An Industry Grade Project For PGP In DevOps

Purdue - Edureka

Git repo – https://github.com/Kowshikreddy137/IGP_Purdue/tree/master

Kowshik reddy Beeravolu

Overview

This project is for ABC Technologies an online retail business, which has acquired a large offline retail business store. Because of the previous company's approaches, ABC Technologies will face significant challenges such as low availability, scalability, performance, and complex maintenance. The goal is to create a seamless integration and delivery pipeline using modern DevOps tools and practices, enhancing the development, testing, packaging, and deployment processes.

Goals

The primary goals of this project are to implement a CI/CD pipeline that enables ABC to:

- Achieve high availability and scalability
- Improve performance
- Simplify building and maintenance
- Expedite development and deployment processes

Milestones

1. The code is pushed to GitHub.
2. Jenkins automates the build process (compile, test, package).
3. Docker is used for containerization.
4. Ansible manages deployment.
5. Kubernetes orchestrates the deployment.
6. Prometheus and Grafana monitor the system.

Pre-requisites

These are the tools and application softwares that are part of the solution.

1. Java
2. Maven
3. Git
4. AWS
5. Jenkins
6. Docker
7. Ansible

8. Kubernetes
9. Grafana
10. Prometheus

Here is the link to my git repo

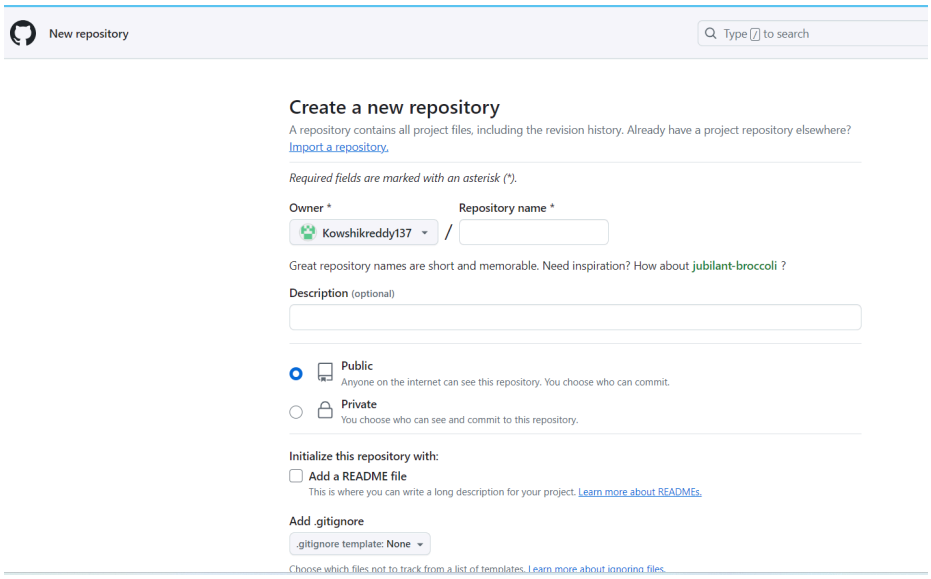
Git repo – https://github.com/Kowshikreddy137/IGP_Purdue/tree/master

Approach to solve

The approach to solving the problem is a multi-step process involving various DevOps tools.

Task 1: Source Code Management

1. Created a new repository on GitHub named IGP_Purdue.



2. Installed git on my local machine and configured git (Already set up)
3. Downloaded the source code from the Edureka website and initiated the git repository in the folder.
4. Using the *git add* command to stage the contents of the folder for the initial commit.
5. Using the *git commit* command to make an initial commit there by recording the staged files.

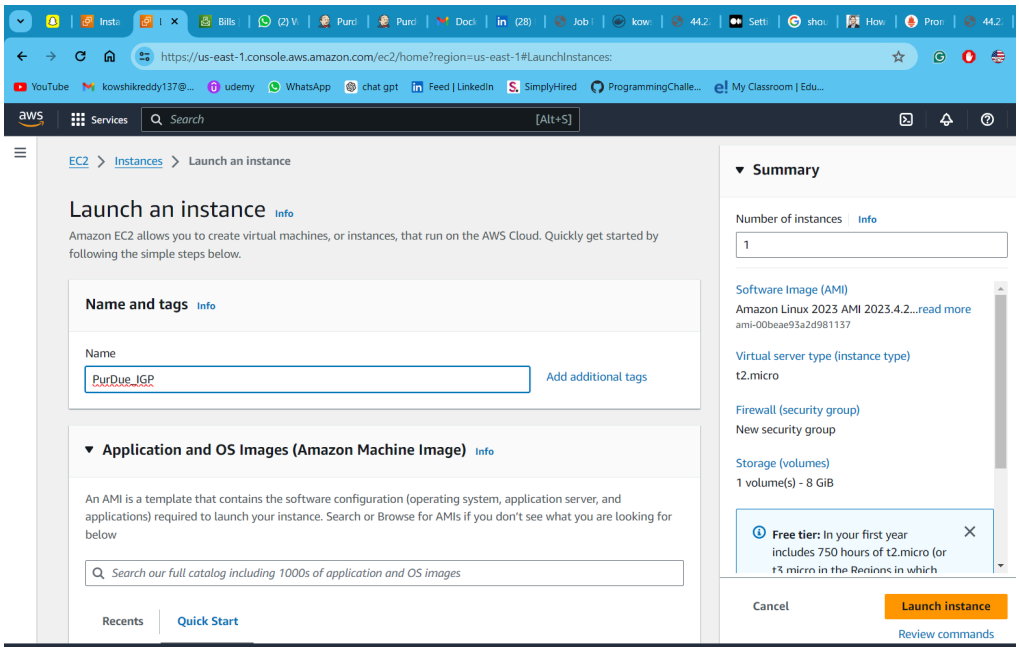
6. As we are using a remote repository, we have to connect to the remote repository using the *git remote add origin git@github.com:Kowshikreddy137/IGP_Purdue.git* command.
7. And then finally we can push our initial commit to the GIT repository using the command *git push -u origin*

```
kowsh@LAPTOP-M7C7I22R MINGW64 ~/Downloads/ABC Technologies
$ git init
Initialized empty Git repository in C:/Users/kowsh/Downloads/ABC Technologies/.git/

kowsh@LAPTOP-M7C7I22R MINGW64 ~/Downloads/ABC Technologies (master)
$ git add .
warning: LF will be replaced by CRLF in src/main/webapp/WEB-INF/web.xml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/main/webapp/index.jsp.
The file will have its original line endings in your working directory

kowsh@LAPTOP-M7C7I22R MINGW64 ~/Downloads/ABC Technologies (master)
$ git commit " Innitial commit"
error: pathspec ' Innitial commit' did not match any file(s) known to git

kowsh@LAPTOP-M7C7I22R MINGW64 ~/Downloads/ABC Technologies (master)
$ git commit -m "Innitial commit"
[master (root-commit) 8c79fde] Innitial commit
 13 files changed, 313 insertions(+)
 create mode 100644 .classpath
 create mode 100644 .project
 create mode 100644 .settings/org.eclipse.jdt.core.prefs
 create mode 100644 .settings/org.eclipse.m2e.core.prefs
 create mode 100644 README.md
 create mode 100644 pom.xml
 create mode 100644 pom.xml.bak
 create mode 100644 src/main/java/com/abc/RetailModule.java
 create mode 100644 src/main/java/com/abc/dataAccessObject/RetailAccessObject.java
 create mode 100644 src/main/java/com/abc/dataAccessObject/RetailDataImp.java
 create mode 100644 src/main/webapp/WEB-INF/web.xml
 create mode 100644 src/main/webapp/index.jsp
 create mode 100644 src/test/java/com/abc/dataAccessObject/ProductImpTest.java
```

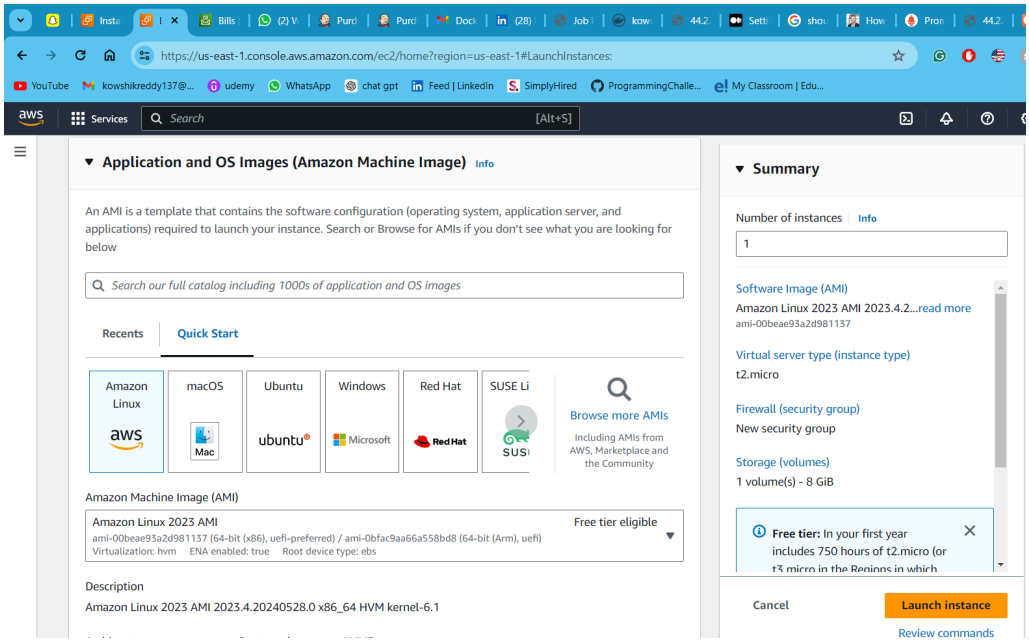
Task 2: Creating an Ec2 Instance

We will be using a Linux machine provisioned on the Ec2 Instance on an AWS to proceed with the next steps.

