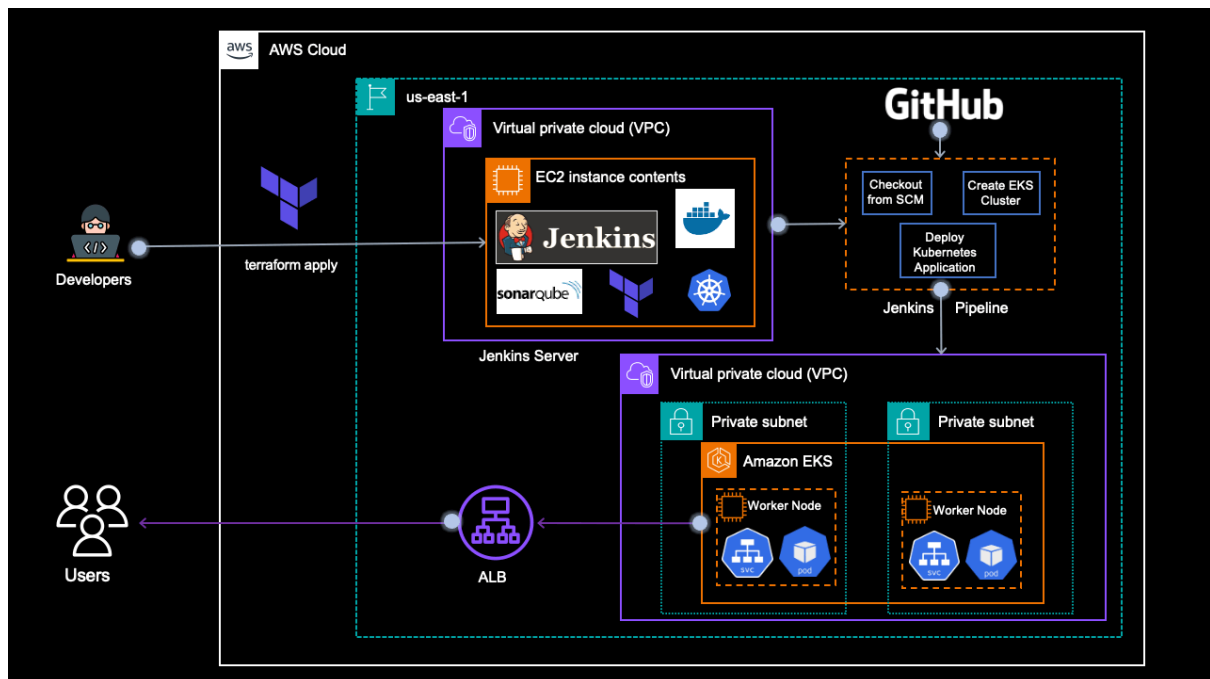


From Scratch to Production: Deploying EKS Clusters and Applications with CI/CD using Jenkins and Terraform



Streamlining EKS Deployment and CI/CD: A Step-by-Step Guide to Automating Application Delivery with Jenkins and Terraform

Welcome to this step-by-step guide on deploying an EKS cluster and application with complete CI/CD!

Are you looking to streamline your application delivery process and automate your infrastructure deployment? Look no further! In this project, I'll take you through the process of setting up an EKS cluster, deploying an application, and creating a CI/CD pipeline using Jenkins and Terraform.

We'll start with the basics and gradually dive deeper into the technical details, so you'll find this guide helpful whether you're a beginner or an experienced DevOps engineer. By the end of this article, you'll have a fully functional EKS cluster and a simple containerized application up and running, with a CI/CD pipeline that automates the entire process from code to production.

Let's get started and explore the world of EKS, CI/CD, and automation

What we'll build

We are going to build and deploy a lot of things. Here is the outline for our project:

I. Setting up Jenkins Server with Terraform

- Creating an EC2 instance with Terraform.
- Installing necessary tools: Java, Jenkins, AWS CLI, Terraform CLI, Docker, Sonar, Helm, Trivy, Kubectl.
- Configuring Jenkins server.

II. Creating EKS Cluster with Terraform

- Writing Terraform configuration files for EKS cluster creation in a private subnet.
- Deploying EKS cluster using Terraform.

III. Deploying NGinx Application with Kubernetes

- Writing Kubernetes manifest files (YAML) for the NGinx application.
- Deploying NGinx application to EKS cluster.

IV. Automating Deployment with Jenkins CI/CD

- Creating Jenkins pipeline for automating EKS cluster creation and Nginx application deployment.
- Integrating Terraform and Kubernetes with the Jenkins pipeline.
- Configuring continuous integration and deployment (CI/CD).

What we'll need

To embark on our CI/CD adventure, we'll need a trusty toolkit:

Terraform — To create configuration files for the EC2 instance which will be used as a Jenkins server and EKS Cluster in a VPC.

Shell Script — To install command line tools in the EC2 instance.

Jenkins file — To create a pipeline in the Jenkins Server.

Kubernetes Manifest files — To create a simple NGINX application in the EKS cluster.

Source Code

You can download the complete source code inside this repository.

Prerequisites

Before creating and working with the project, let's set up some dev tools first -

1. It's better to have an IDE to develop your project. I am using Visual Studio Code for the same. You can install it from the following link based on the operating system— <https://code.visualstudio.com/download>
2. Install the CLI tools — [AWS-CLI](#), and [Terraform-CLI](#).
3. Make sure you have an AWS Free Tier Account. And then create a user in IAM Console and finally create an Access Key ID and Secret Access Key in AWS Console for that user. You need to download these keys and then export those credentials in your terminal as follows —

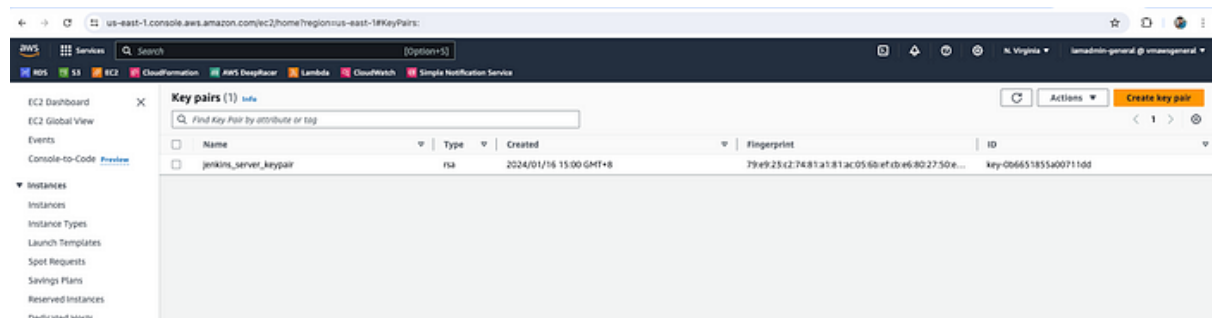
```
export AWS_ACCESS_KEY_ID=<Copy this from the credentials file downloaded>
```

```
export AWS_SECRET_ACCESS_KEY=<Copy this from the credentials file downloaded>
```

Stage 1: Configure and Build Jenkins Server

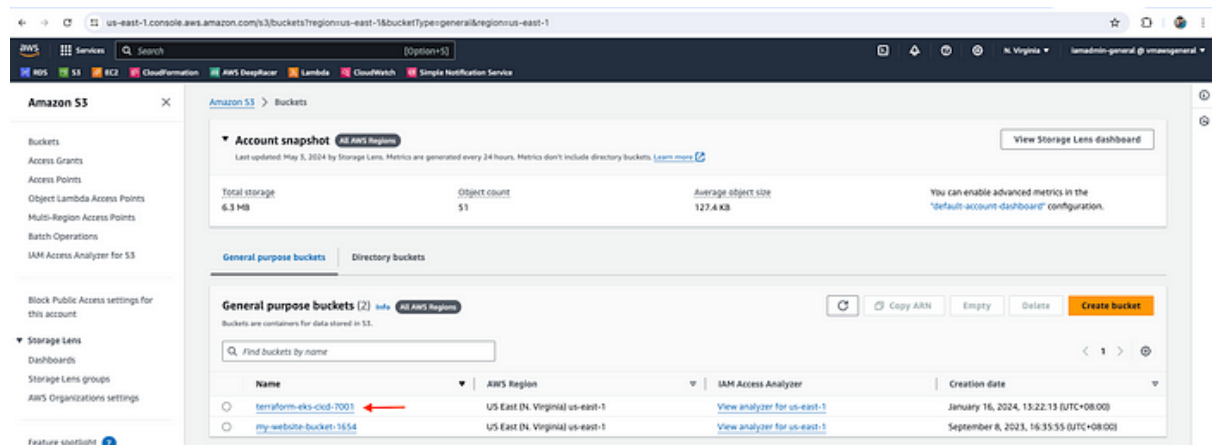
The first thing we have to do is to create a new key pair for login into the EC2 instance and create an S3 bucket for storing terraform state files. This is the only manual step we are doing.

So, in the AWS management console go to "EC2" and select "Key pairs" in the listed overview of your resources, and then select "Create key pair" at the top right corner. You need to download these key pairs so that you can use them later for logging into the EC2 instance.



Create Key pairs for the EC2 instance

Next, let's create a S3 bucket to store the terraform remote states. You can also create a S3 bucket via Terraform but in that case, you need to apply this configuration first as the S3 bucket must already exist before using it as a remote backend in Terraform. Hence, go to S3 and create bucket → terraform-eks-cicd-7001 (Use some random number at the end to make it unique).



Create an S3 bucket to store terraform remote state

Now, let's start writing terraform configuration for our EC2 instance which will be used as a Jenkins server. So, we will create the instance first and then we will install the necessary tools like jenkins etc via a build script.

Here are the Terraform configuration files -

backend.tf

```
terraform {  
  
  backend "s3" {  
  
    bucket = "terraform-eks-cicd-7001"  
  
    key   = "jenkins/terraform.tfstate"  
  
    region = "us-east-1"  
  
  }  
}
```

```
}
```

```
data.tf
```

```
data "aws_availability_zones" "azs" {}
```

```
# Get latest Amazon Linux AMI
```

```
data "aws_ami" "amazon-linux" {
```

```
    most_recent = true
```

```
    owners      = ["amazon"]
```

```
    filter {
```

```
        name = "name"
```

```
        values = ["amzn2-ami-*-x86_64-gp2"]
```

```
    }
```

```
    filter {
```

```
        name = "virtualization-type"
```

```
        values = ["hvm"]
```

```
    }
```

```
}
```

```
main.tf
```

```
# We'll be using publicly available modules for creating different services instead of resources
```

```
# https://registry.terraform.io/browse/modules?provider=aws
```

```
# Creating a VPC
```

```
module "vpc" {
```

```
    source = "terraform-aws-modules/vpc/aws"
```

```
    name = var.vpc_name
```

```
    cidr = var.vpc_cidr
```

```
    azs          = data.aws_availability_zones.azs.names
```

```
    public_subnets = var.public_subnets
```

```
    map_public_ip_on_launch = true
```

```
enable_dns_hostnames = true
```

```
tags = {  
  Name      = var.vpc_name  
  Terraform = "true"  
  Environment = "dev"  
}
```

```
public_subnet_tags = {  
  Name = "jenkins-subnet"  
}  
}
```

```
# SG
```

```
module "sg" {  
  source = "terraform-aws-modules/security-group/aws"
```

```
  name      = var.jenkins_security_group  
  description = "Security Group for Jenkins Server"  
  vpc_id    = module.vpc.vpc_id
```

```
  ingress_with_cidr_blocks = [  
    {  
      from_port = 8080  
      to_port   = 8080  
      protocol  = "tcp"  
      description = "JenkinsPort"  
      cidr_blocks = "0.0.0.0/0"  
    },  
  ]  
}
```

```
from_port = 443
to_port   = 443
protocol  = "tcp"
description = "HTTPS"
cidr_blocks = "0.0.0.0/0"
},
{
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    description = "HTTP"
    cidr_blocks = "0.0.0.0/0"
},
{
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    description = "SSH"
    cidr_blocks = "0.0.0.0/0"
},
{
    from_port = 9000
    to_port   = 9000
    protocol  = "tcp"
    description = "SonarQubePort"
    cidr_blocks = "0.0.0.0/0"
}
]
```

```
egress_with_cidr_blocks = [
{
```

```
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = "0.0.0.0/0"
  }
]
```

```
tags = {
  Name = "jenkins-sg"
}
}
```

EC2

```
module "ec2_instance" {
  source = "terraform-aws-modules/ec2-instance/aws"
```

```
  name = var.jenkins_ec2_instance
```

```
  instance_type      = var.instance_type
  ami                 = "ami-0e8a34246278c21e4"
  key_name            = "jenkins_server_keypair"
  monitoring          = true
  vpc_security_group_ids = [module.sg.security_group_id]
  subnet_id           = module.vpc.public_subnets[0]
  associate_public_ip_address = true
  user_data            = file("../scripts/install_build_tools.sh")
  availability_zone     = data.aws_availability_zones.azs.names[0]
```

```
tags = {
  Name      = "Jenkins-Server"
  Terraform = "true"
```

```
Environment = "dev"  
}  
}
```

```
install_build_tools.sh
```

```
#!/bin/bash
```

```
# Ref - https://www.jenkins.io/doc/book/installing/linux/
```

```
# Installing jenkins
```

```
sudo yum install wget -y
```

