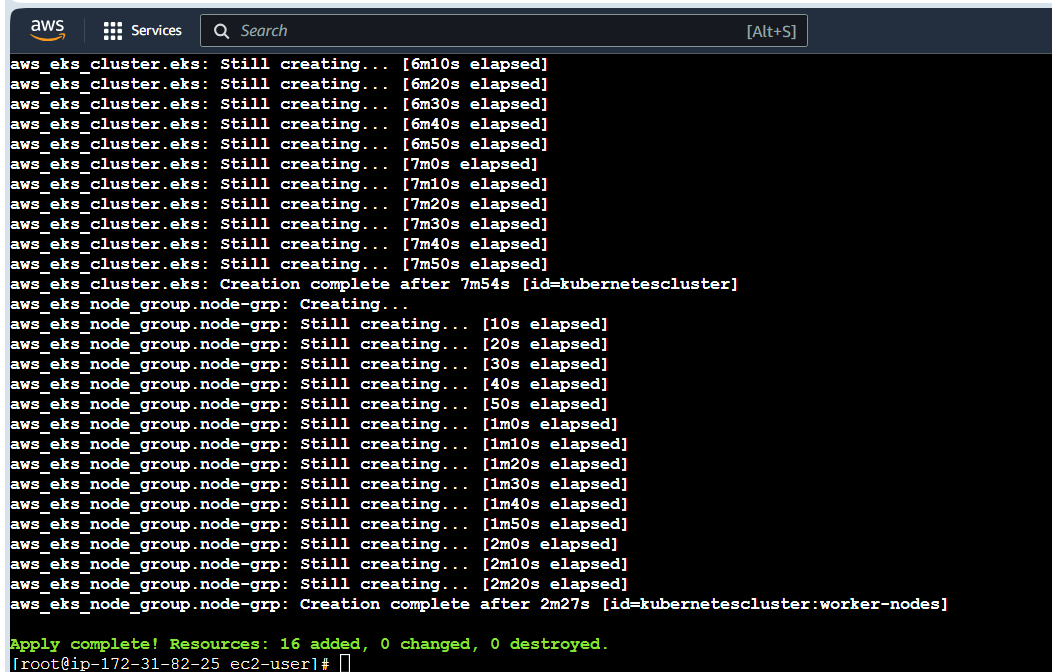
**terraform init**

****

**terraform plan**

****

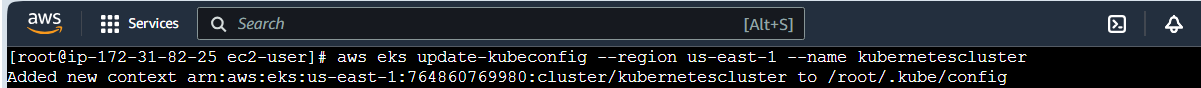
**terraform apply**

****

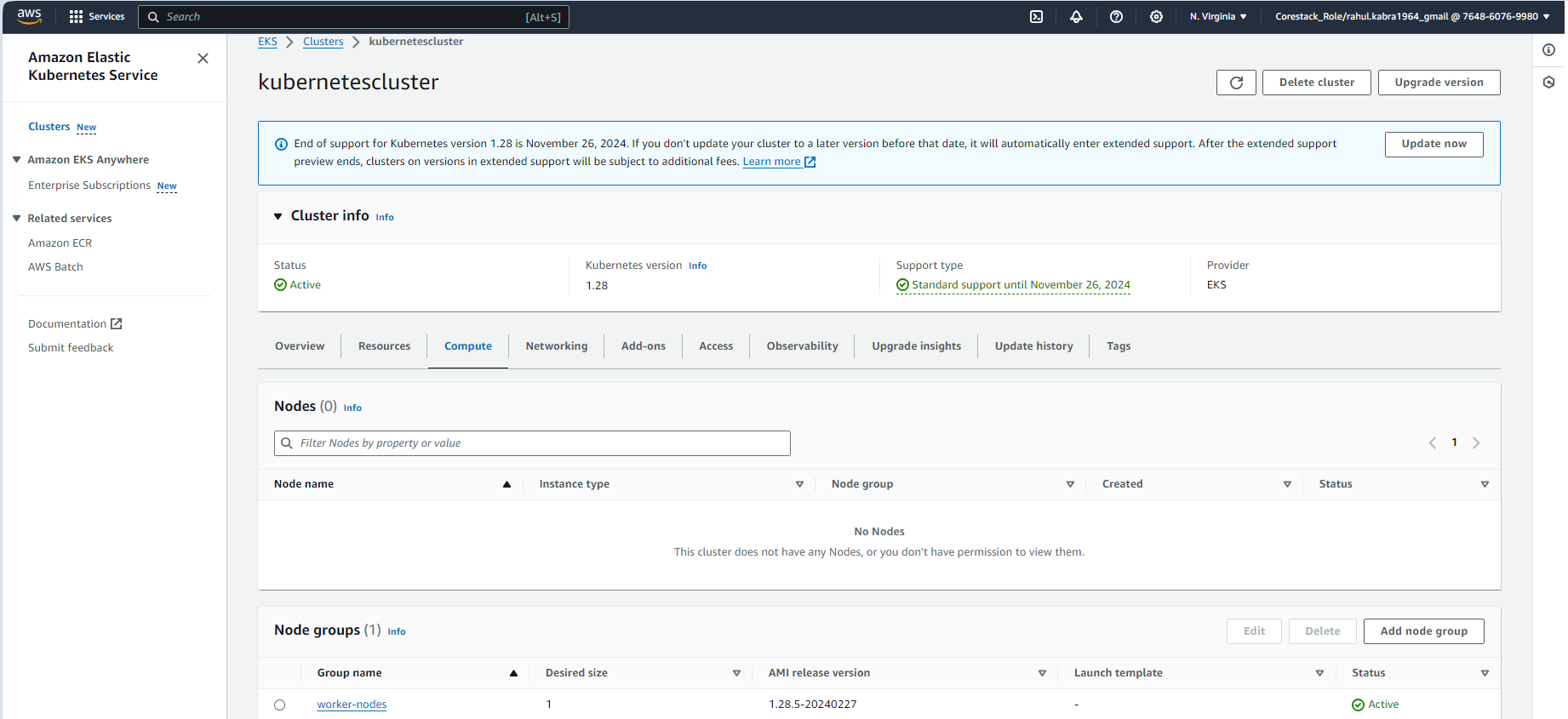
8.3 Now configure the kubeconfig file to connect to the EKS cluster by executing below