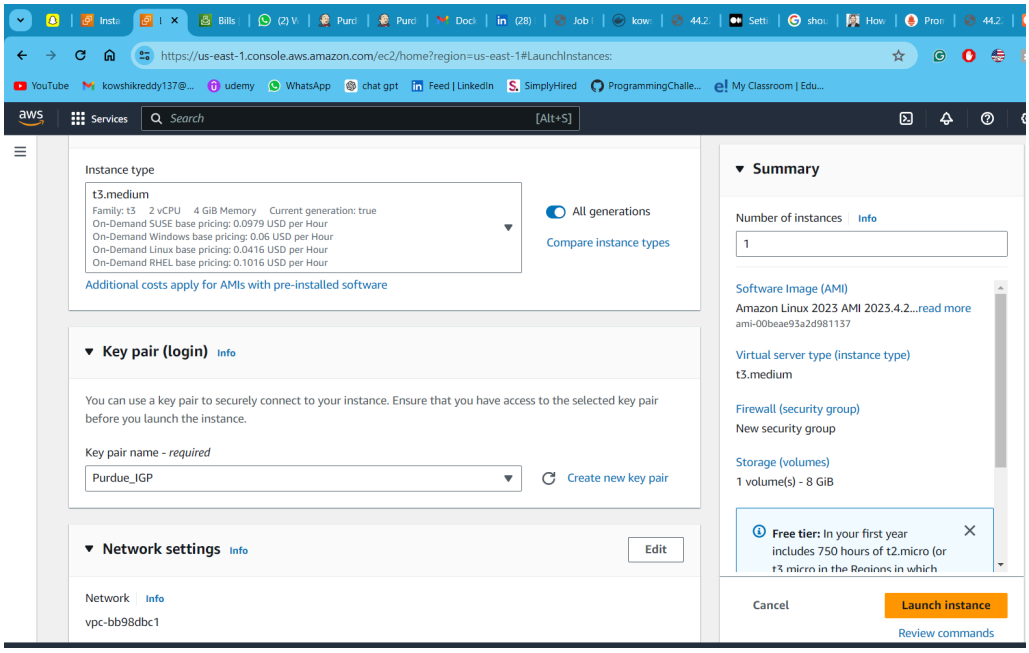
1. In the AWS Management Console, locate and select “EC2” under the “Services” menu.
2. On the EC2 Dashboard, click the “Launch Instance” button to launch the instance.
3. Add a Name to your instance to help identify and manage your instances.



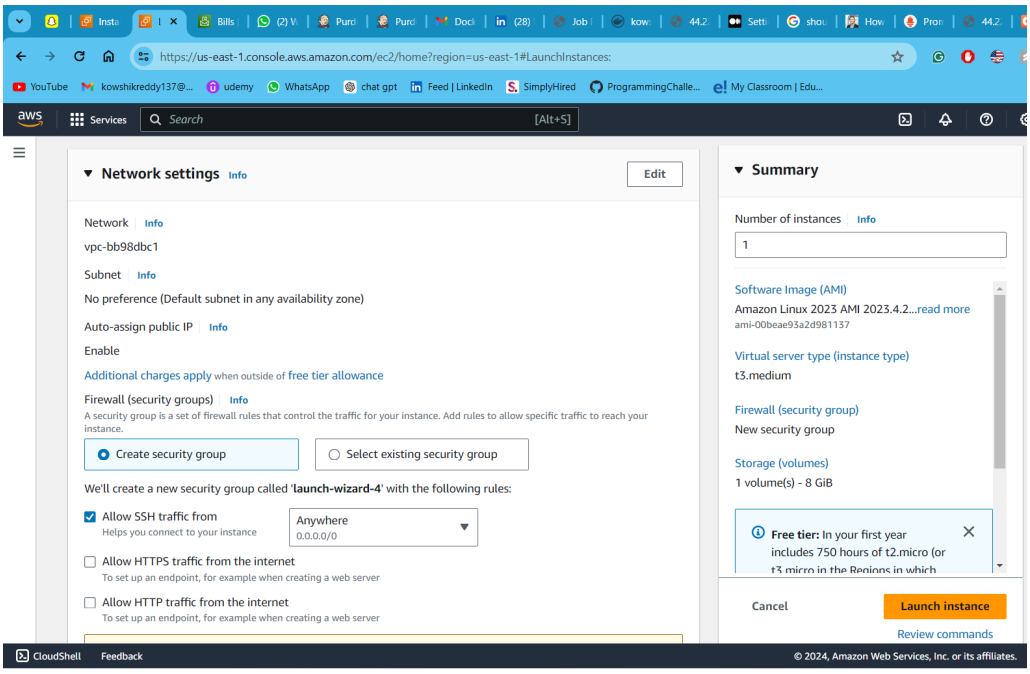
4. Select an instance type based on your required CPU, memory, storage, and network performance.



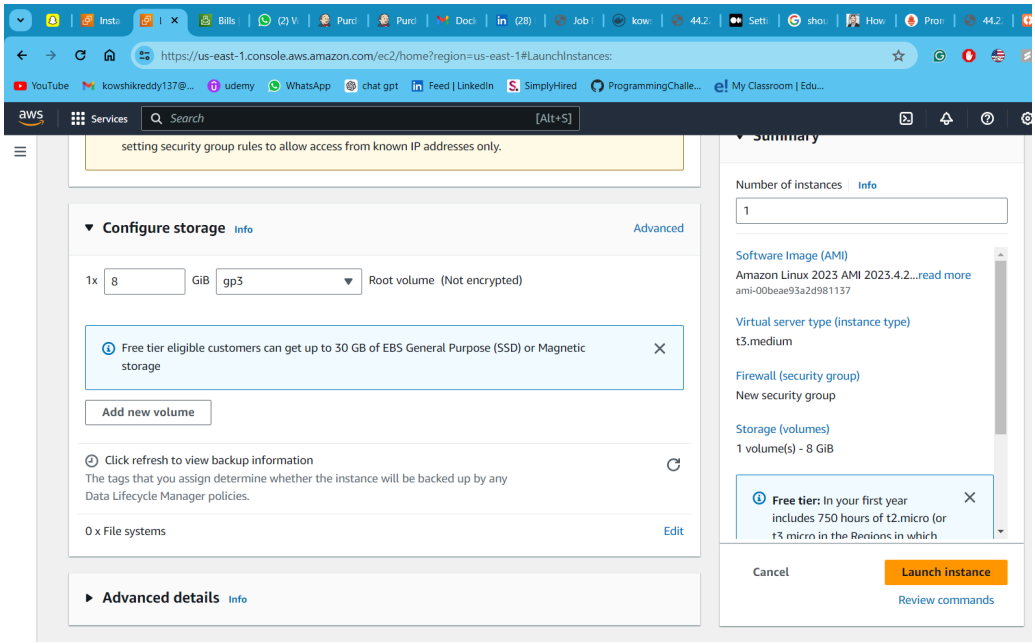
5. Select an existing key pair or create a new one to securely connect to your instance.