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
```

```
    https://pkg.jenkins.io/redhat/jenkins.repo
```

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

```
sudo yum upgrade -y
```

```
# Add required dependencies for the jenkins package
```

```
sudo yum install java-17-amazon-corretto-devel -y
```

```
sudo yum install jenkins -y
```

```
sudo systemctl daemon-reload
```

```
# Starting Jenkins
```

```
sudo systemctl enable jenkins
```

```
sudo systemctl start jenkins
```

```
sudo systemctl status jenkins
```

```
# Ref - https://www.atlassian.com/git/tutorials/install-git
```

```
# Installing git
```

```
sudo yum install -y git
```

```
git --version
```

```
# Installing Docker
```

```
# Ref - https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/
```

```
sudo yum update
```



```
sudo yum install docker -y
```

```
sudo usermod -a -G docker ec2-user
```

```
sudo usermod -aG docker jenkins
```

Add group membership for the default ec2-user so you can run all docker commands without using the sudo command:

```
id ec2-user
```

```
newgrp docker
```

```
sudo systemctl enable docker.service
```

```
sudo systemctl start docker.service
```

```
sudo systemctl status docker.service
```

```
sudo chmod 777 /var/run/docker.sock
```

Run Docker Container of Sonarqube

```
docker run -d --name sonar -p 9000:9000 sonarqube:its-community
```

Installing AWS CLI

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

```
sudo apt install unzip -y
```

```
unzip awscliv2.zip
```

```
sudo ./aws/install
```

Ref - <https://developer.hashicorp.com/terraform/cli/install/yum>

Installing terraform

```
sudo yum install -y yum-utils
```

```
sudo yum-config-manager --add-repo
```

```
https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
```

```
sudo yum -y install terraform
```

Ref - <https://pwittrock.github.io/docs/tasks/tools/install-kubect/>

Installing kubectl

```
sudo curl -LO https://storage.googleapis.com/kubernetes-  
release/release/v1.23.6/bin/linux/amd64/kubectl
```

```
sudo chmod +x ./kubectl
```

```
sudo mkdir -p $HOME/bin && sudo cp ./kubectl $HOME/bin/kubectl && export  
PATH=$PATH:$HOME/bin
```

Installing Trivy

Ref - <https://aquasecurity.github.io/trivy-repo/>

```
sudo tee /etc/yum.repos.d/trivy.repo << 'EOF'
```

```
[trivy]
```

```
name=Trivy repository
```

```
baseurl=https://aquasecurity.github.io/trivy-repo/rpm/releases/$basearch/
```

```
gpgcheck=1
```

```
enabled=1
```

```
gpgkey=https://aquasecurity.github.io/trivy-repo/rpm/public.key
```

```
EOF
```

```
sudo yum -y update
```

```
sudo yum -y install trivy
```

Installing Helm

Ref - <https://helm.sh/docs/intro/install/>

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
```

```
chmod 700 get_helm.sh
```

```
./get_helm.sh
```

Please note a few points before running terraform apply.

- Use the correct key pair name in the EC2 instance module (main.tf) and it must exist before creating the instance.
- Use the correct bucket name in the configuration for the remote backend S3 in thebackend.tf

- You need to use `user_data = file("../scripts/install_build_tools.sh")` in the EC2 module to specify the script to be executed after EC2 instance creation.

Let's run terraform apply and create this. Please make sure to run terraform init if you are doing this for the first time. Also, double-check your current working directory where you are running the terraform cli commands.

(ecsproject_py310) Project-7 \$ pwd 4:18PM

/Users/vishalmishra/Study/medium/AWSDevOpsProjects/Project-7

(ecsproject_py310) Project-7 \$ cd jenkins_server 4:18PM

(ecsproject_py310) jenkins_server \$ cd tf-aws-ec2 4:19PM

(ecsproject_py310) tf-aws-ec2 \$ pwd 4:19PM

/Users/vishalmishra/Study/medium/AWSDevOpsProjects/Project-7/jenkins_server/tf-aws-ec2

(ecsproject_py310) tf-aws-ec2 \$ export AWS_ACCESS_KEY_ID=xxxxxx 4:19PM

export AWS_SECRET_ACCESS_KEY=xxxxxx

(ecsproject_py310) tf-aws-ec2 \$ terraform apply -var-file=variables/dev.tfvars -auto-approve

The screenshot shows a VS Code editor with a Terraform configuration file open. The configuration defines an EC2 instance module. The terminal window at the bottom shows the command history from the previous steps, including directory navigation and the execution of the Terraform apply command.

```

89 module "ec2_instance" {
90   source = "terraform-aws-modules/ec2-instance/aws"
91
92   name = var.jenkins_ec2_instance
93
94   instance_type = var.instance_type
95   ami            = "ami-0eda34246278c21e4"
96   key_name       = "jenkins_server_keypair"
97   monitoring     = true
98   vpc_security_group_ids = [module.sg.security_group_id]
99   subnet_id      = module.vpc.public_subnets[0]
100  associate_public_ip_address = true
101  user_data        = file("../scripts/install_build_tools.sh")
102  availability_zone = data.aws_availability_zones.azs.names[0]
103
104  tags = {
105    Name        = "Jenkins-Server"
106    Terraform   = "true"
107    Environment = "dev"
108  }
109 }

```

Terminal output (4:18PM to 4:19PM):

```

(ecspjroct_py310) Project-7 $ pwd
/Users/vishalmishra/Study/medium/AWSDevOpsProjects/Project-7
(ecspjroct_py310) Project-7 $ cd jenkins_server
(ecspjroct_py310) jenkins_server $ cd tf-aws-ec2
(ecspjroct_py310) tf-aws-ec2 $ pwd
/Users/vishalmishra/Study/medium/AWSDevOpsProjects/Project-7/jenkins_server/tf-aws-ec2
(ecspjroct_py310) tf-aws-ec2 $ export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
(ecspjroct_py310) tf-aws-ec2 $ terraform apply -var-file=variables/dev.tfvars -auto-approve

```

terraform apply

The screenshot shows the output of the Terraform apply command. It details the creation of VPC resources and the EC2 instance, including the instance ID and IP address.

```

module.vpc.aws_subnet.public[0]: Creation complete after 13s [id=subnet-022ecd5a855f6e884]
module.vpc.aws_route_table_association.public[0]: Creating...
module.ec2_instance.aws_instance.this[0]: Creating...
module.vpc.aws_route_table_association.public[0]: Creation complete after 1s [id=rtbassoc-0ec6091c185bd26b8]
module.ec2_instance.aws_instance.this[0]: Still creating... [10s elapsed]
module.ec2_instance.aws_instance.this[0]: Still creating... [20s elapsed]
module.ec2_instance.aws_instance.this[0]: Creation complete after 28s [id=i-06a74965cc4e467ce]

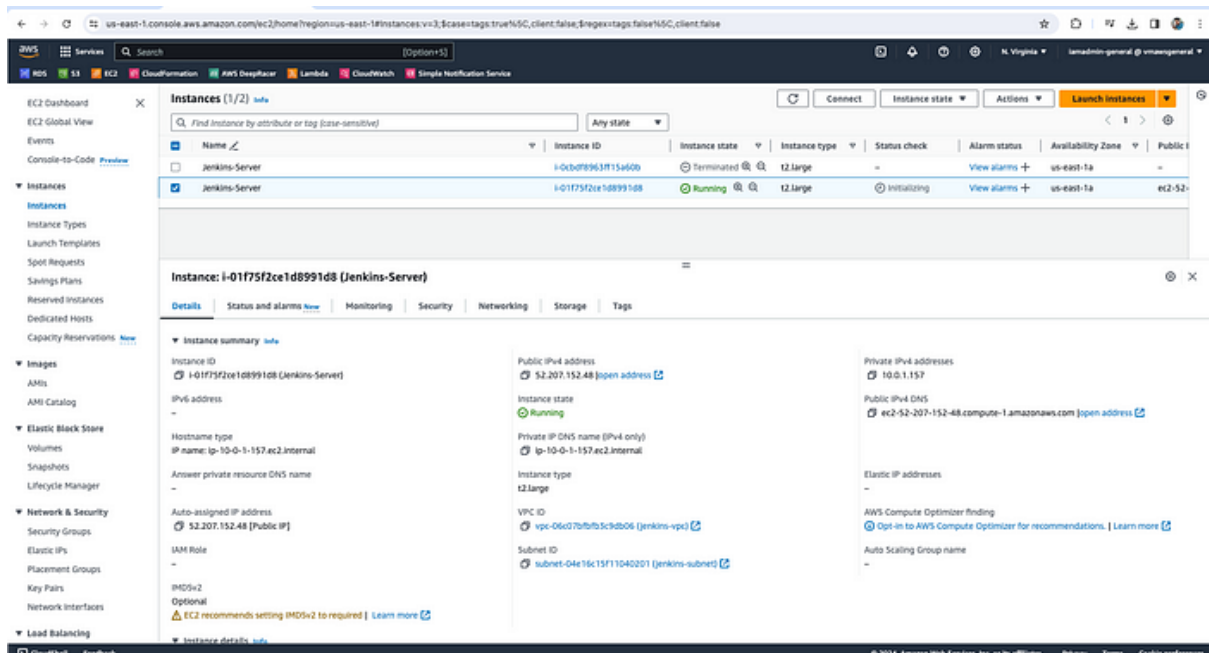
Apply complete! Resources: 17 added, 0 changed, 0 destroyed.

Outputs:
ec2_instance_ip = "184.72.83.48"
(ecspjroct_py310) tf-aws-ec2 $
(ecspjroct_py310) tf-aws-ec2 $

```

terraform apply

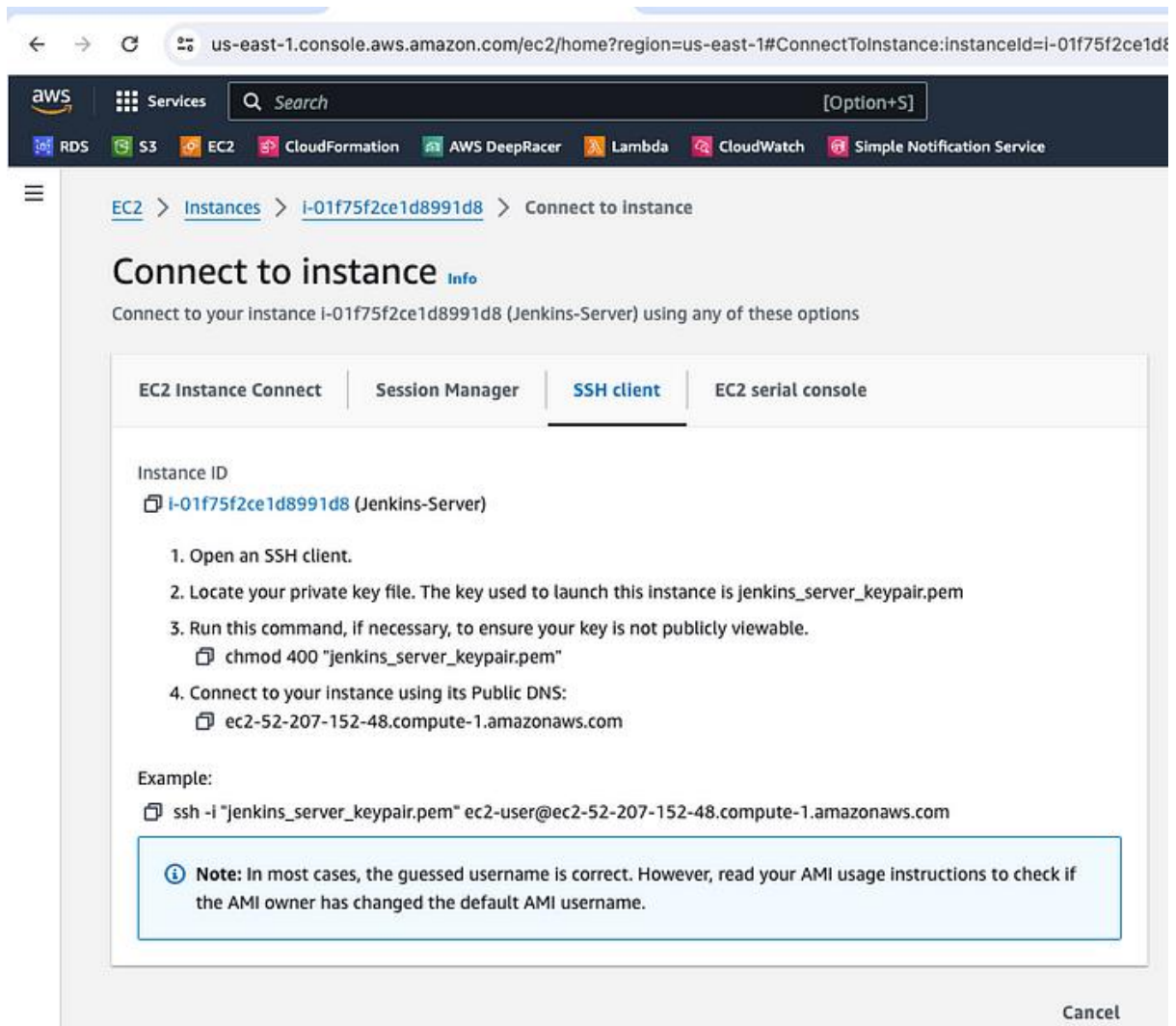
Give it some time before you go to the AWS EC2 console and check the instance status. Even though the instance is running, it may still be installing the tools.