command

**aws eks update-kubeconfig --region us-east-1 --name kubernetescluster**

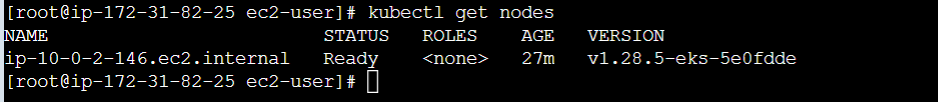
****

8.4 Verify in aws console if the eks cluster is created



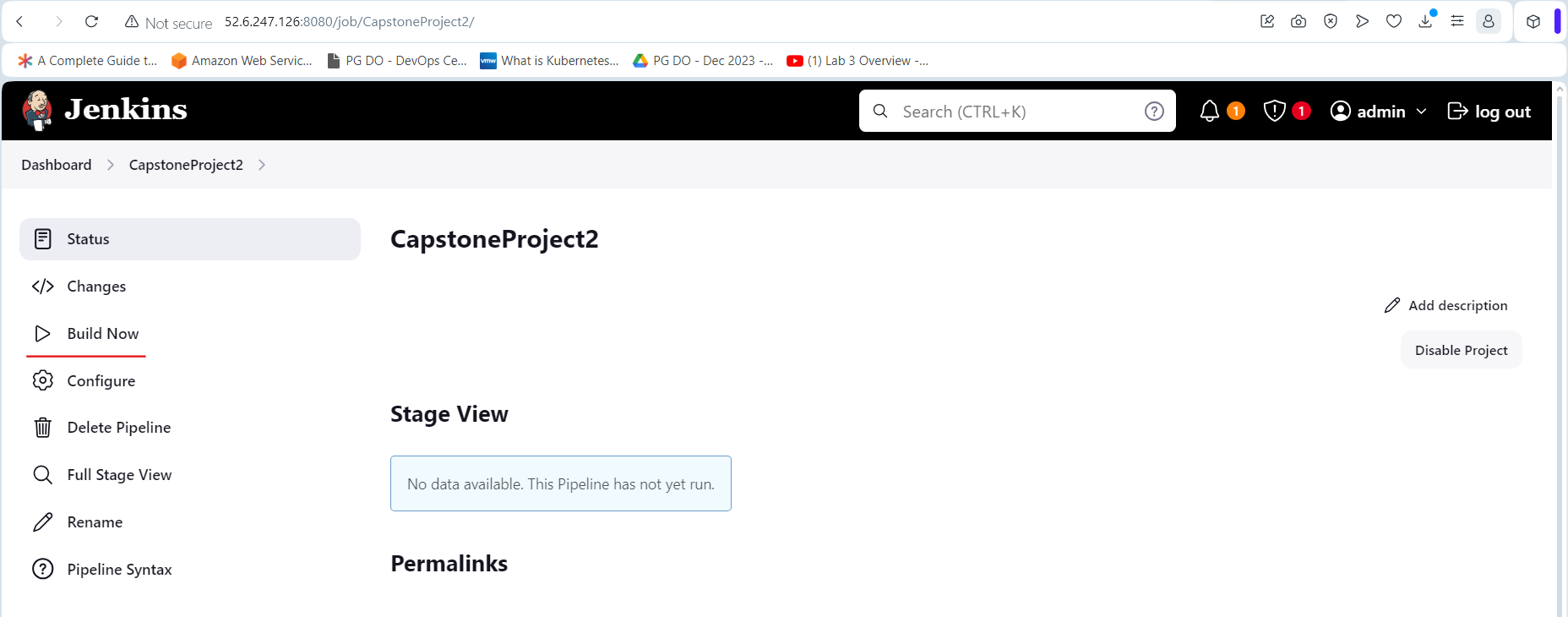
8.5 Verify nodes in the shell terminal using below command