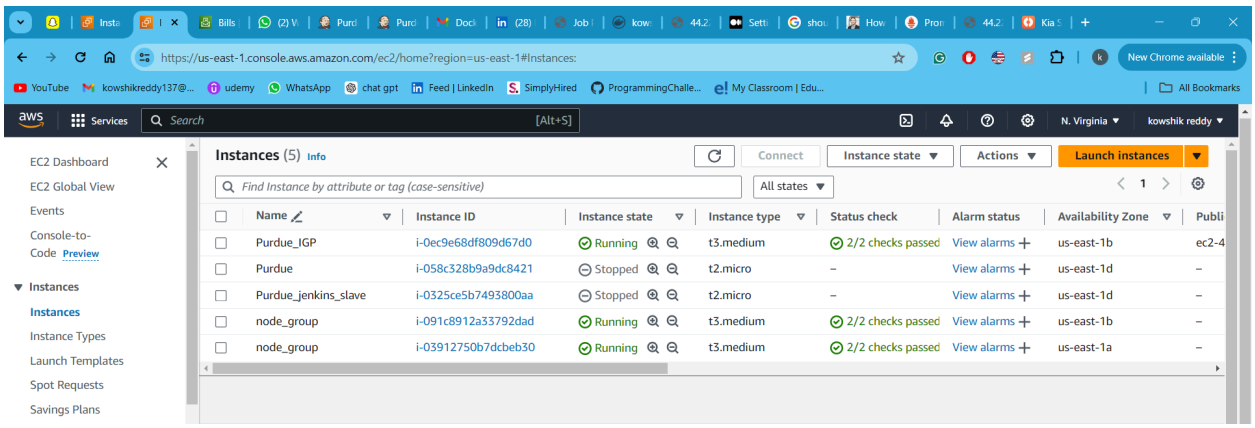
6. Configure networking settings, including the VPC, subnet, and auto-assign public IP options.
7. Create a new security group or select an existing one to define firewall rules for your instance.



8. Configure the storage volume for your instance. You can add additional volumes or increase the size of the root volume.
9. Click “Review and Launch” after configuring the security group.



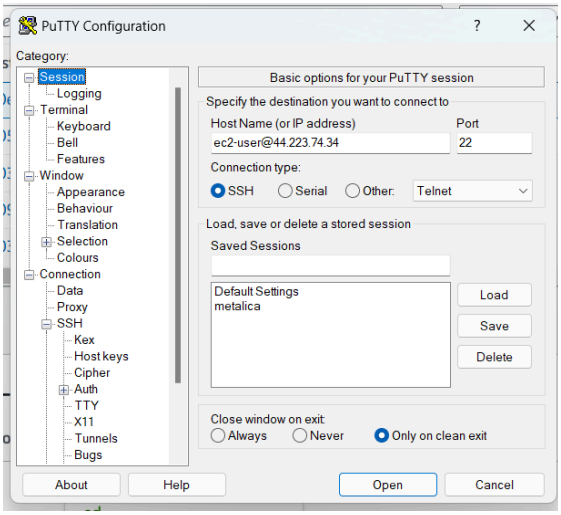
Finally, this is how we will see the instances on the dashboard.



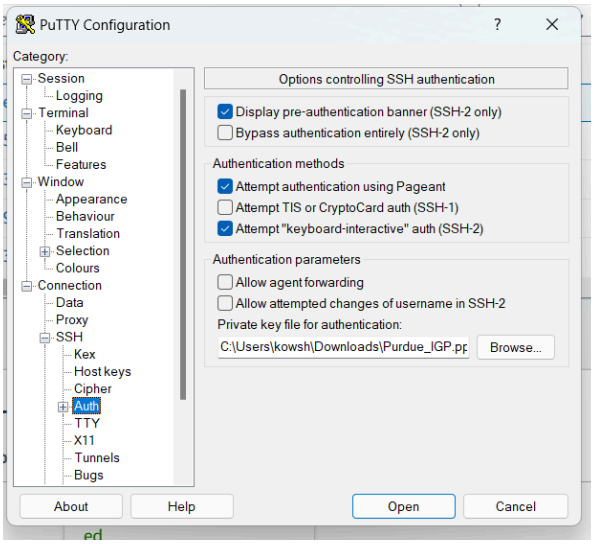
Task 3: Jenkins Pipeline

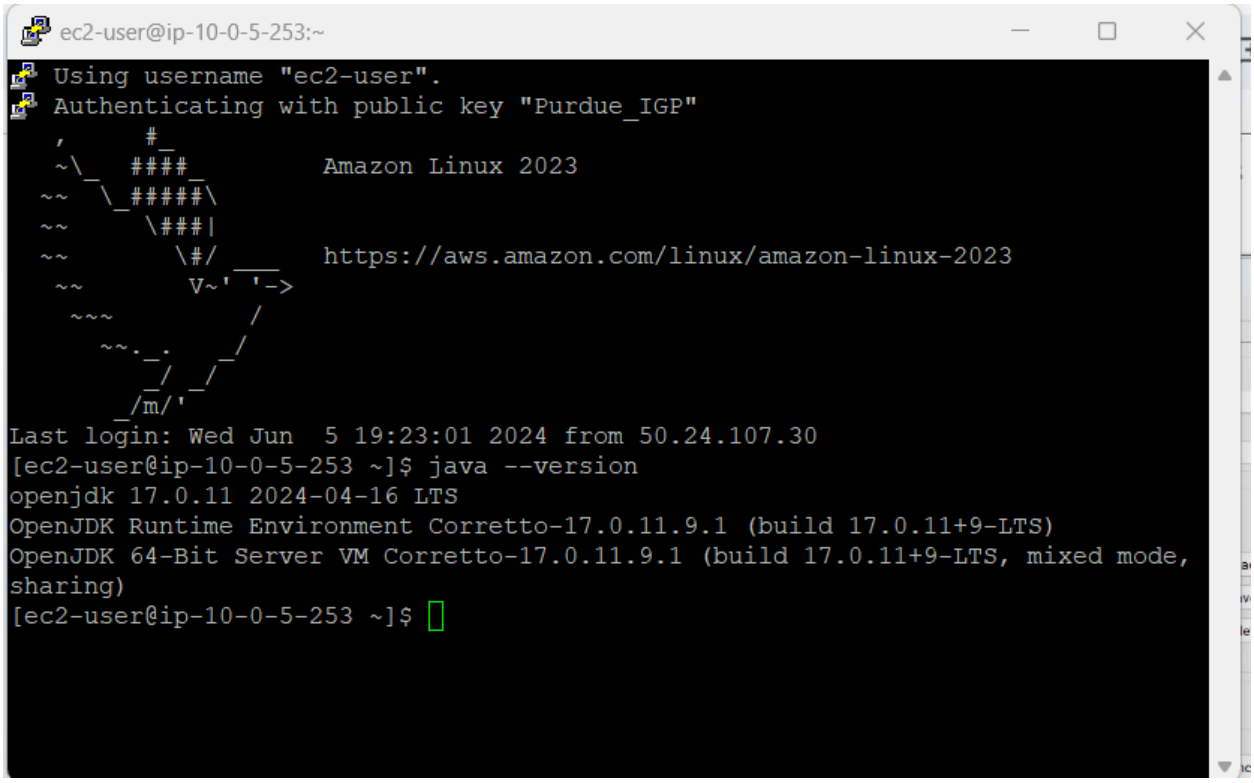
Connecting to the Instance

1. First, we have to connect to our ec2 instance (Linux machine) which we have created in the previous step.
2. To do that we will be using putty and when the dialogue box opens we can see the session window in which we have to enter the hostname.
3. As I was using an amazon Linux my machine username is ***ec2-user*** filled by the IP Address of the instance we created.



4. Then we have to navigate to the **connection tab** and then to the **Auth tab** (authentication) where we will browse for our pem file (the key pair which we selected while creating the instance) and then click on open to establish a connection to instance.





This is how a window will open and we can start working on the next steps.

Installing Jenkins

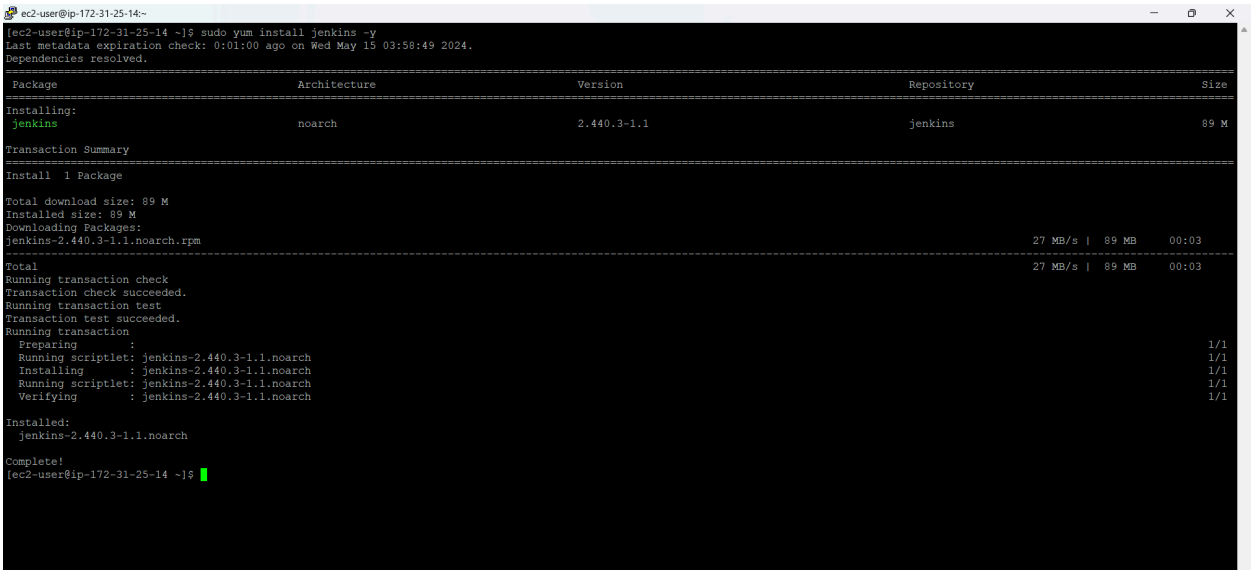
1. Jenkins requires Java to run. Amazon Linux 2 doesn't come with Java by default, so you need to install it. By following the commands below i have installed Java on the Linux machine.

```
$ sudo dnf install java-17-amazon-corretto -y

$ java --version

$ echo $JAVA_HOME

$ export JAVA_HOME=/usr/lib/jvm/java-17-openjdk
```