Jenkins Build Server Created

Now, let's log in to the Jenkins server and verify if all the tools have been installed correctly or not.

So, let's select the EC2 instance and hop on to connect and copy the ssh command.




ssh commands to log in to EC2 instance

Next, go to your terminal, and paste the ssh commands. But before this make sure you have the keypair file downloaded in your workstation.

```
Last login: Sat Mar  9 13:48:27 on ttys008
~ $ pwd
/Users/vishalmishra
~ $ cd Downloads
Downloads $
Downloads $ ls -ltr jenkins_server_keypair.pem
-r-----@ 1 vishalmishra  staff  1674 Jan 16 15:00 jenkins_server_keypair.pem
Downloads $
```

```
Last login: Sat Mar 9 13:48:27 on ttys008
~ $ pwd
/Users/vishalmishra
~ $ cd Downloads
Downloads $ 
Downloads $ ls -ltr jenkins_server_keypair.pem
-r-----@ 1 vishalmishra staff 1674 Jan 16 15:00 jenkins_server_keypair.pem
Downloads $ ssh -i "jenkins_server_keypair.pem" ec2-user@ec2-52-207-152-48.compute-1.amazonaws.com
The authenticity of host 'ec2-52-207-152-48.compute-1.amazonaws.com (52.207.152.48)' can't be established.
ED25519 key fingerprint is SHA256:9FDcPNoRWJXEIqRYKG14YtcsmFsusbRwfBt4Tx46QPE.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-207-152-48.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
```



```
#_
~\_ #####      Amazon Linux 2
~~~\_#####\
~~~~\_###|       AL2 End of Life is 2025-06-30.
~~~~\_#/\
~~~~V-' '->
~~~~_/
~~~~_-/_-/
~~~~/_/_-/      Amazon Linux 2023, GA and supported until 2028-03-15.
~~~~/_m/'        https://aws.amazon.com/linux/amazon-linux-2023/

-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory
[ec2-user@ip-10-0-1-157 ~]$
```

successfully connected to the EC2 instance

We can now verify the versions of all the tools installed. Let's copy and paste the below commands.

```
jenkins --version
```

```
docker --version
```

```
docker ps
```

```
terraform --version
```

```
kubectl version
```

```
aws --version
```

```
trivy --version
```

```
helm version
```

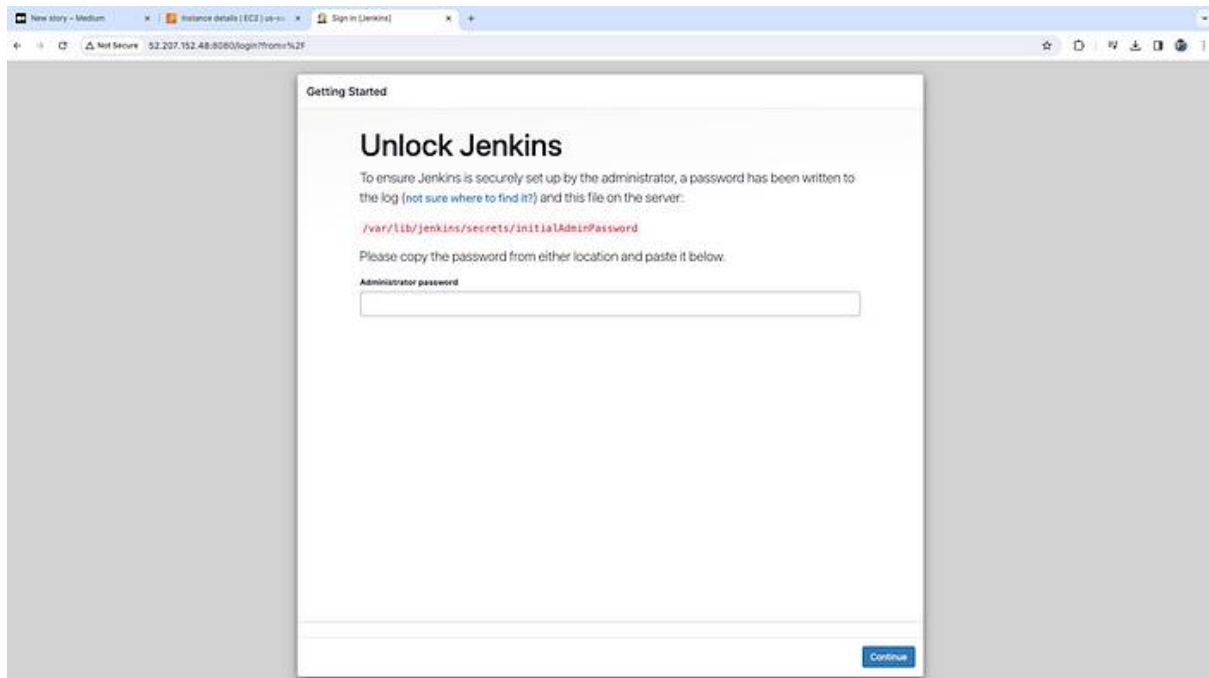
Here is the output.

```
[ec2-user@ip-10-0-1-157 ~]$ jenkins --version
2.448
[ec2-user@ip-10-0-1-157 ~]$ docker --version
Docker version 20.10.25, build b82b9f3
[ec2-user@ip-10-0-1-157 ~]$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
13c68707b4ce   sonarqube:lts-community             "/opt/sonarqube/dock... 22 minutes ago Up 22 minutes 0.0.0
.0:9000->9000/tcp, :::9000->9000/tcp   sonar
[ec2-user@ip-10-0-1-157 ~]$ terraform --version
Terraform v1.7.4
on linux_amd64
[ec2-user@ip-10-0-1-157 ~]$ kubectl version
Client Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.6", GitCommit:"ad338546da947756e8
a88aa6822e9c11e7eac22", GitTreeState:"clean", BuildDate:"2022-04-14T08:49:13Z", GoVersion:"go1.17.9", Co
mpiler:"gc", Platform:"linux/amd64"}
Error from server (Forbidden): <html><head><meta http-equiv='refresh' content='1;url=/login?from=%2Fvers
ion%3Ftimeout%3D32s'><script id='redirect' data-redirect-url='/login?from=%2Fversion%3Ftimeout%3D32s' s
rc='/static/66d1fa38/scripts/redirect.js'></script></head><body style='background-color:white; color:whi
te;'>
Authentication required
<!--
-->

</body></html>
[ec2-user@ip-10-0-1-157 ~]$ aws --version
aws-cli/2.15.27 Python/3.11.8 Linux/5.10.198-187.748.amzn2.x86_64 exe/x86_64.amzn.2 prompt/off
[ec2-user@ip-10-0-1-157 ~]$ trivy --version
Version: 0.49.1
[ec2-user@ip-10-0-1-157 ~]$ helm version
version.BuildInfo{Version:"v3.14.2", GitCommit:"c309b6f0ff63856811846ce18f3bdc93d2b4d54b", GitTreeState:
"clean", GoVersion:"go1.21.7"}
```

Tools version installed in the Jenkins Server (EC2 instance)

Let's configure the Jenkins in the EC2 instance. So, copy the EC2 instance's public IP address and paste it into the browser by adding the 8080 port which we have provided in the security group settings for Jenkins.

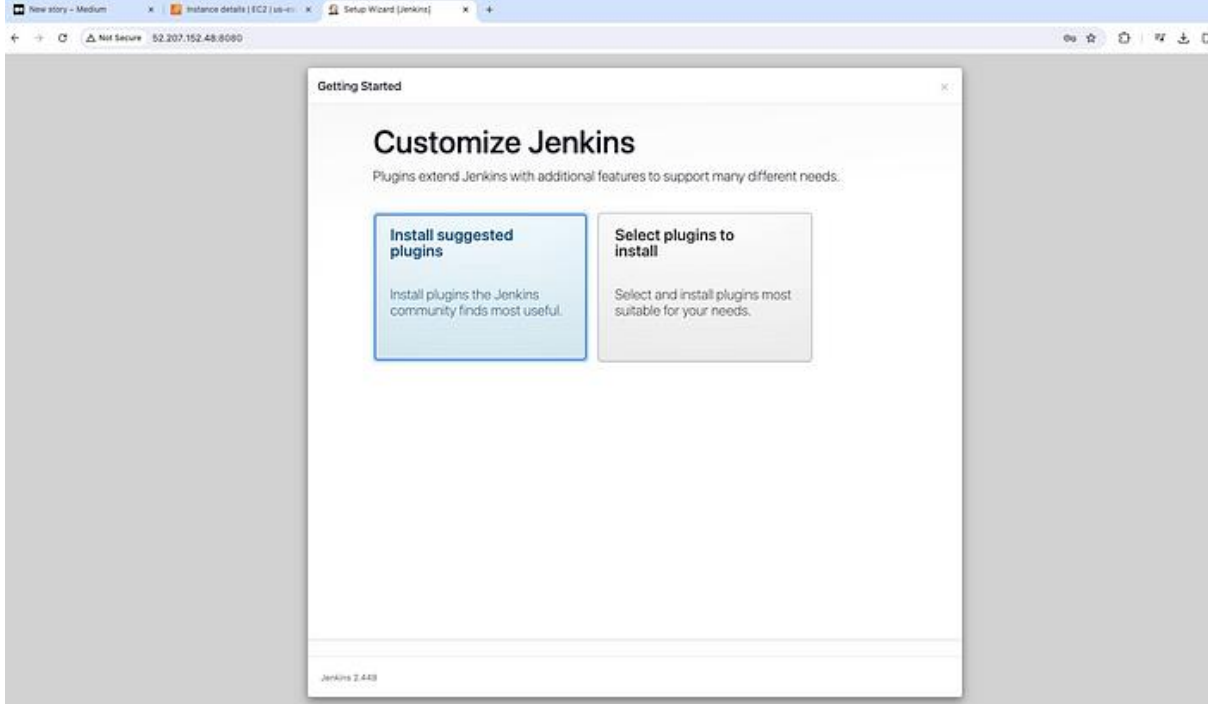


Now, copy the administrator password from the below path and paste and continue.

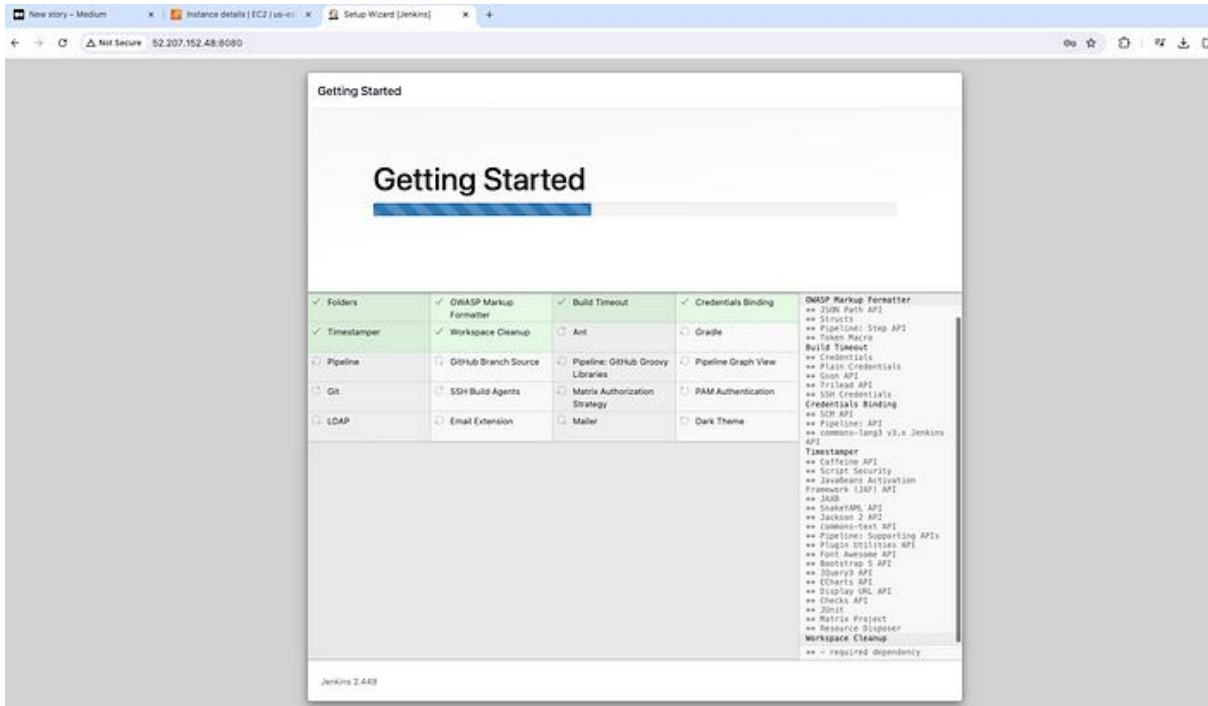
```
[ec2-user@ip-10-0-1-157 ~]$ cat /var/lib/jenkins/secrets/initialAdminPassword
cat: /var/lib/jenkins/secrets/initialAdminPassword: Permission denied
[ec2-user@ip-10-0-1-157 ~]$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
75f9498b7f0c40b49cfaf7d90c27d556
[ec2-user@ip-10-0-1-157 ~]$
```

Copy the Admin password

You will get the below screen. Click on Install Suggested Plugins.