**Kubectl get nodes**

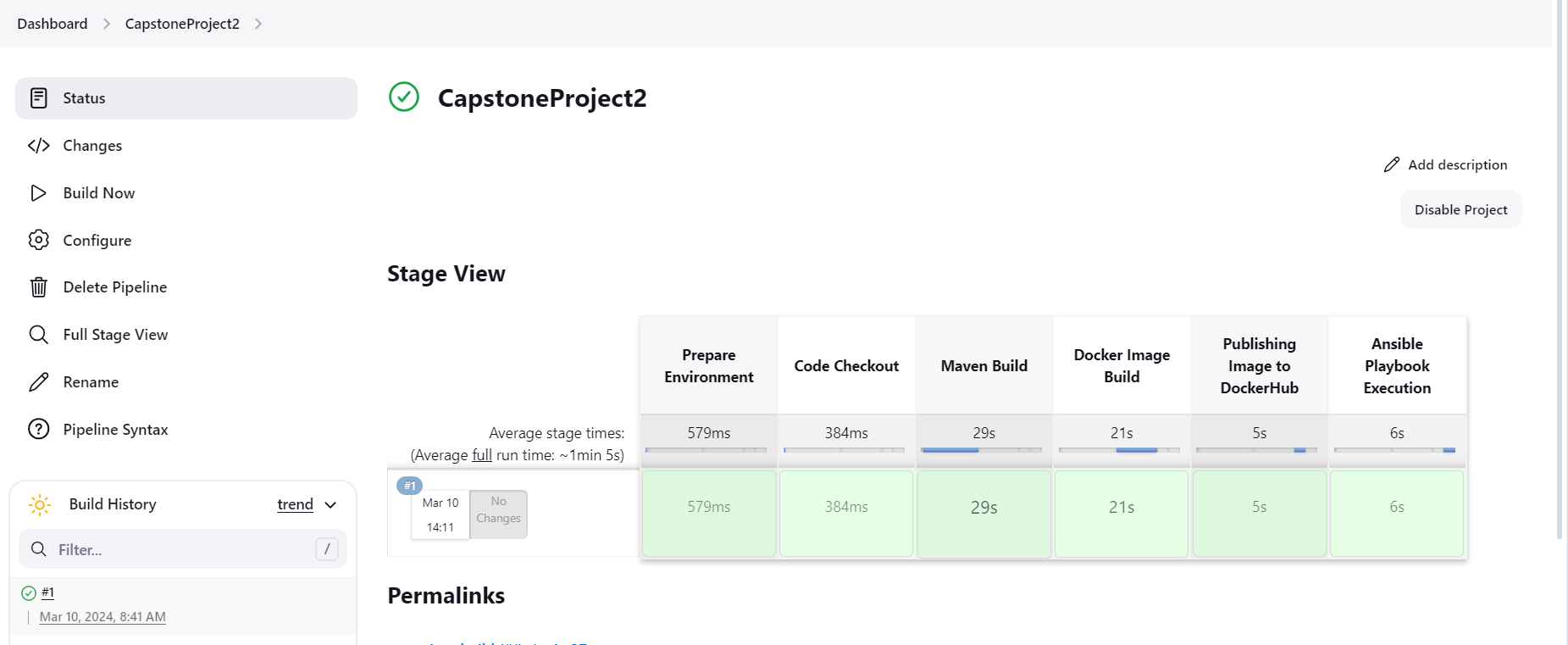