2. **Add Jenkins repository**: Jenkins is not available in the default Amazon Linux 2 repositories, so you need to add the Jenkins repository.


```
$ 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.key
```

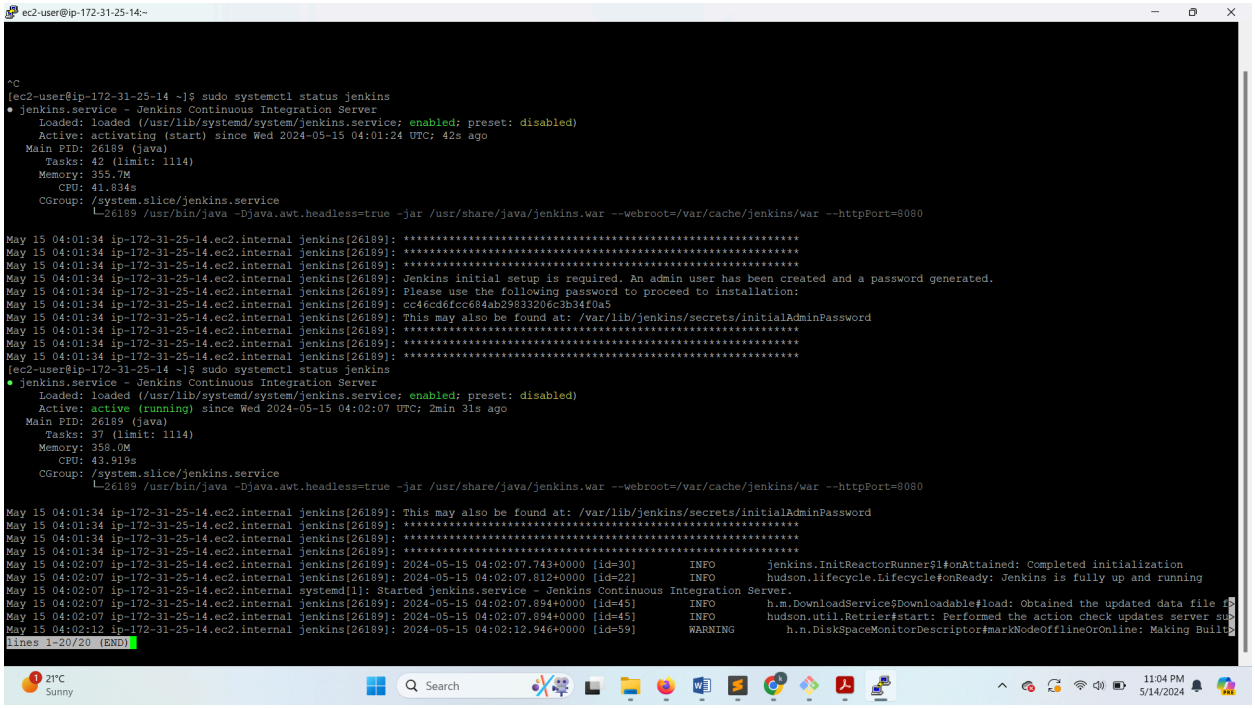
3. With the repository added, we can now install Jenkins.



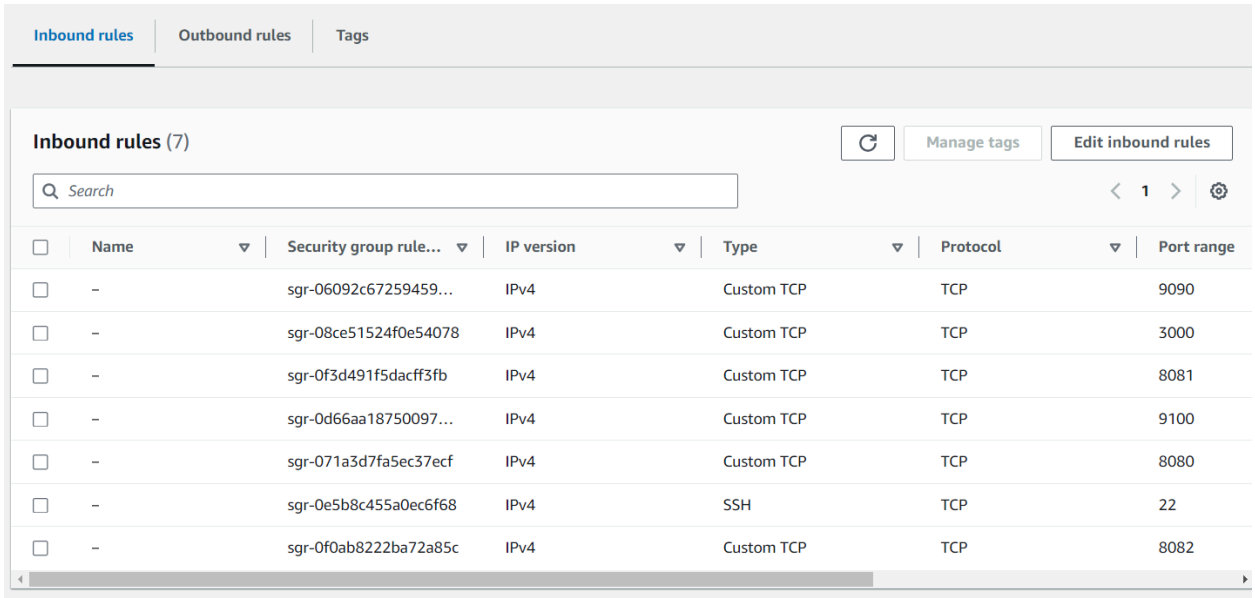
4. After installation, start and enable the Jenkins service so that it starts automatically upon system boot

```
$ sudo systemctl start jenkins
$ sudo systemctl enable jenkins
```

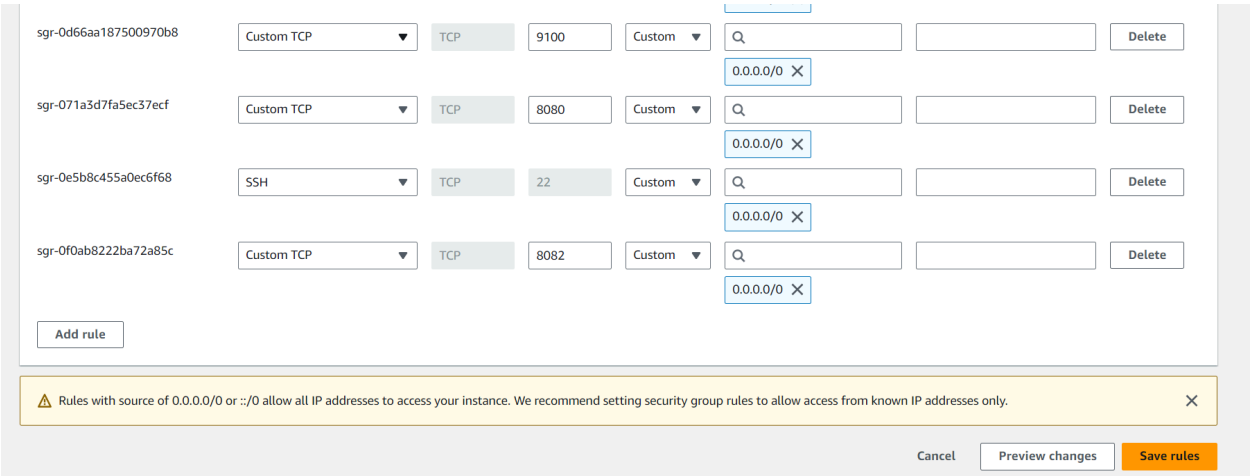
5.