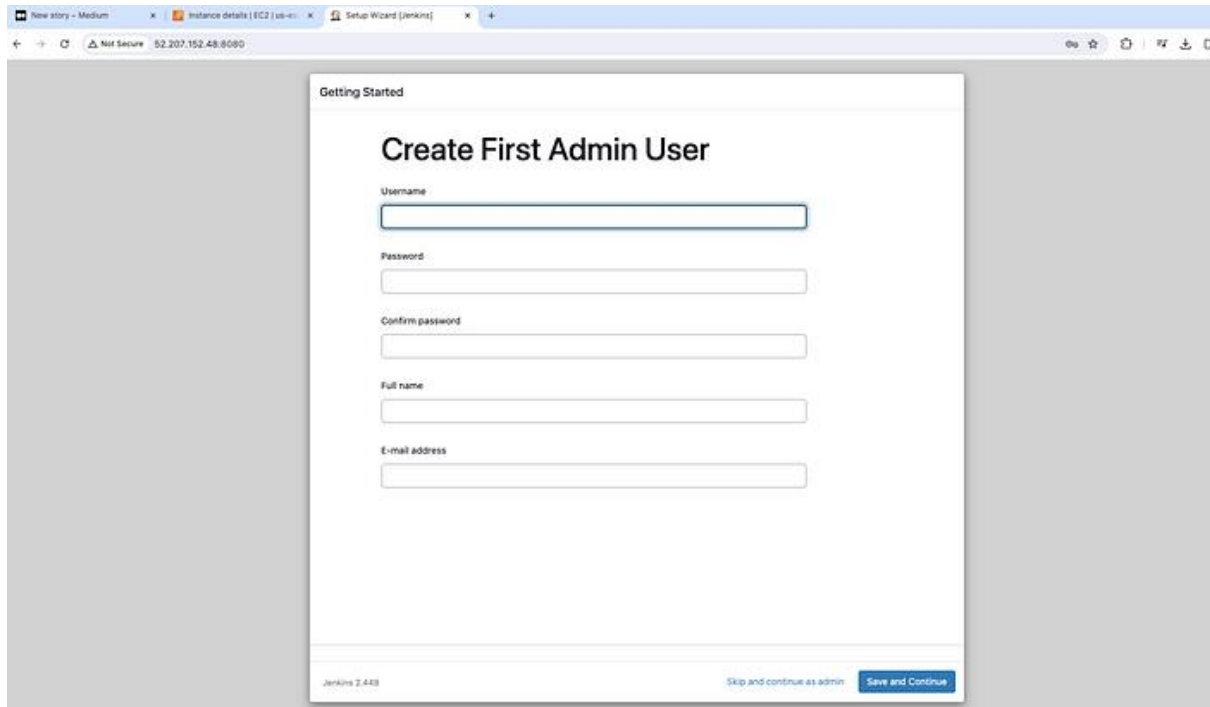


Install Suggested Plugin



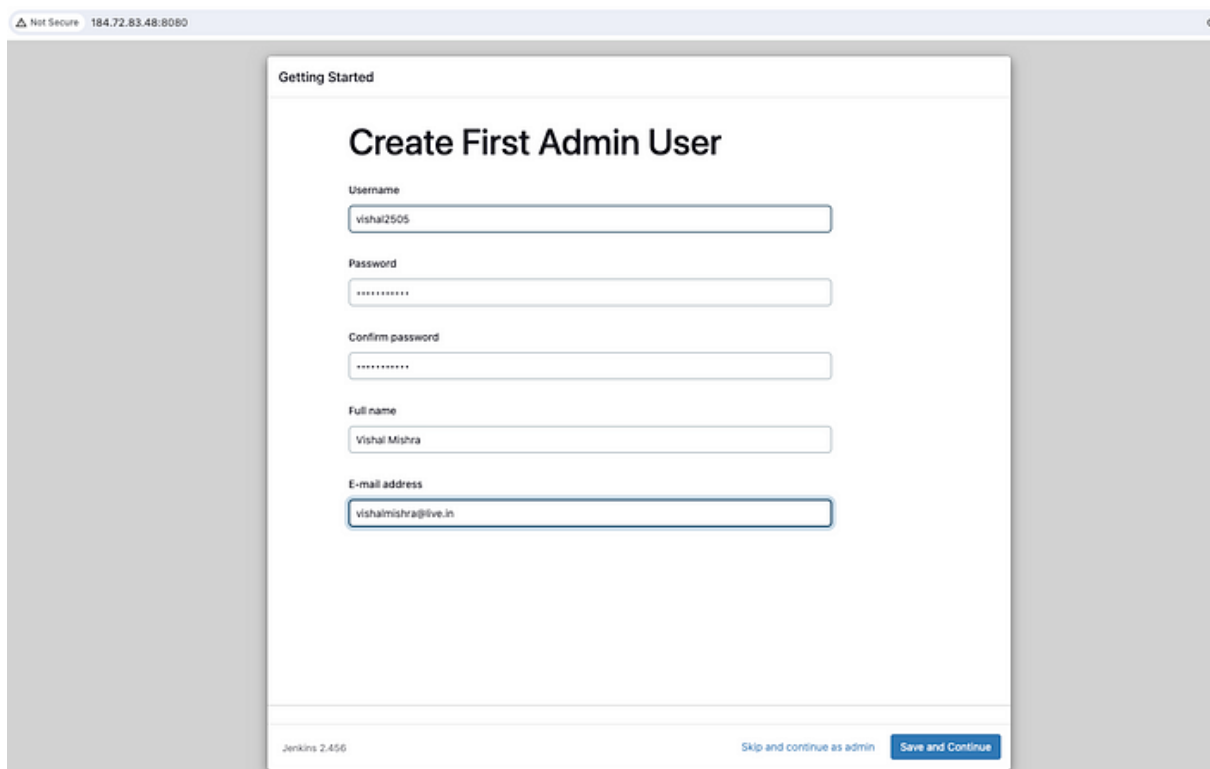
Plugin Installing

Once all the plugins are installed, you will be presented with the following screen. Here, you can continue as an admin (click on skip and continue as admin) or create a new user and password then click Save and Continue.



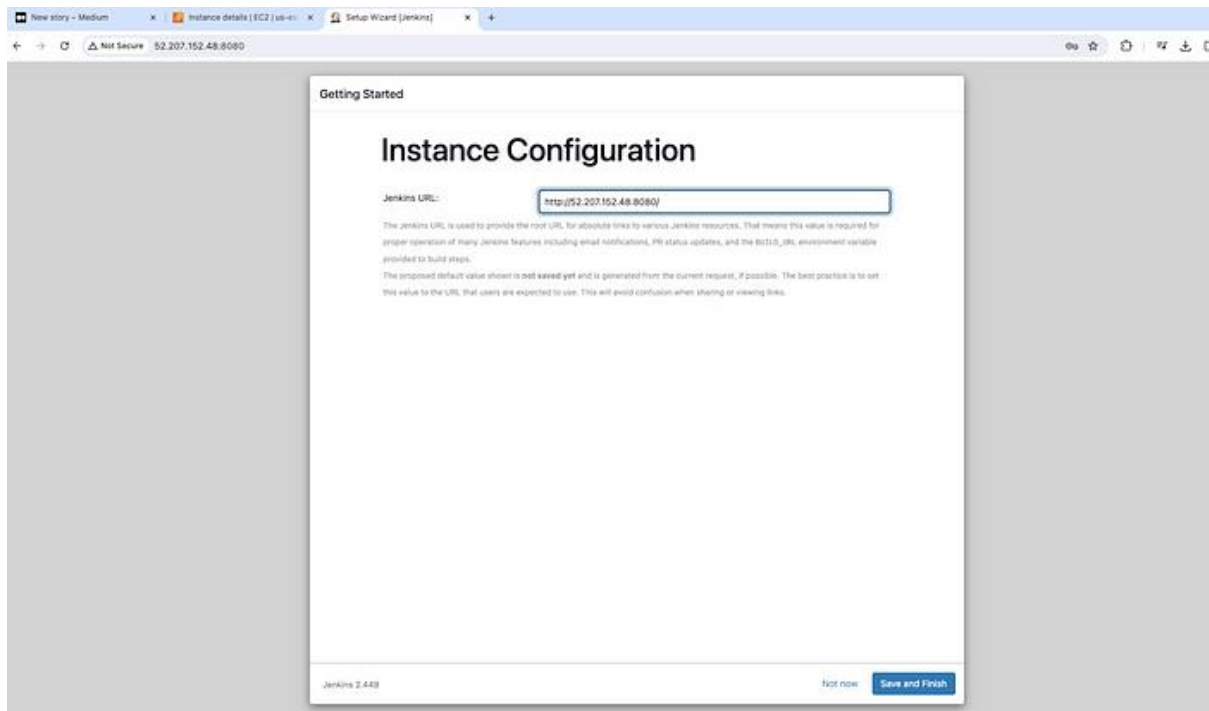
The screenshot shows the Jenkins 'Getting Started' screen with the 'Create First Admin User' form. The form has the following fields: Username, Password, Confirm password, Full name, and E-mail address. At the bottom, there are two buttons: 'Skip and continue as admin' and 'Save and Continue'. The Jenkins version is 2.443.

Create First Admin User

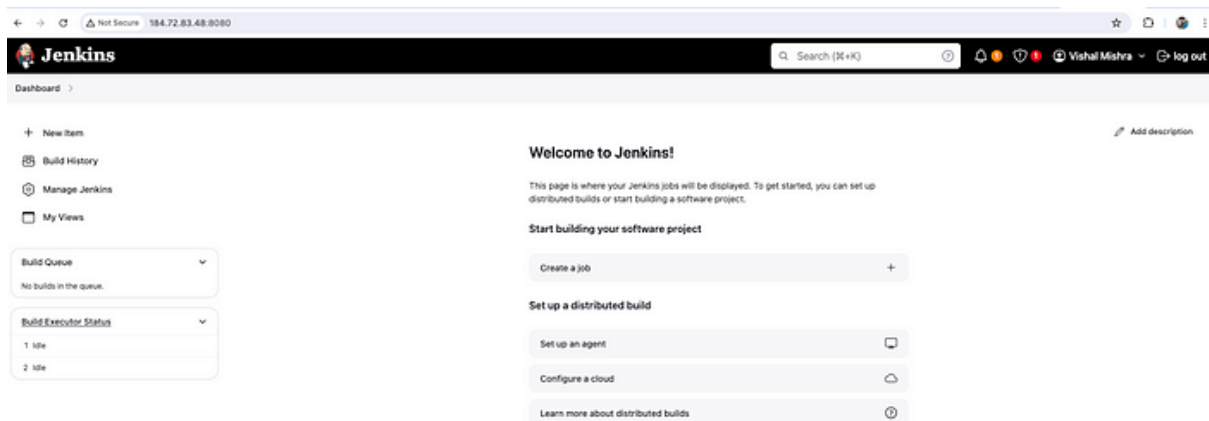


The screenshot shows the same Jenkins 'Create First Admin User' form, but with the following data entered: Username: vishal2505, Password: (masked with dots), Confirm password: (masked with dots), Full name: Vishal Mishra, and E-mail address: vishalmishra@live.in. The 'Save and Continue' button is highlighted. The Jenkins version is 2.456.

Click on Save and Finish and then on the next page Start using Jenkins.



Finally, you will get the below **Jenkins Dashboard**. At this point, we are ready with our Jenkins server. We'll configure the pipeline later.



Jenkins Dashboard

Stage 2: Create Terraform configuration files for creating the EKS Cluster

Task 1: Create Terraform configuration files

Moving on, let's start writing terraform configurations for the EKS cluster in a private subnet.

We'll use the same bucket but a different key/folder for the terraform remote state file.

backend.tf

```
terraform {  
  backend "s3" {
```

```

    bucket = "terraform-eks-cicd-7001"
    key    = "eks/terraform.tfstate"
    region = "us-east-1"
  }
}

vpc.tf

# We'll be using publicly available modules for creating different services instead of resources
# https://registry.terraform.io/browse/modules?provider=aws

# Creating a VPC
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name = var.vpc_name
  cidr = var.vpc_cidr

  azs          = data.aws_availability_zones.azs.names
  public_subnets = var.public_subnets
  private_subnets = var.private_subnets

  enable_dns_hostnames = true
  enable_nat_gateway   = true
  single_nat_gateway    = true

  tags = {
    "kubernetes.io/cluster/my-eks-cluster" = "shared"
    Terraform = "true"
    Environment = "dev"
  }
}

```

```
public_subnet_tags = {  
  "kubernetes.io/cluster/my-eks-cluster" = "shared"  
  "kubernetes.io/role/elb" = 1  
}
```

```
private_subnet_tags = {  
  "kubernetes.io/cluster/my-eks-cluster" = "shared"  
  "kubernetes.io/role/internal-elb" = 1  
}
```

```
}
```

eks.tf

Ref - <https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest>

```
module "eks" {  
  source = "terraform-aws-modules/eks/aws"  
  version = "~> 20.0"
```

```
  cluster_name = "my-eks-cluster"  
  cluster_version = "1.29"
```

```
  cluster_endpoint_public_access = true
```

```
  vpc_id = module.vpc.vpc_id  
  subnet_ids = module.vpc.private_subnets
```

```
  eks_managed_node_groups = {  
    nodes = {  
      min_size = 1  
      max_size = 3  
      desired_size = 2
```

```
    instance_types = ["t2.small"]
    capacity_type = "SPOT"
  }
}
```

```
tags = {
    Environment = "dev"
    Terraform   = "true"
}
}
```

Please make a note that we are using a private subnet for our EKS cluster as we don't want it to be publicly accessed.

dev.tfvars

```
Copyaws_region = "us-east-1"
aws_account_id = "12345678"
vpc_name       = "eks-vpc"
vpc_cidr       = "192.168.0.0/16"
public_subnets = ["192.168.1.0/24", "192.168.2.0/24", "192.168.3.0/24"]
private_subnets = ["192.168.4.0/24", "192.168.5.0/24", "192.168.6.0/24"]
instance_type   = "t2.small"
```

Task 2: Validate the terraform configuration files

Although we are going to create the AWS EKS infrastructure via the Jenkins pipeline, we first need to validate the configuration files that we have created in the previous step.