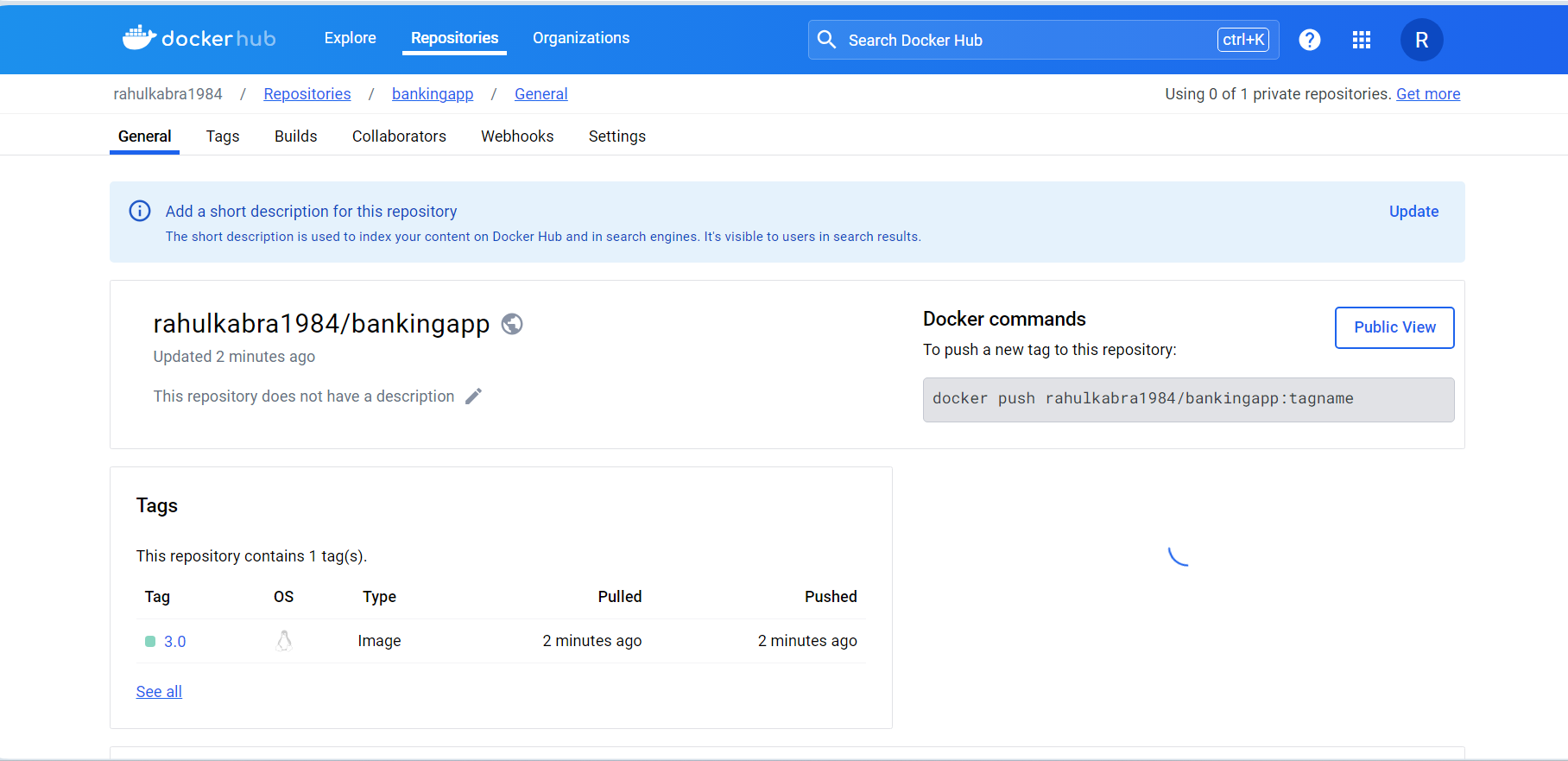
**Step 9 : Execute Jenkins Build and Validate it**