6. Enable the 8080 port on the Ec2 Machine security group to allow traffic to that particular port. We can do that by selecting the instance and below we will be able to see information about the security groups.
7. Click on security group, select inbound rules, and click on edit.



8. We can add the required ports and fill in the information like port type, range, and source then click on save rules.

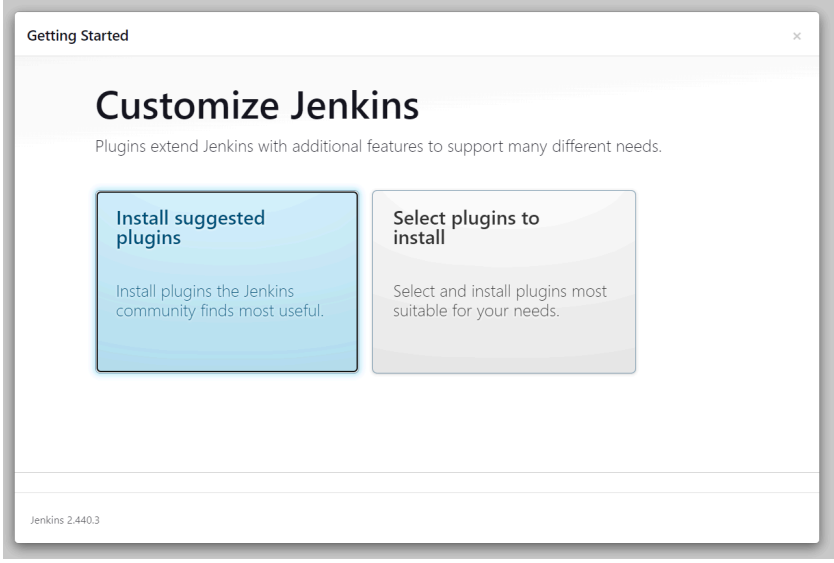


9. Now we can access our Jenkins Machine on the machine's IP Address ,followed by the port number 8080.

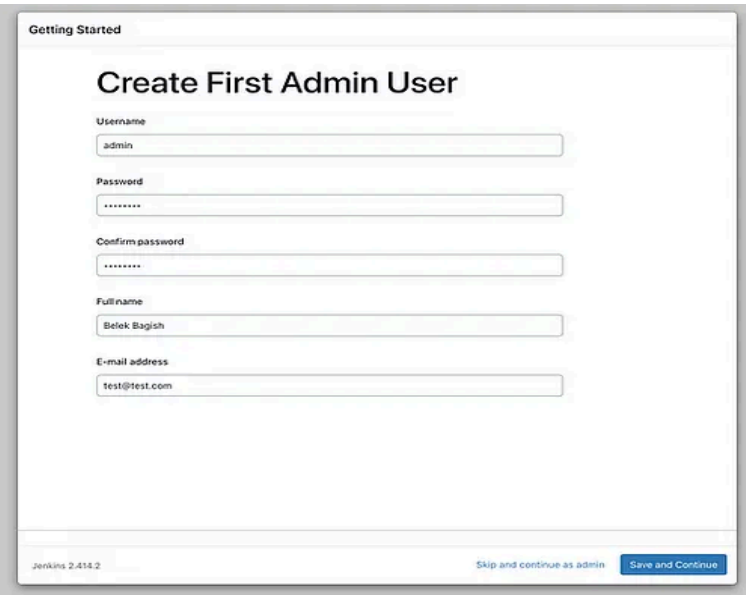
http://<ip-address>:8080

10. This will redirect us to a page where we have to enter the Administrator password as we are logging in for the first time. We will be able to find the password from this location ***/var/lib/jenkins/secrets/initialAdminPassword***

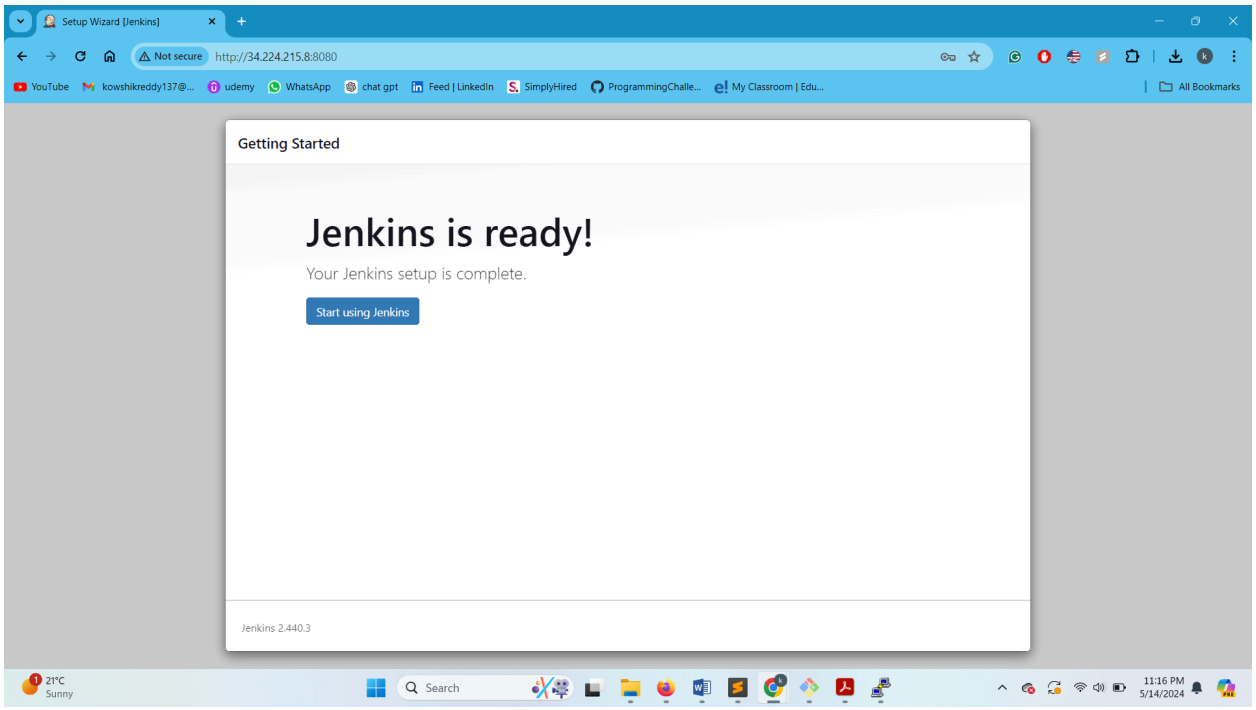
After accessing Jenkins in your web browser, we will be greeted by the Jenkins setup wizard. We will be prompted to install plugins. You can opt for installing suggested plugins or select plugins to install based on your preference. The suggested plugins cover most general use cases.



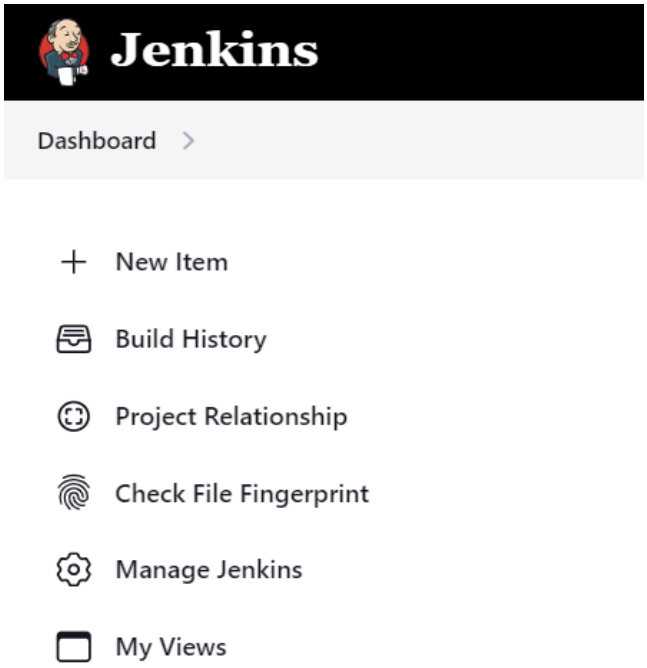
Next, We will set up the first admin user. Fill in the username, password, name, and email fields as prompted.