So, let's move the tf-aws-eks directory, initialize our working directory, and then run terraform plan and validate.

```
terraform init
```

```
PROBLEMS OUTPUT TERMINAL PORTS CODE REFERENCE LOG GITLENS DEBUG CONSOLE
(ecspjroject_py310) tf-aws-eks $ pwd
/Users/vishalaishra/Study/medium/AWSDevOpsProjects/Project-7/tf-aws-eks
(ecspjroject_py310) tf-aws-eks $ terraform init

3:36PM
3:36PM

Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/eks/aws 20.6.0 for eks...
- eks in .terraform/modules/eks
- eks.eks_managed_node_group in .terraform/modules/eks/modules/eks-managed-node-group
- eks.eks_managed_node_group.user_data in .terraform/modules/eks/modules/_user_data
- eks.fargate_profile in .terraform/modules/eks/modules/fargate-profile
Downloading registry.terraform.io/terraform-aws-modules/kms/aws 2.1.0 for eks.kms...
- eks.kms in .terraform/modules/eks.kms
- eks.self_managed_node_group in .terraform/modules/eks/modules/self-managed-node-group
- eks.self_managed_node_group.user_data in .terraform/modules/eks/modules/_user_data
Downloading registry.terraform.io/terraform-aws-modules/vpc/aws 5.5.3 for vpc...
- vpc in .terraform/modules/vpc

Initializing provider plugins...
- Finding hashicorp/cloudinit versions matching ">= 2.0.0"...
- Finding hashicorp/aws versions matching ">= 4.33.0, >= 5.20.0, 5.25.0, >= 5.40.0"...
- Finding hashicorp/tls versions matching ">= 3.0.0"...
- Finding hashicorp/time versions matching ">= 0.9.0"...
```

terraform init

Copyterraform validate

```
(ecspjroject_py310) tf-aws-eks $ terraform validate
Success! The configuration is valid.

3:41PM
```

terraform validate

Copyterraform plan

```
PROBLEMS OUTPUT TERMINAL PORTS CODE REFERENCE LOG GITLENS DEBUG CONSOLE
+ description      = "my-eks-cluster cluster encryption key"
+ enable_key_rotation = true
+ id               = (known after apply)
+ is_enabled       = true
+ key_id           = (known after apply)
+ key_usage        = "ENCRYPT_DECRYPT"
+ multi_region     = false
+ policy           = (known after apply)
+ tags             = {
+   "Environment" = "dev"
+   "Terraform"   = "true"
+   "terraform-aws-modules" = "eks"
+ }
+ tags_all         = {
+   "Environment" = "dev"
+   "Terraform"   = "true"
+   "terraform-aws-modules" = "eks"
+ }
}

Plan: 5S to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
(ecspjroject_py310) tf-aws-eks $

3:43PM
AWS X CodeWhisperer Ln 12, Col 2 Spaces: 2 UTF-8 LF () Terraform
```

terraform plan

Configuration files are all validated and terraform plan is running fine which means we are ready to run the terraform apply in the Jenkins pipeline.

Stage 3: Configure Jenkins pipeline

Let's proceed to the Jenkins URL again and start configuring the pipeline.

Click on "Create a Job", type "eks-cicd-pipeline" and select pipeline then OK.

New Item

Enter an item name

eks-cicd-pipeline

Select an item type

- Freestyle project**
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.
- Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.

OK

Create a Job -> Pipeline

On the next screen, provide "**description**", move to the bottom, and click on "**Save**".

Jenkins Dashboard

Build Queue: No builds in the queue.

Build Executor Status: 1 Idle, 2 Idle.

S	W	Name	Last Success	Last Failure	Last Duration
🕒	🔔	eks-cicd-pipeline	N/A	N/A	N/A

Pipeline created

Since we are going to run terraform commands in the pipeline, which will talk to our AWS environment, we need to provide/store AccessKey and SecretAccessKey somewhere in the vault so that the pipeline can use that.

Jenkins provides a facility to store secret credentials in the vault.

So, head on to the **Dashboard -> Manage Jenkins -> Credentials -> System -> Global credentials (unrestricted)**

Not Secure 184.72.83.48:8080/manage/credentials/store/system/domain/_newCredentials

Jenkins Search (H+K) Vishal Mishra

Manage Jenkins > Credentials > System > Global credentials (unrestricted)

New credentials

Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: [REDACTED]

ID: AWS_ACCESS_KEY_ID

Description:

Create

Create Secret text for AWS_ACCESS_KEY_ID

Not Secure 184.72.83.48:8080/manage/credentials/store/system/domain/_newCredentials

Jenkins Search (H+K) Vishal Mishra

Manage Jenkins > Credentials > System > Global credentials (unrestricted)

New credentials

Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: [REDACTED]

ID: AWS_SECRET_ACCESS_KEY

Description:

Create

Create Secret text for AWS_SECRET_ACCESS_KEY

Not Secure 184.72.83.48:8080/manage/credentials/store/system/domain/_

Jenkins Search (H+K) Vishal Mishra

Manage Jenkins > Credentials > System > Global credentials (unrestricted)

Global credentials (unrestricted) + Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
AWS_ACCESS_KEY_ID	AWS_ACCESS_KEY_ID	Secret text	
AWS_SECRET_ACCESS_KEY	AWS_SECRET_ACCESS_KEY	Secret text	

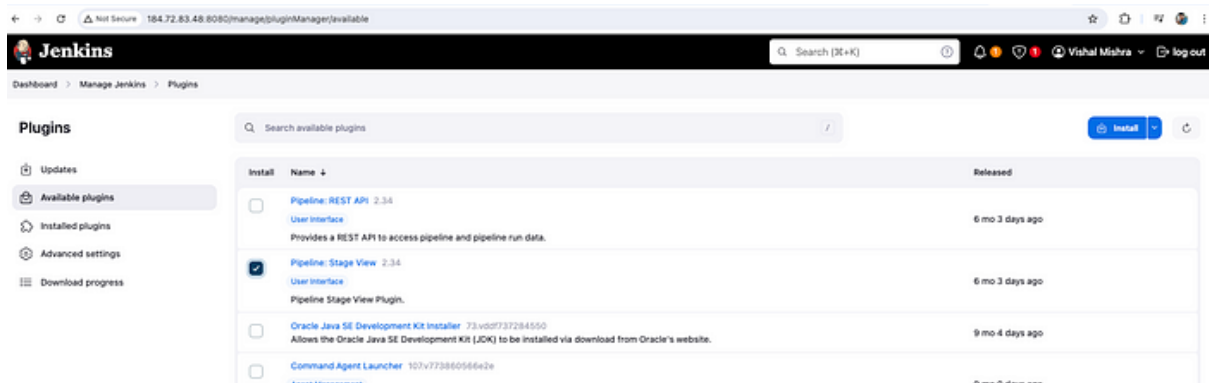
Icons: S M L

Access Keys created

You need to install one plugin to see the stage view in the pipeline.

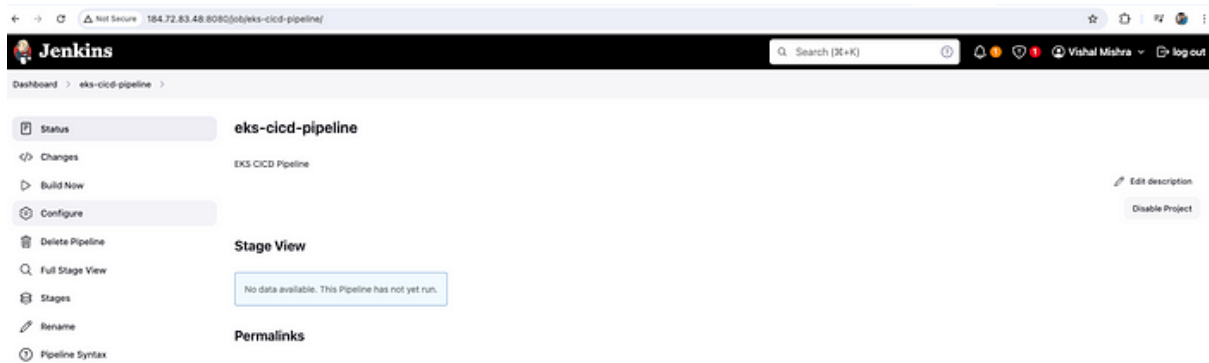
Go to **Dashboard -> Manage Jenkins -> Plugins -> Available plugins**

and select **Pipeline: Stage View** and click on **Install**.



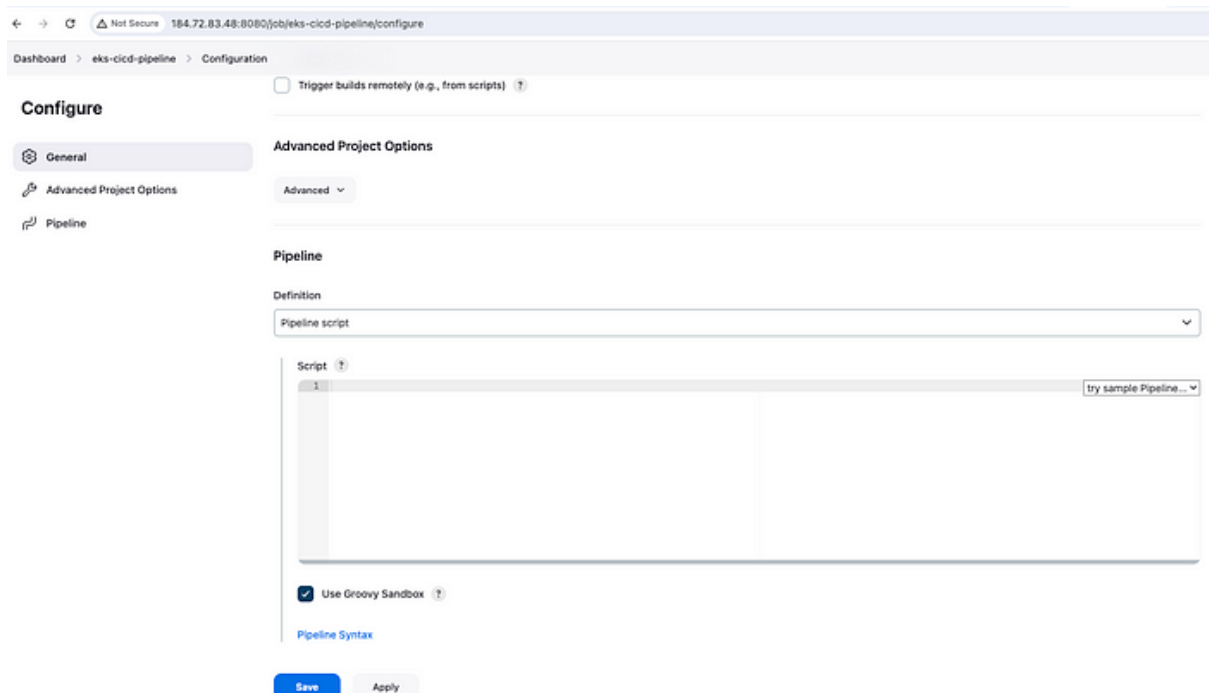
Install Plugin — Pipeline: Stage View

Finally, let's start configuring our pipeline. Go to your Dashboard and click on **Configure** →



Configure Pipeline

Now move to the bottom and start typing pipeline script using stages and tasks. You can also take help from Pipeline Syntax —



However, I have included the pipeline code in Jenkinsfile as below. Let's observe a few things here —

- We need to provide AWS credential variables that we added already in Jenkins.
- We need to provide Github location for the code with the current branch. Since this repository is public, we don't have to specify the GitHub token or credentials to access git.

Jenkinsfile

```
pipeline{
  agent any
  environment {
    AWS_ACCESS_KEY_ID = credentials('AWS_ACCESS_KEY_ID')
    AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCESS_KEY')
    AWS_DEFAULT_REGION = "us-east-1"
  }
  stages {
    stage('Checkout SCM') {
      steps {
        script {
          checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url:
'https://github.com/vishal2505/terraform-eks-cicd.git']])
        }
      }
    }
    stage('Initializing Terraform'){
      steps {
        script {
          dir('tf-aws-eks'){
            sh 'terraform init'
          }
        }
      }
    }
    stage('Validating Terraform'){
      steps {
```

```

    script {
        dir('tf-aws-eks'){
            sh 'terraform validate'
        }
    }
}

stage('Terraform Plan'){
    steps {
        script {
            dir('tf-aws-eks'){
                sh 'terraform plan -var-file=variables/dev.tfvars'
            }
        }
    }
}
}

```

Copy and Save the code and click on Build Now If everything is correct, you will see something similar to below.