9.1 Navigate to Jenkins job created in above step and click on Build now to start running build for Jenkins job created.



9.2 Once Jenkins build is completed validate if Docker image really gets uploaded to Docker hub or not.





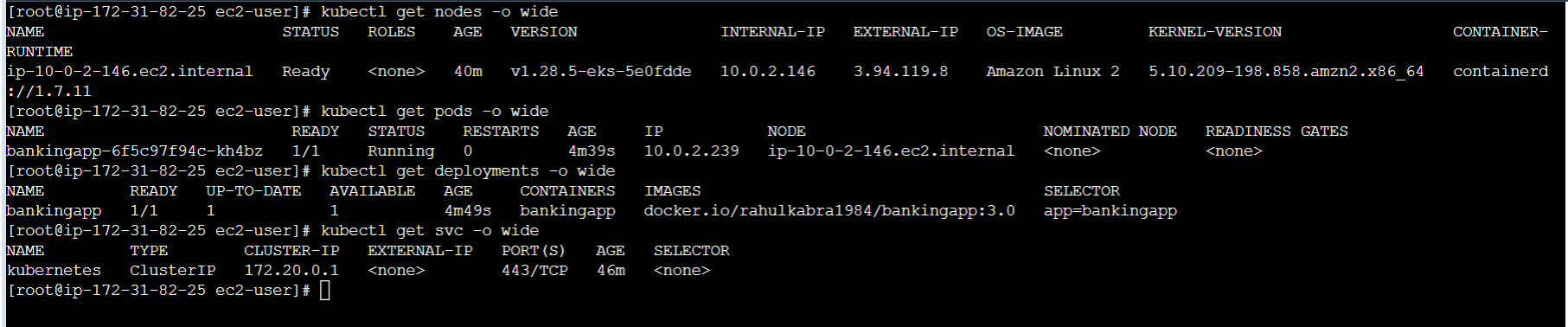
9.3 Verify in the terminal using below commands the pods created

**kubectl get nodes -o wide**

**kubectl get pods -o wide**

**kubectl get deployments -o wide**

**kubectl get svc -o wide**

****

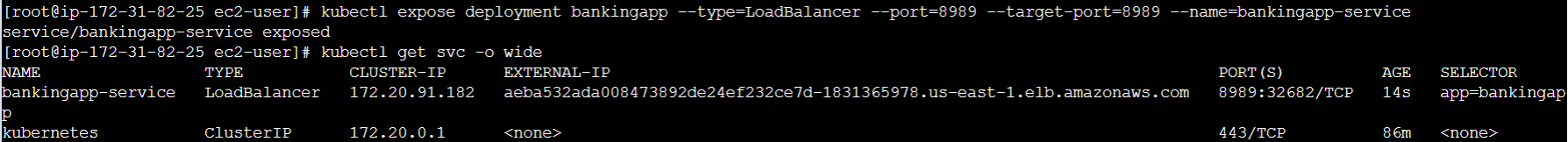
9.4 Now execute the below script in the shell script to create loadbalancer service and

verify it.