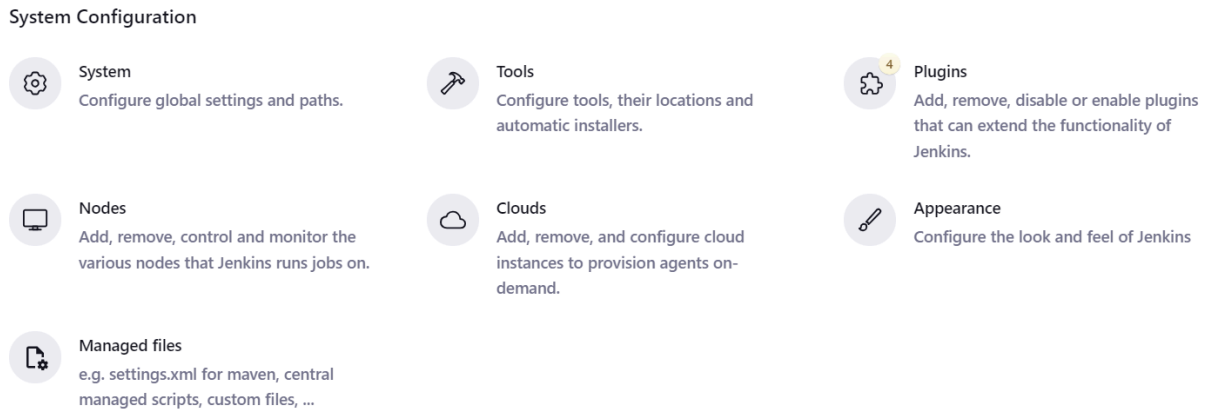
After we click on save and continue. Jenkins is ready to use.



Then we can install required plugins like maven, git, ansible, and docker by using the manage Jenkins option on the dashboard.

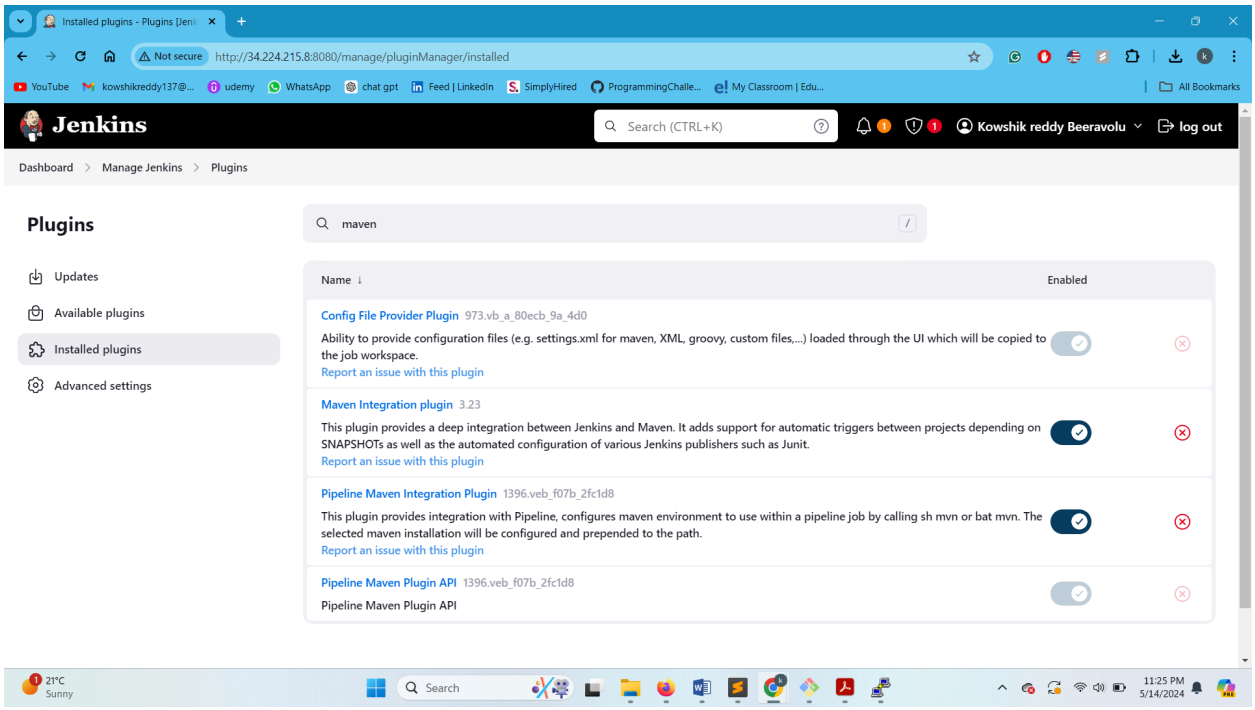


Then we will be redirected to Manage Jenkins. The "Manage Jenkins" page in Jenkins is the central hub for administrative tasks and configurations. It provides various options for managing the Jenkins environment, plugins, security settings, and more.

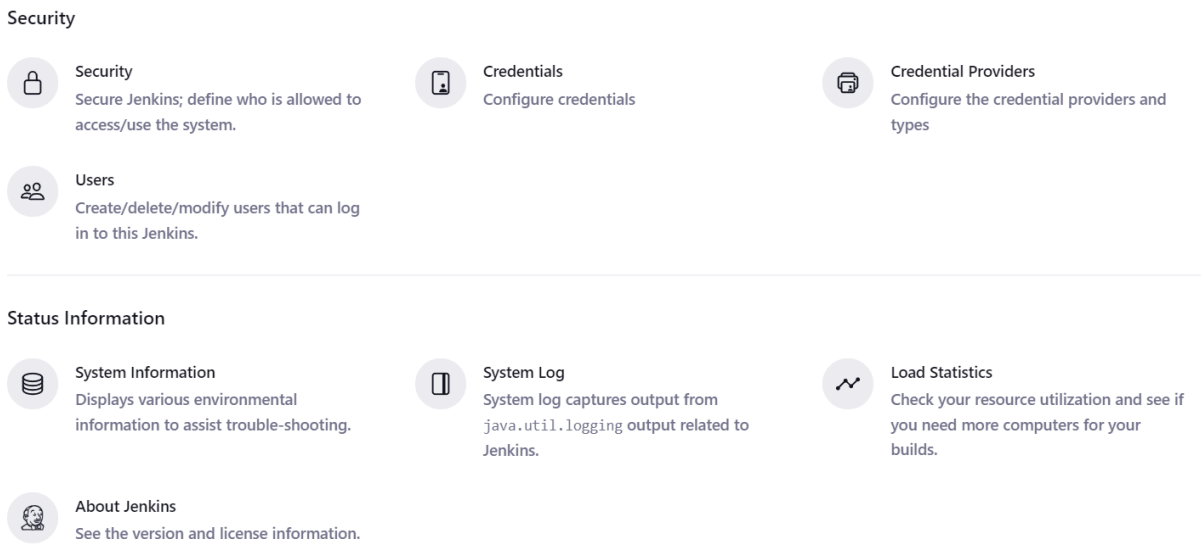


In the plugins section, we can install new plugins, and manage available plugins.

In the Nodes section, it allows you to configure and manage Jenkins agents (nodes) and cloud configurations for distributed builds.



Manage credentials that are used by Jenkins for various tasks such as accessing repositories or other external systems.



System Information displays detailed information about the Jenkins environment, including the Java version, environment variables, and system properties.

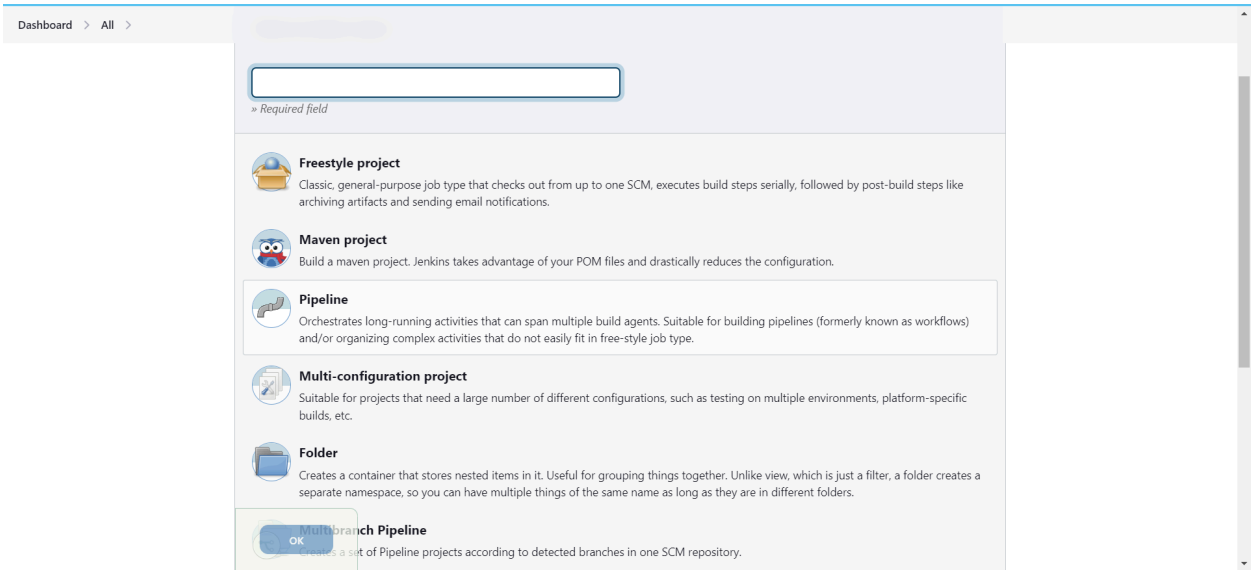
Creating Jenkins Pipeline

1. Click on New Item in the left sidebar.
2. Provide a name for your new pipeline job.
3. Choose Pipeline as the job type and click OK.
4. After creating the pipeline job, we will be directed to the configuration page.
5. Scroll down to the Pipeline section.
6. Choose the Definition type:

   Pipeline script: Here we can enter the pipeline script directly in the Jenkins interface.