The screenshot shows the Jenkins web interface for a pipeline named 'eks-cicd-pipeline'. The left sidebar contains navigation options: Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Stages, Rename, and Pipeline Syntax. The main area displays the 'Stage View' for the pipeline, showing a table of stages and their durations. Below the table, there is a 'Permalinks' section. A 'Build History' panel is visible in the bottom left corner, showing a list of builds with filters and a 'Build Now' button.

Stage	Checkout SCM	Initializing Terraform	Validating Terraform	Terraform Plan
Average stage times:	3s	9s	5s	6s
(Average full run time: ~29s)				
May 04 19:51	3s	9s	5s	6s

Build Now

Check the logs by clicking on #1 and then console output. We can see that the pipeline is successful and terraform plan got executed by showing 56 resources to add.

A screenshot of a web browser displaying the Jenkins console output for a pipeline named 'eks-cicd-pipeline' at step #1. The browser's address bar shows '184.72.83.48:8080/job/eks-cicd-pipeline/console'. The console output shows Terraform plan details for an EKS cluster. It lists tags for 'Environment' (dev), 'Terraform' (true), and 'terraform-aws-modules' (eks). It also shows a plan for a module 'eks.module.eks_managed_node_group["nodes"]' and a null resource 'validate_cluster_service_cidr'. The plan indicates 56 resources to be added. At the bottom, a note states: 'Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.' The pipeline ends with 'Finished: SUCCESS'.

Console Output

Now, add one more steps in the Jenkinsfile for terraform apply and then click on Save and Build Now.

Jenkinsfile

```
stage('Terraform Apply'){
    steps {
        script {
            dir('tf-aws-eks'){
                sh 'terraform apply -var-file=variables/dev.tfvars'
            }
        }
    }
}
```

This build will fail as we need to provide a flag-auto-approve otherwise it will prompt for "yes" which you can not confirm in the pipeline.

Build Failed

However, if you still want someone to approve and go ahead with the pipeline, you can make use of input() in the pipeline script.

There is one more change we are going to do which is to add a parameter action — >apply/destroy in the pipeline since we need to destroy the cluster when not in use.

Here is the updated code for the pipeline.

Jenkinsfile

```
pipeline{
    agent any
    environment {
        AWS_ACCESS_KEY_ID = credentials('AWS_ACCESS_KEY_ID')
        AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCESS_KEY')
        AWS_DEFAULT_REGION = "us-east-1"
    }
    stages {
        stage('Checkout SCM') {
            steps {
                script {
                    checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url:
'https://github.com/vishal2505/terraform-eks-cicd.git']])
                }
            }
        }
        stage('Initializing Terraform'){
            steps {
                script {
                    dir('tf-aws-eks'){
                        sh 'terraform init'
                    }
                }
            }
        }
        stage('Validating Terraform'){
            steps {
                script {
```

```

        dir('tf-aws-eks'){
            sh 'terraform validate'
        }
    }
}

stage('Terraform Plan'){
    steps {
        script {
            dir('tf-aws-eks'){
                sh 'terraform plan -var-file=variables/dev.tfvars'
            }
            input(message: "Are you sure to proceed?", ok: "Proceed")
        }
    }
}

stage('Creating/Destroying EKS Cluster'){
    steps {
        script {
            dir('tf-aws-eks'){
                sh 'terraform $action -var-file=variables/dev.tfvars -auto-approve'
            }
        }
    }
}
}

```

! Not Secure 184.72.83.48:8080/job/eks-cicd-pipeline/configure

> eks-cicd-pipeline > Configuration

Configure

General

Selected Project Options

ne

- ☐ Github project
- ☐ Pipeline speed/durability override ?
- ☐ Preserve stashes from completed builds ?
- ☒ This project is parameterized ?

Choice Parameter ?

Name ?

action

Choices ?

apply
destroy

Description ?

Plain text [Preview](#)

Add Parameter ▾

- ☐ Throttle builds ?

Build Triggers

[Save](#) [Apply](#)

Adding parameter

Now, let's run the pipeline again by clicking on Build with Parameters —



Dashboard > eks-cicd-pipeline >

- Status
- Changes
- Build with Parameters
- Configure
- Delete Pipeline
- Full Stage View
- Stages
- Rename
- Pipeline Syntax

Pipeline eks-cicd-pipeline

This build requires parameters:

action

Build

Cancel

Build History trend ▾

Filter... /

#2

| 04-May-2024, 11:57 am

#1

| 04-May-2024, 11:51 am

[Atom feed for all](#) [Atom feed for failures](#)

Build with Parameters

← → ↻ Not Secure 184.72.83.48:8080/job/eks-cicd-pipeline/

Jenkins

Dashboard > eks-cicd-pipeline >

Status

Changes

Build with Parameters

Configure

Delete Pipeline

Full Stage View

Stages

Rename

Pipeline Syntax

eks-cicd-pipeline

EKS CICD Pipeline

Stage View

	Checkout SCM	Initializing Terraform	Validating Terraform	Terraform Plan	Creating/Destroying EKS Cluster
Average stage times:	314ms	4s	5s	7s	5s
#3 May 04 20:32 No Changes	314ms	4s	5s	7s (skipped for 37s)	5s

Permalinks

- Last build (#3), 0.3 sec ago
- Last stable build (#1), 40 min ago
- Last successful build (#1), 40 min ago
- Last failed build (#2), 34 min ago
- Last unsuccessful build (#2), 34 min ago
- Last completed build (#2), 34 min ago

Build History trend

Filter...

- #3 04-May-2024, 12:32 pm
- #2 04-May-2024, 11:57 am
- #1 04-May-2024, 11:51 am

Atom feed for all Atom feed for failures

Pipeline running

We need to wait at least 15 mins for the pipeline to be finished.

← → ↻ Not Secure 184.72.83.48:8080/job/eks-cicd-pipeline/

Jenkins

Dashboard > eks-cicd-pipeline >

Status

Changes

Build with Parameters

Configure

Delete Pipeline

Full Stage View

Stages

Rename

Pipeline Syntax

eks-cicd-pipeline

EKS CICD Pipeline

Stage View

	Checkout SCM	Initializing Terraform	Validating Terraform	Terraform Plan	Creating/Destroying EKS Cluster
Average stage times:	314ms	4s	5s	7s	14min 1s
#3 May 04 20:32 No Changes	314ms	4s	5s	7s (skipped for 37s)	14min 1s

Permalinks

- Last build (#3), 26 min ago
- Last stable build (#3), 26 min ago
- Last successful build (#3), 26 min ago
- Last failed build (#2), 1 hr 1 min ago
- Last unsuccessful build (#2), 1 hr 1 min ago
- Last completed build (#3), 26 min ago

Build History trend

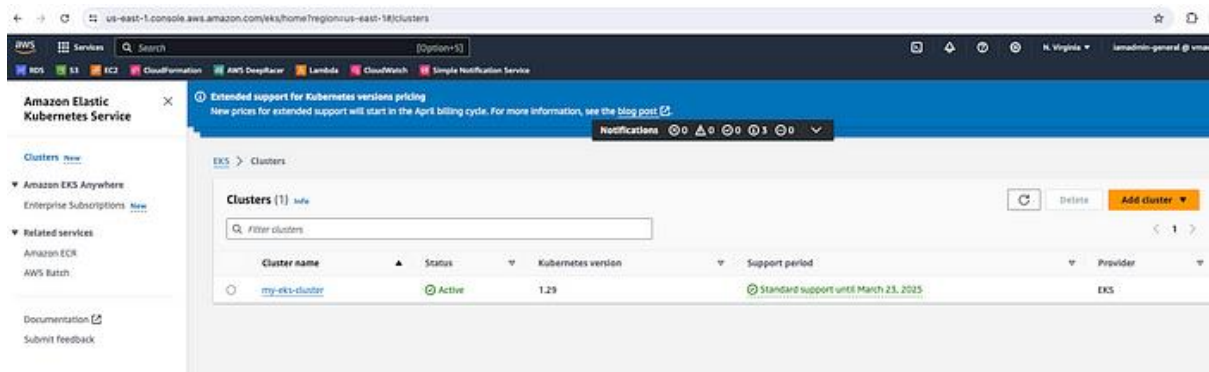
Filter...

- #3 04-May-2024, 12:32 pm
- #2 04-May-2024, 11:57 am
- #1 04-May-2024, 11:51 am

Atom feed for all Atom feed for failures

Pipeline Success

Let's verify the EKS cluster in the AWS Console.



EKS Cluster — Created

Stage 4: Adding Kubernetes manifest files for the Nginx Application

We have come to the last stage where we are going to **deploy** a simple Kubernetes application to the cluster. Ideally, in a production scenario, there will be different pipelines for the infrastructure (EKS Cluster) and the application, and again if the application is 2-tier or 3-tier, there will be separate pipelines for each tier to maintain microservices architecture.

We are going to create a simple Nginx application that we are going to access via the LoadBalancer endpoint. Hence, let's create 2 manifest files — deployment.yaml and service.yaml

deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx

spec:

selector:

matchLabels:

app: nginx

replicas: 1

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

```
image: nginx

ports:

  - containerPort: 80
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Keep these files in another directory for example manifest . And also add another stage in the Jenkins pipeline. This stage is gonna apply manifest files using kubectl utility that we installed in the EC2 instance (Jenkins Server) initially.

Jenkinsfile

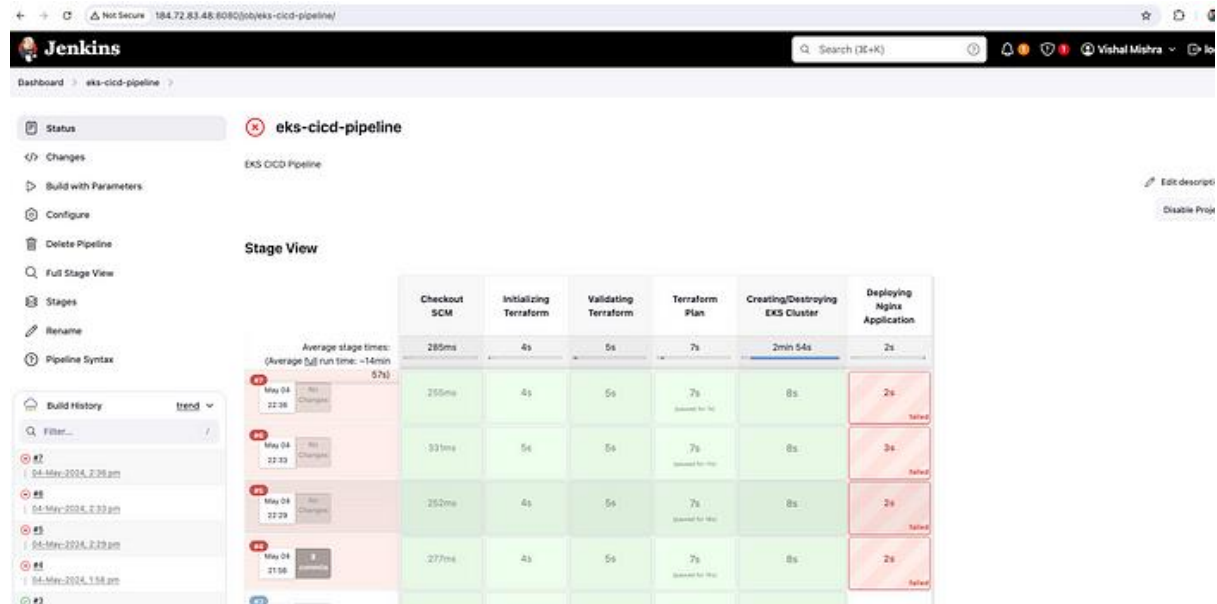
```
stage('Deploying Nginx Application') {
  steps{
    script{
      dir('manifest') {
        sh 'aws eks update-kubeconfig --name my-eks-cluster'
        sh 'kubectl create namespace eks-nginx-app'
        sh 'kubectl apply -f deployment.yaml'
        sh 'kubectl apply -f service.yaml'
      }
    }
  }
}
```

```

    }
  }
}

```

However, once you run the pipeline, it is going to fail again -



Pipeline failed

Let's check the log of the failed pipeline.