**kubectl expose deployment bankingapp --type=LoadBalancer --port=8989 --target-**

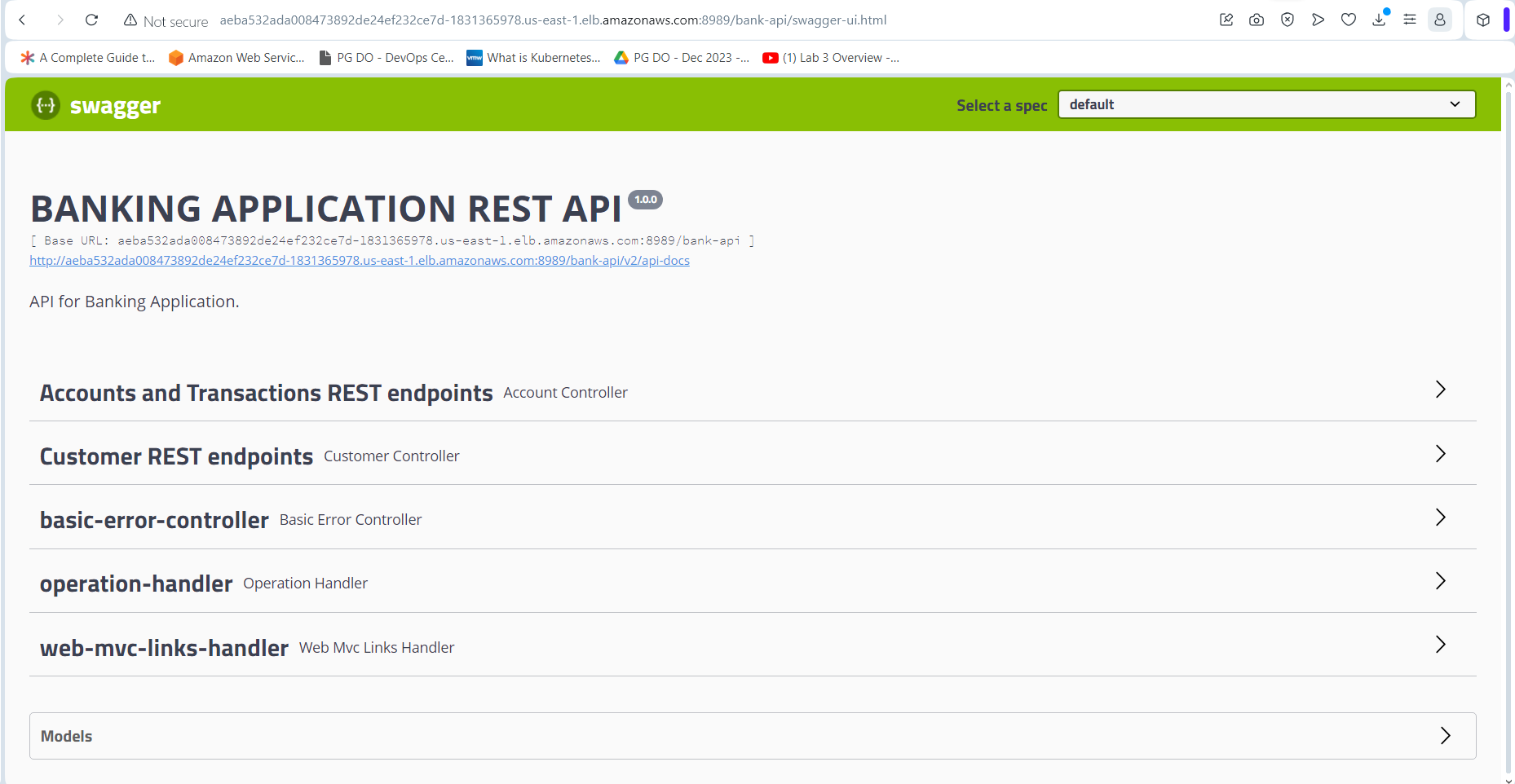
**port=8989 --name=bankingapp-service**

**kubectl get svc -o wide**

****

9.5 Now verify if the wesite is opening on the below address

http://aeba532ada008473892de24ef232ce7d-1831365978.us-east- 1.elb.amazonaws.com:8989/bank-api/swagger-ui.html