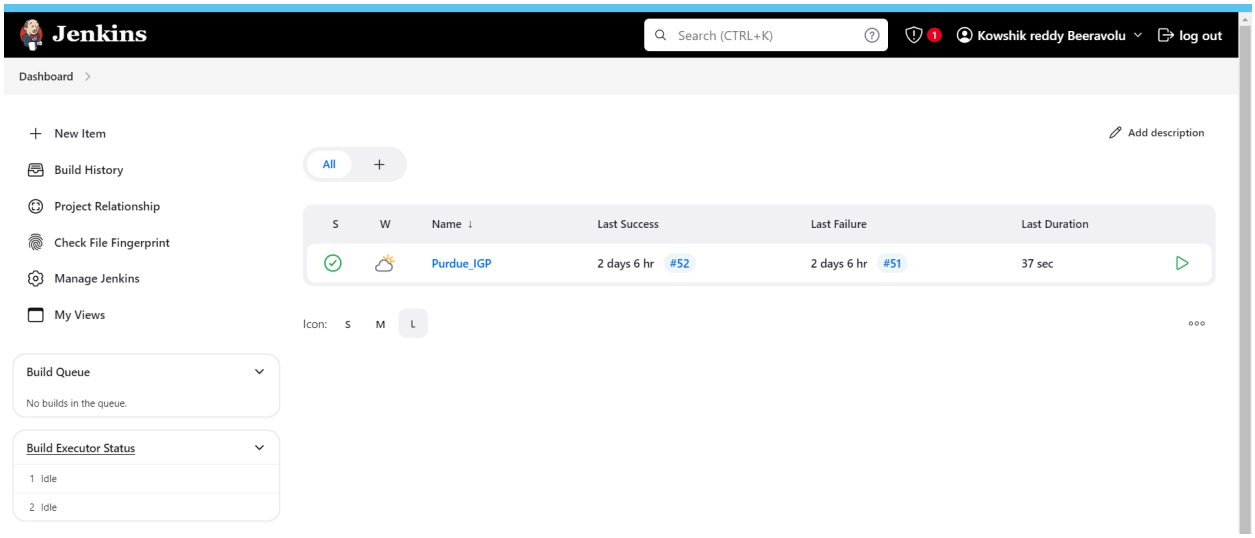
7. After configuring the pipeline, click **Save** at the bottom of the configuration page.
8. To run the pipeline, click **Build Now** in the left sidebar of the pipeline job page.
9. We can monitor the build progress and view logs by clicking on the build number in the Build History section.

Selecting a pipeline job and giving the pipeline a name.



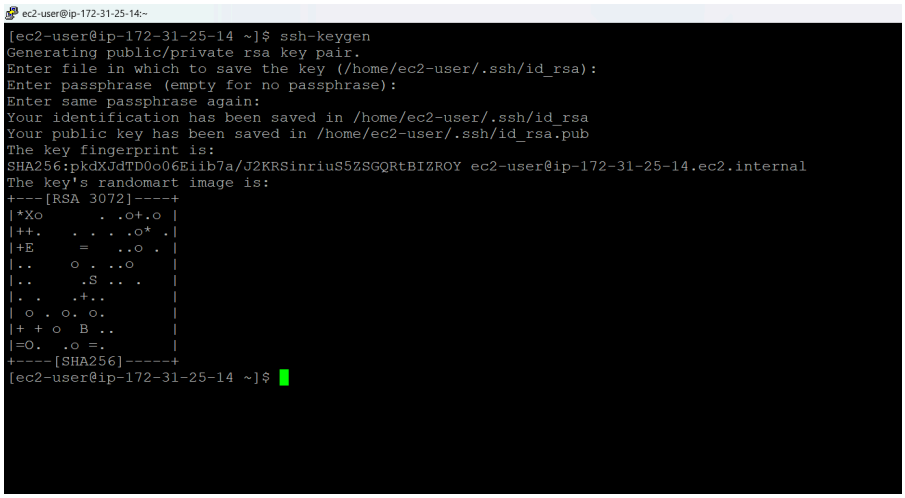
Selecting the pipeline script as the definition and writing the pipeline script which includes various stages of the pipeline like Code Checkout, code build, code test, and Environment variables.

This is how it will appear after we save the pipeline script.

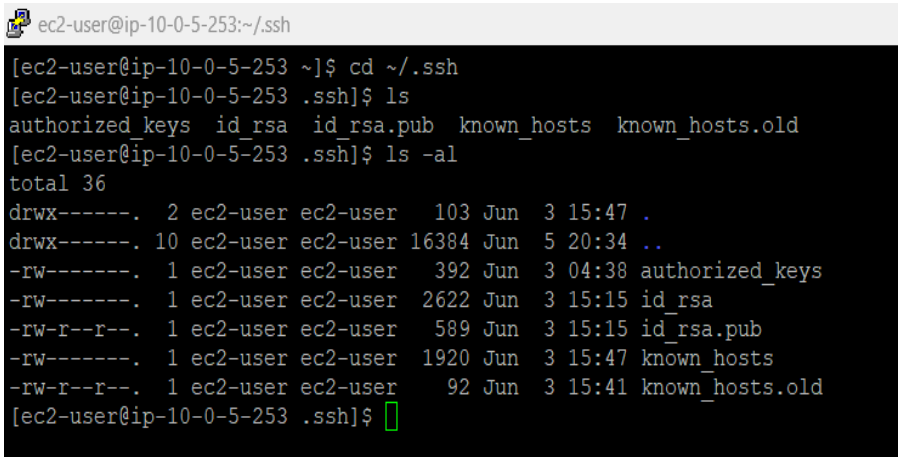


Now we will have to configure credentials to run the pipeline as we have to checkout the code from the GIT. To do this I have followed the steps,

First I have generated an SSH key on my ec2 machine by using **the *ssh-keygen*** command.