```

[0m [In [32]
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
[0m
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { [Deploying Nginx Application]
[Pipeline] script
[Pipeline] {
[Pipeline] dir
Running in /var/lib/jenkins/workspace/eks-cicd-pipeline/tf-aws-eks/manifest
[Pipeline] {
[Pipeline] sh
+ aws eks update-kubeconfig --name my-eks-cluster
Added new context arn:aws:eks:us-east-1:593382476592:cluster/my-eks-cluster to /var/lib/jenkins/.kube/config
[Pipeline] sh
+ kubectl apply -f deployment.yaml
error: You must be logged in to the server (the server has asked for the client to provide credentials)
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 1
Finished: FAILURE

```

Deployment failed — Unauthorized

You might see either of these 2 errors -

ERROR - 1:

```
+ aws eks update-kubeconfig --name my-eks-cluster
```

Added new context arn:aws:eks:us-east-1:503382476502:cluster/my-eks-cluster to
/var/lib/jenkins/.kube/config

[Pipeline] sh

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

error: You must be logged in to the server (the server has asked for the client to provide credentials)

```
+ aws eks update-kubeconfig --name my-eks-cluster
```

Updated context arn:aws:eks:us-east-1:12345678:cluster/my-eks-cluster in
/var/lib/jenkins/.kube/config

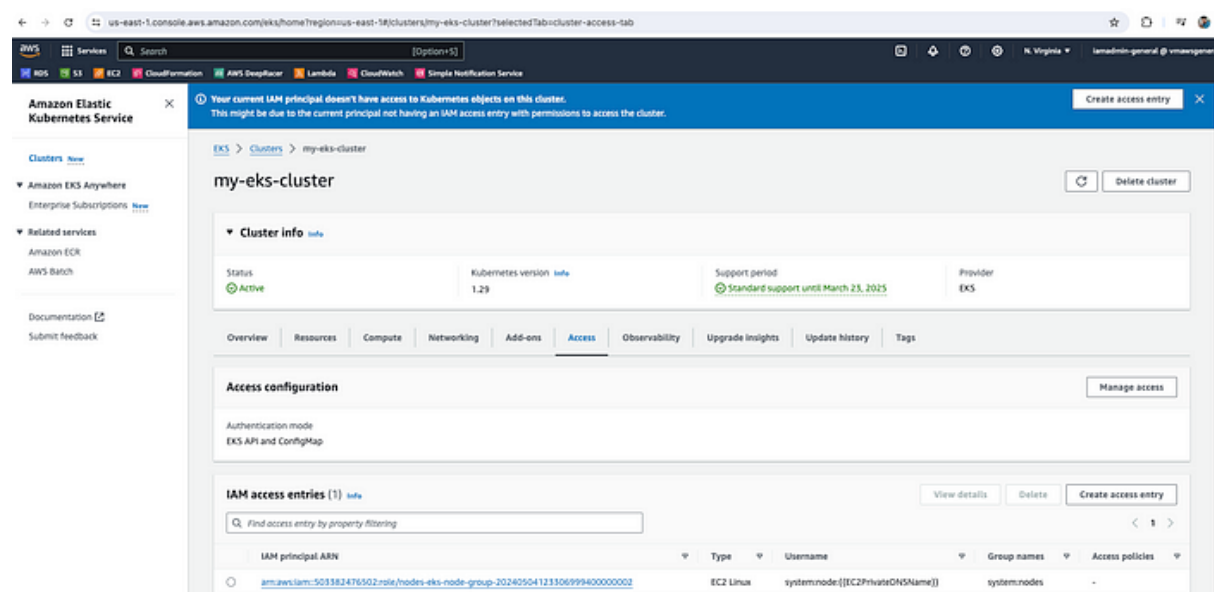
[Pipeline] sh

```
+ kubectl create namespace eks-nginx-app
```

Error from server (Forbidden): namespaces is forbidden: User
"arn:aws:iam::12345678:user/iamadmin-general" cannot create resource "namespaces" in API group
"" at the cluster scope

The problem is even though you created the EKS cluster via root user or admin user, by default
nobody has permission to access the EKS cluster. You will also see a notification in the EKS cluster
console → **Your current IAM principal doesn't have access to Kubernetes objects on this cluster.**

So, you need to create an access entry to resolve this issue.



EKS Cluster — Create access entry

Now, select the admin/root user ARN.

us-east-1.console.aws.amazon.com/eks/home?region=us-east-1#/clusters/my-eks-cluster/create-access-entry?set-default=true

Services Search [Option+S]

RDS S3 EC2 CloudFormation AWS DeepRacer Lambda CloudWatch Simple Notification Service

EKS > Clusters > my-eks-cluster > IAM access entries > Create access entry

Step 1
Configure IAM access entry

Step 2 - optional
[Add access policy](#)

Step 3
[Review and create](#)

Configure IAM access entry

IAM principal [Info](#)
The IAM principal that you want to grant access to Kubernetes objects on your cluster.

IAM principal ARN

The IAM principal ARN can't be changed after access entry creation.

Type - Optional [Info](#)
By selecting an option other than Standard, EKS automatically associates the permissions required for nodes using the IAM role for this access entry to join the cluster.

Type
Select the type of IAM access entry

The type can't be changed after access entry creation.

Username - Optional [Info](#)
If you don't specify a username, EKS will auto-generate one for you while it creates the access entry.

Username

EKS Cluster — Select IAM principal ARN

Next, select the policy/permission. Please select AmazonEKSClusterAdminPolicy then click Next and Create.

us-east-1.console.aws.amazon.com/eks/home?region=us-east-1#/clusters/my-eks-cluster/create-access-entry?set-default=true

Services Search [Option+S]

RDS S3 EC2 CloudFormation AWS DeepRacer Lambda CloudWatch Simple Notification Service

EKS > Clusters > my-eks-cluster > IAM access entries > Create access entry

Step 1
Configure IAM access entry

Step 2 - optional
Add access policy

Step 3
Review and create

Add access policy - optional

Access policies info
Select an access policy to associate to the access entry and the scope of the access policy.

Policy name
Policy to associate
AmazonEKSAAdminPolicy

Access scope
Type of access scope
☒ Cluster
☐ Kubernetes namespace

Add policy

Added policies
No access policies added.

Cancel Previous Next

Select permission — AmazonEKSClusterAdminPolicy

us-east-1.console.aws.amazon.com/eks/home?region=us-east-1#/clusters/my-eks-cluster/access-entries/arn:aws:iam::31450338247650293:user%2Fiamadmin-general

Services Search [Option+S]

RDS S3 EC2 CloudFormation AWS DeepRacer Lambda CloudWatch Simple Notification Service

Amazon Elastic Kubernetes Service

Clusters
Amazon EKS Anywhere
Enterprise Subscriptions
Related services
Amazon ECR
AWS Batch
Documentation
Submit feedback

IAM access entry info

IAM principal ARN arn:aws:iam::[redacted]:user/iamadmin-general	Type Standard	Created 2 hours ago
Username arn:aws:iam::[redacted]:user/iamadmin-general	Access entry ARN arn:aws:eks:us-east-1:[redacted]:access-entry/my-eks-cluster/user/[redacted]/iamadmin-general/1a7c7a1fe-ee58-40da-e874-920754949c94	

Group names (0)

The group names that you've specified as subjects in Kubernetes RoleBinding or ClusterRoleBinding objects.

Filter by group name

Group name

No Kubernetes groups
No Kubernetes groups to display

Access policies (1)

The access policies associated to the access entry and Kubernetes namespaces that you've scoped the access policies to.

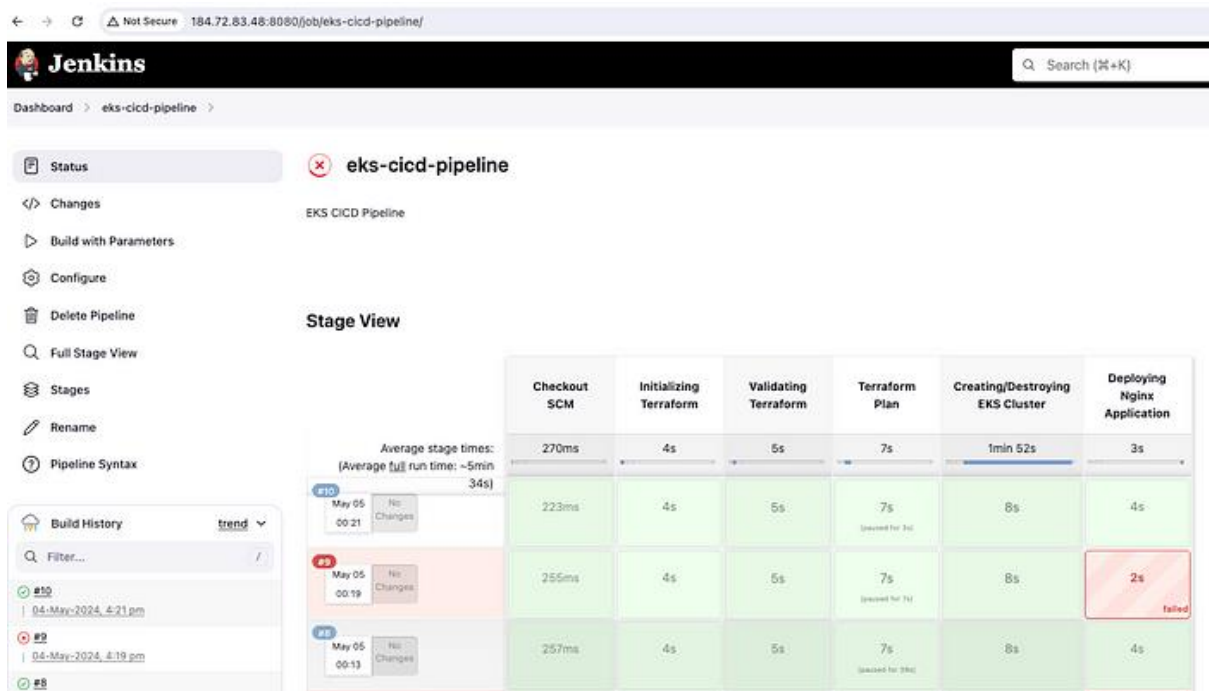
Filter by policy name or Kubernetes namespaces

Policy name	Kubernetes namespaces
<input type="radio"/> AmazonEKSClusterAdminPolicy	

Remove Edit Add access policy

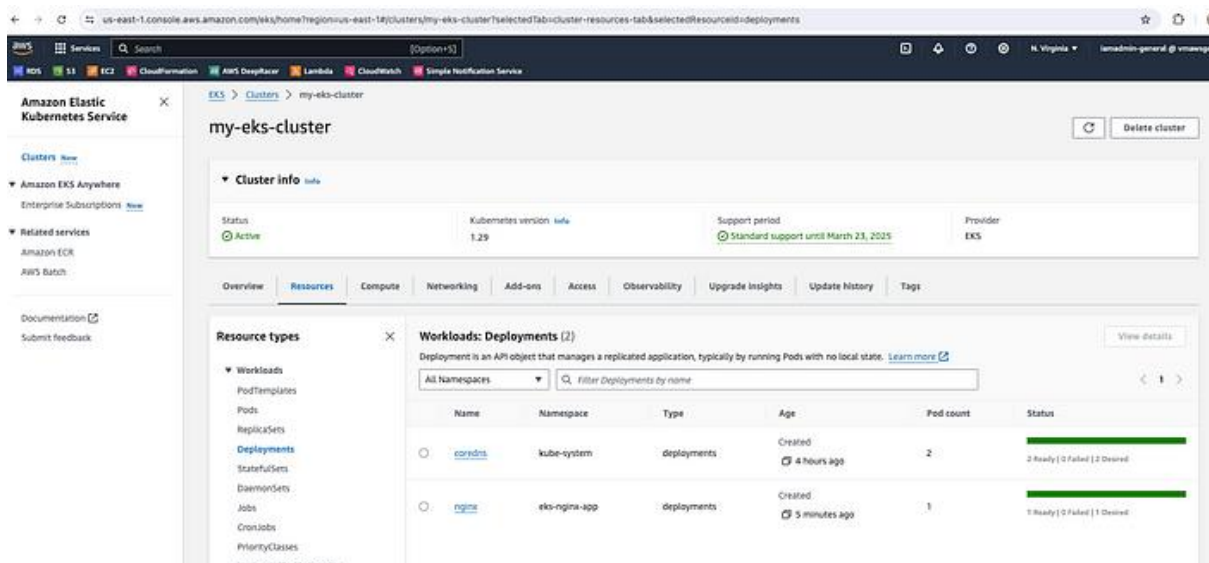
Access Entry created

Let's rerun the pipeline and check the status. This time our pipeline will be successful.

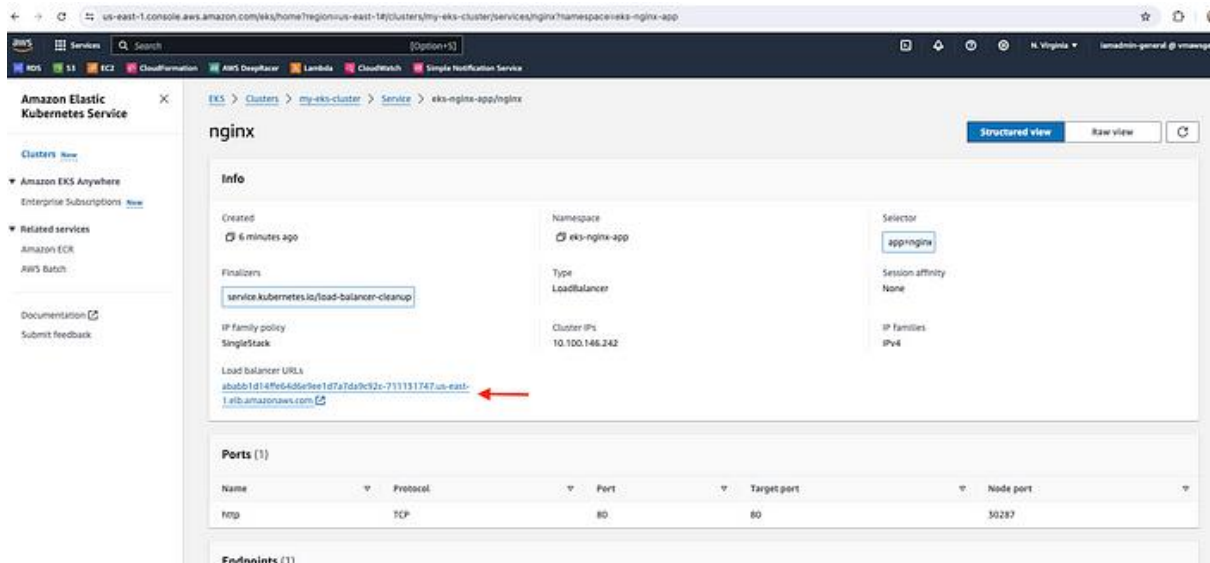


eks-cicd-pipeline successful

Let's validate the resources in the AWS EKS console -



nginx deployment is running



nginx service is running

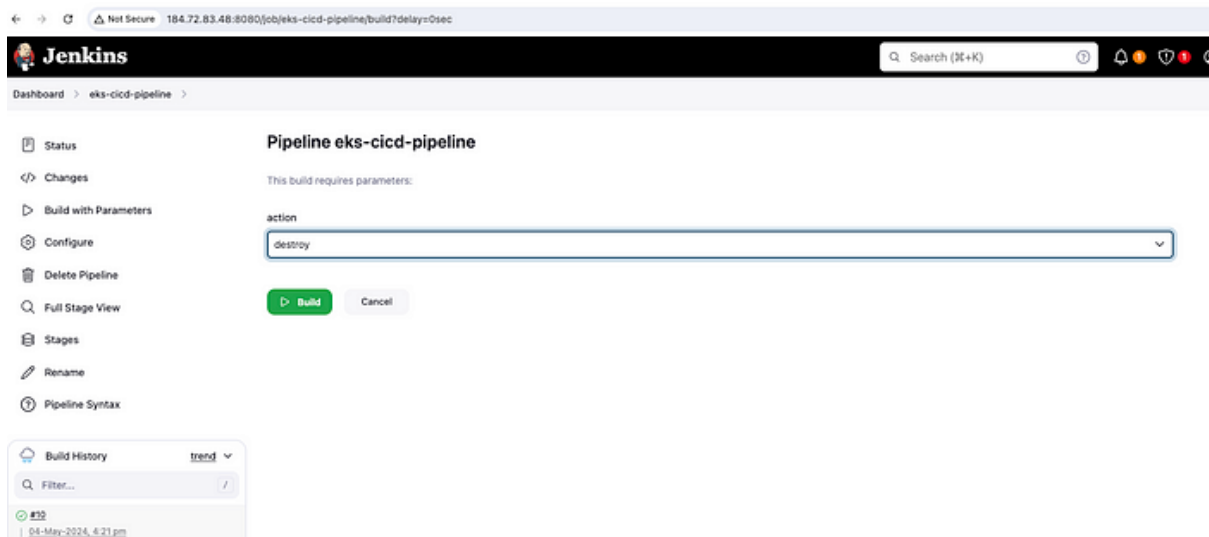
Now copy the load balancer URL and hit it in the browser. We'll be able to access the application.



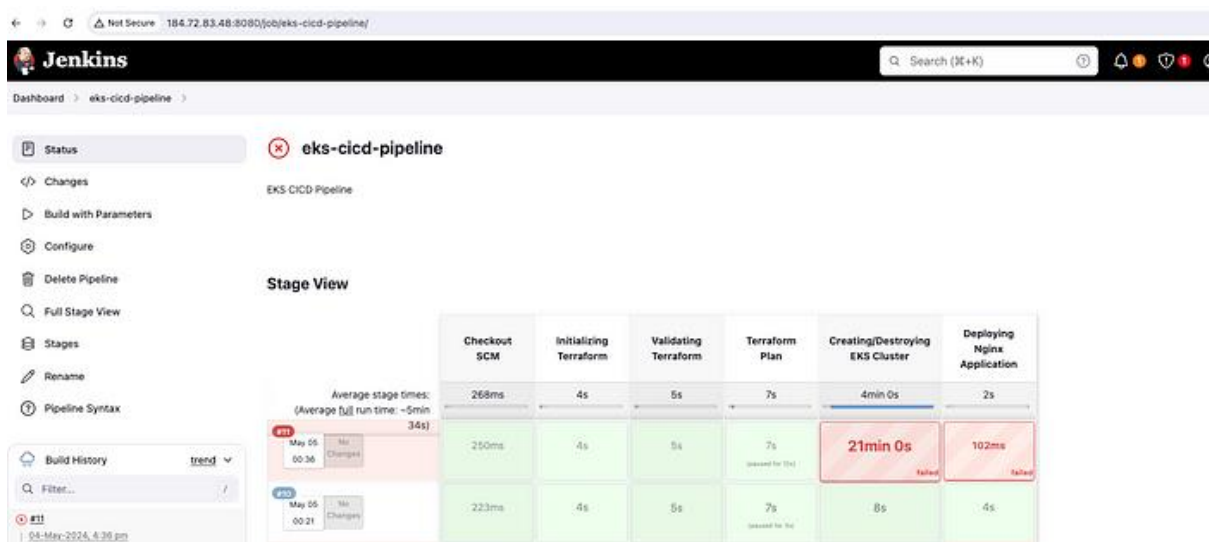
Nginx application is running in the browser

Stage 5: Teardown resources

Finally, we have come to the end of this guide. We have to destroy our resources to save on the cost. Deleting applications and destroying EKS cluster can be done via the Jenkins pipeline by just selecting the action destroy while doing Build with Parameters.

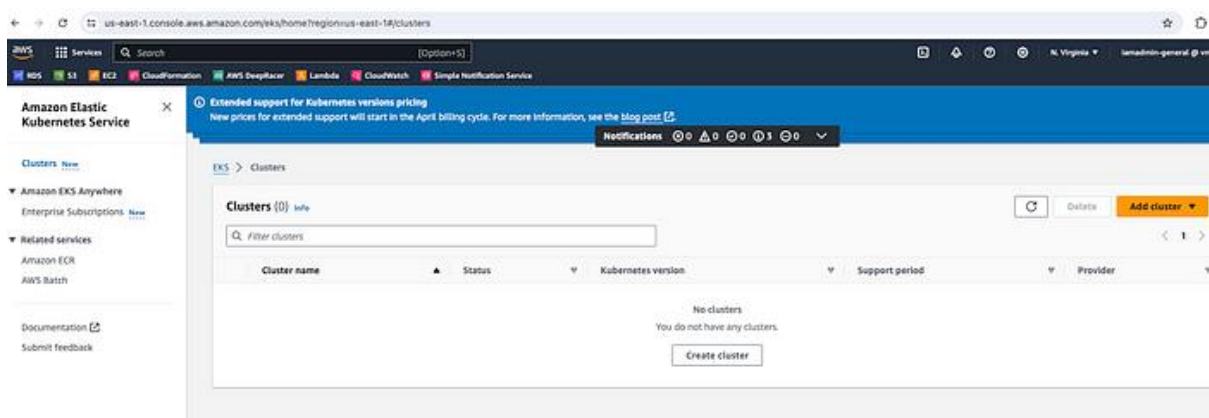


Action — destroy



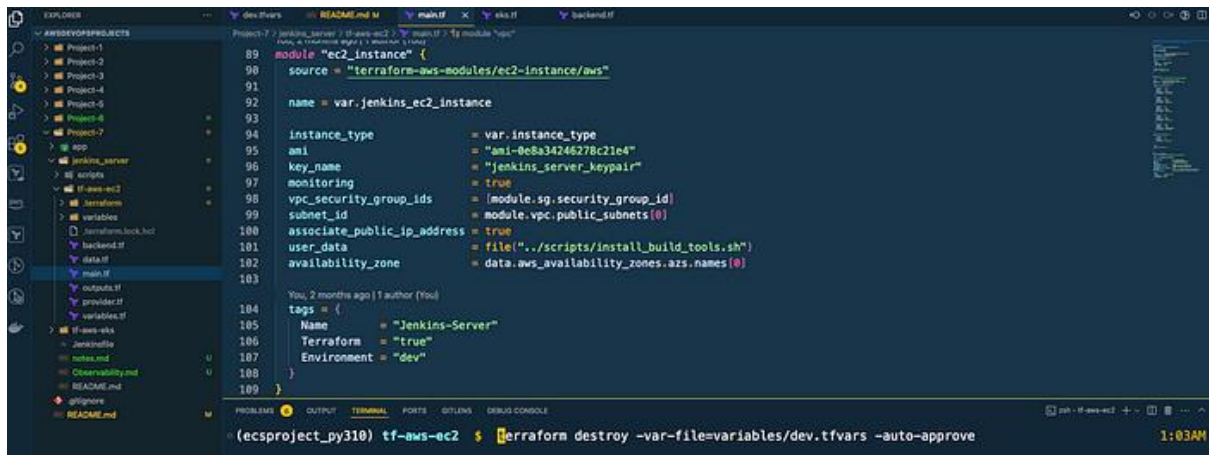
Destroy Pipeline

Even though my pipeline has failed, I can see there are no EKS clusters in the AWS console.



EKS Cluster — Deleted

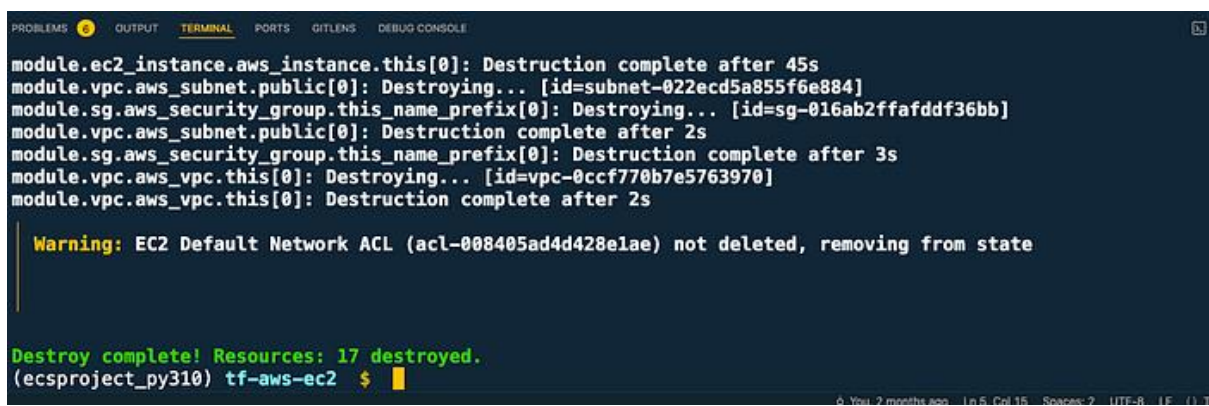
Let's also delete the Jenkins Server by running terraform destroy via local CLI.