**Step 10 : Validate autoscaling is working**

Now go to the autoscaling group of cluster and in autoscaling group create a dynamic autoscaling rule

10.1 Create a new file stress.yaml and pu the following code in it

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: nginx-stress**

**spec:**

**containers:**

**- name: nginx**

**image: ubuntu/nginx**

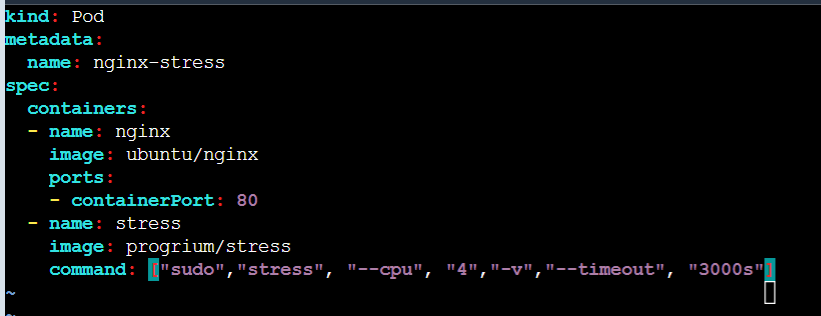
**ports:**

**- containerPort: 80**

**- name: stress**

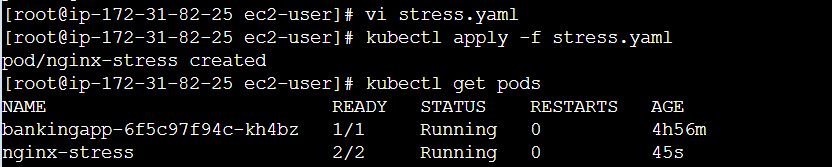
**image: progrium/stress**

**command: ["sudo", "stress", "--cpu", "4", "-v", "--timeout", "3000s"]**

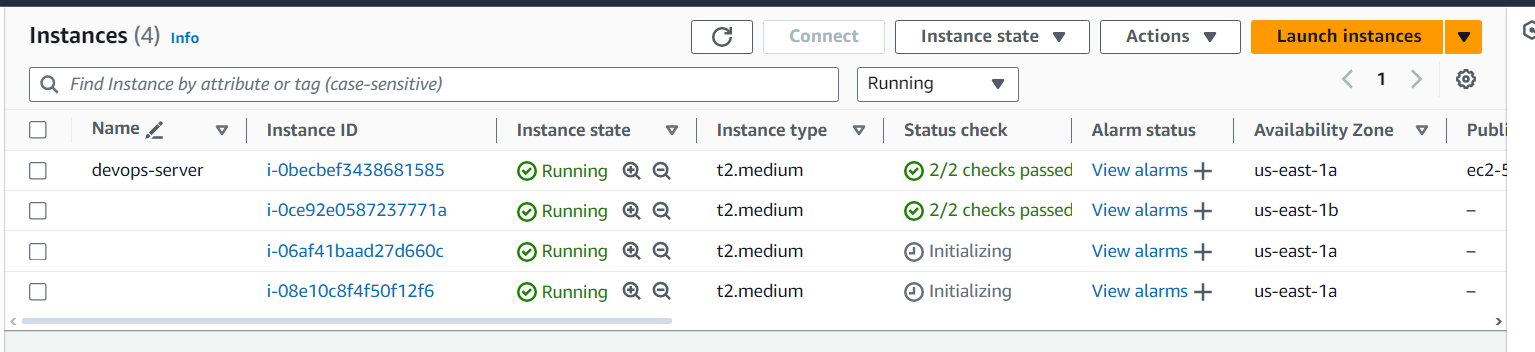
****

10.2 Execute below set of commands to install stress utility:

**kubectl apply -f stress.yaml**

****

10.3 Verify if one more instance gets created

****