```
89 module "ec2_instance" {
90   source = "terraform-aws-modules/ec2-instance/aws"
91
92   name = var.jenkins_ec2_instance
93
94   instance_type           = var.instance_type
95   ami                    = "ami-0c8a34246278c21e4"
96   key_name                = "jenkins_server_keypair"
97   monitoring              = true
98   vpc_security_group_ids = [module.sg.security_group_id]
99   subnet_id              = module.vpc.public_subnets[0]
100   associate_public_ip_address = true
101   user_data               = file("../scripts/install_build_tools.sh")
102   availability_zone       = data.aws_availability_zones.aws.names[0]
103
104   tags = {
105     Name        = "Jenkins-Server"
106     Terraform   = "true"
107     Environment = "dev"
108   }
109 }
```

(ecsproject_py310) tf-aws-ec2 \$ terraform destroy -var-file=variables/dev.tfvars -auto-approve

terraform destroy



```
module.ec2_instance.aws_instance.this[0]: Destruction complete after 45s
module.vpc.aws_subnet.public[0]: Destroying... [id=subnet-022ecd5a855f6e884]
module.sg.aws_security_group.this_name_prefix[0]: Destroying... [id=sg-016ab2ffafdddf36bb]
module.vpc.aws_subnet.public[0]: Destruction complete after 2s
module.sg.aws_security_group.this_name_prefix[0]: Destruction complete after 3s
module.vpc.aws_vpc.this[0]: Destroying... [id=vpc-0ccf770b7e5763970]
module.vpc.aws_vpc.this[0]: Destruction complete after 2s

Warning: EC2 Default Network ACL (acl-008405ad4d428e1ae) not deleted, removing from state

Destroy complete! Resources: 17 destroyed.
(ecsproject_py310) tf-aws-ec2 $
```

terraform destroy completed

Please recheck your AWS Console manually to see if there is any resource remaining for example — EC2 key pair and S3 bucket and delete them manually.

Conclusion

We have successfully implemented a robust and automated infrastructure provisioning and deployment pipeline using Terraform, EKS, and Jenkins. We have not only designed and implemented a scalable and efficient CI/CD pipeline but also deployed a simple **Nginx** application in the EKS cluster. However, this is not the end but the beginning of creating complex production CI/CD applications.

Further Improvements

There are a lot of areas for improvement in this pipeline. Some of them are as below —

- **CI/CD Pipeline Enhancements:** We can explore additional Jenkins features, such as automated trigger, code review, testing, and artifact management, to further streamline our pipeline.
- **Security Enhancements:** Implement additional security measures, such as network policies, secret management, and role-based access control.
- **Kubernetes Advanced Features:** Experiment with Kubernetes advanced features, like StatefulSets, Deployments, and Persistent Volumes, to improve our application's resilience and efficiency.

- **Monitoring and Logging:** Integrate monitoring tools (e.g., Prometheus, Grafana) and logging solutions (e.g., ELK Stack) to ensure real-time visibility and insights.