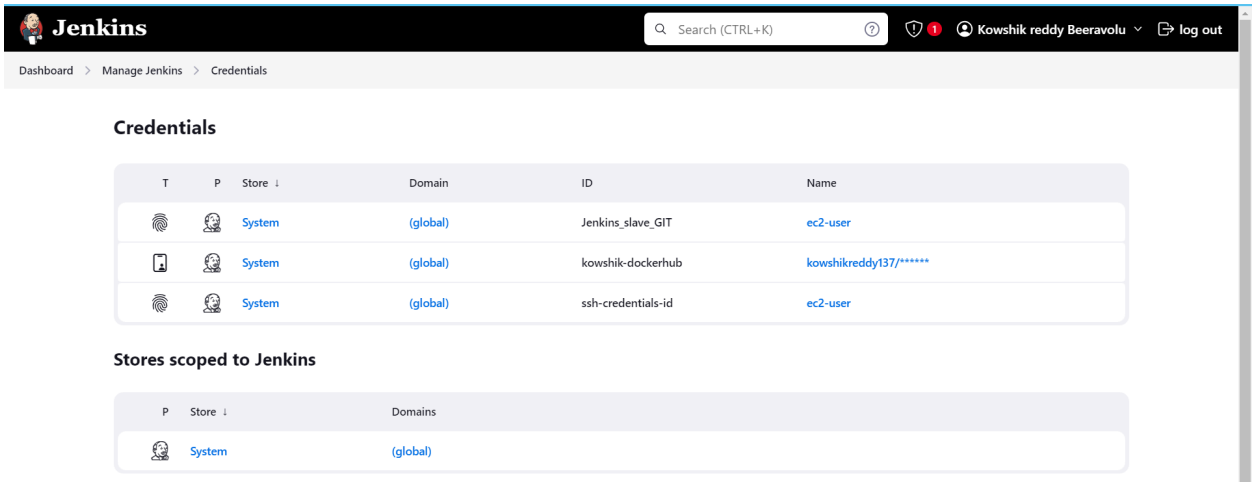


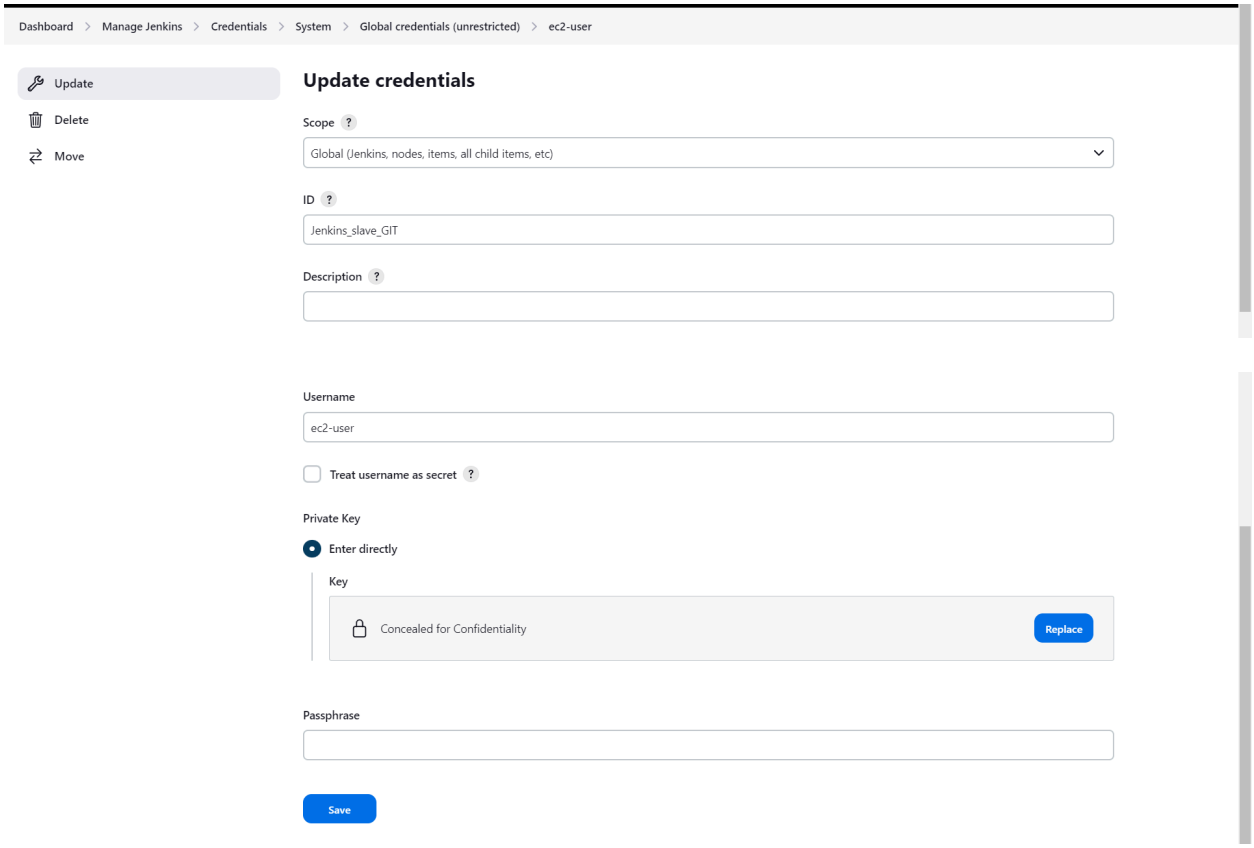
Then i can navigate to *~/.ssh* location on my ec2 machine and check for the keys and copy the id_rsa file (private key) and paste it the appropriate place in the credentials manager.



Navigate to Credential Management

1. On the Jenkins dashboard, click on Manage Jenkins in the left sidebar.
2. Click on Manage Credentials under the Security section.
3. In the Manage Credentials page, select the appropriate store. The default store is typically (global).
4. Click on the Add Credentials link on the left side of the page.
5. Select the type of credential you want to add.
    a. SSH Username with a private key is the one I have used for the GIT.
    b. Username with password is the one I have used for docker hub credentials.





Scope: Select the scope of the credential (e.g., Global or System).

ID: Provide an identifier for the credential. If left blank, Jenkins generates one automatically.

Description: (Optional) Add a description to help identify the credential.

Private Key: (For SSH) Provide the SSH private key.

After filling in the necessary fields, click OK to save the credentials.

Jenkins Pipeline Script

The pipeline consists of several stages, including code checkout, compilation, testing, packaging, running an Ansible playbook, and deploying to Kubernetes.

Detailed breakdown

1. **Agent:**
    a. **agent any**: Specifies that the pipeline can run on any available Jenkins agent.
2. **Environment Variables:**

    environment: Defines environment variables to be used within the pipeline.

a. **JOB_NAME = "${env.JOB_NAME}"**: The name of the Jenkins job.
b. **BUILD_NUMBER = "${env.BUILD_NUMBER}"**: The build number of the current job.
c. **DOCKER_HUB = credentials('kowshik-dockerhub')**: Retrieves Docker Hub credentials stored in Jenkins with the ID 'kowshik-dockerhub'.

3. Stage: Code Checkout:

stage('Code Checkout'): Defines the code checkout stage.

a. **steps** : Contains the steps to be executed in this stage.
b. **git**: Checks out code from the specified Git repository using the provided credentials.

4. Stage: Code Compile:

- **stage('Code Compile') { ... }**: Defines the code compilation stage.
  a. **sh 'mvn compile'**: Executes the Maven compile command.

5. Stage: Test:

- **stage('Test') { ... }**: Defines the testing stage.
  a. **sh 'mvn test'**: Executes the Maven test command.

6. Stage: Build:

- **stage('Build') { ... }**: Defines the build/package stage.
  a. **sh 'mvn package'**: Executes the Maven package command.

I Have other stages like Run Ansible Playbook and Deploy k8s which I will discuss in the later parts of the document.

The post actions steps defines actions to be taken after the pipeline execution. These execute whether the pipeline failed or succeeded. This block has always, success and failure actions. These print messages like pipelines have completed their execution.

always { ... }: Actions to perform regardless of the pipeline outcome.

success { ... }: Actions to perform if the pipeline succeeds

failure { ... }: Actions to perform if the pipeline fails

```
pipeline {
    // Define the agent to be used for the pipeline. 'any' means it can run
on any available agent.
    agent any
    environment {
        // Define environment variables
        JOB_NAME = "${env.JOB_NAME}"
        BUILD_NUMBER = "${env.BUILD_NUMBER}"
        DOCKER_HUB = credentials('kowshik-dockerhub')
           }

    // Define the stages of the pipeline
    stages {
        stage('Code Checkout') {
            // Stage to check out code from the repository
            steps {
                // Use SSH URL for Git checkout. Ensure the Jenkins agent
has access to the private key.
                git changelog: false, credentialsId: 'Jenkins_slave_GIT',
poll: false, url: 'git@github.com:Kowshikreddy137/IGP_Purdue.git'
            }
        }

        stage('Code Compile') {
            // Stage to compile the code
            steps {
                // Run the Maven compile command
                sh 'mvn compile'
            }
        }

        stage('Test') {
            // Stage to run tests
            steps {
                // Run the Maven test command
                sh 'mvn test'
            }
        }

        stage('Build') {
            // Stage to build/package the application
            steps {
```

```
                // Run the Maven package command
                sh 'mvn package'
            }
        }

        stage('Run Ansible Playbook') {
            steps {
                // Run the Ansible playbook
                ansiblePlaybook(
                    playbook:
'/var/lib/jenkins/workspace/Purdue_IGP/playbook.yml',
                    inventory: '/etc/ansible/hosts',
                    extras: "-e job_name=${JOB_NAME} -e
build_number=${BUILD_NUMBER} -e dockerhub_username=${DOCKER_HUB_USR} -e
dockerhub_password=${DOCKER_HUB_PSW}"
                )
            }
        }

        stage('Deploy k8s') {
            steps {
                    sh 'chmod +x
/var/lib/jenkins/workspace/Purdue_IGP/apply_resources.sh'
                    sh './apply_resources.sh ${BUILD_NUMBER}'
            }
        }
    }
    post {
        // Actions to perform regardless of the pipeline outcome
        always {
            echo 'Pipeline completed.'
        }
        // Actions to perform on pipeline success
        success {
            echo 'Pipeline succeeded.'
        }
        // Actions to perform on pipeline failure
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

Maven Installation:

Before building the pipeline I installed **Maven** on the Ec2 Machine.

```
sudo wget
http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo
-O /etc/yum.repos.d/epel-apache-maven.repo

sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo

sudo yum install -y apache-maven

mvn --version
```

Task 4: Docker Integration

Step 1: Write a Dockerfile

1. Create a Dockerfile in the project directory and push it to git.

Dockerfile contents

```
# Use the official Tomcat image as the base
FROM tomcat:9-jdk8-openjdk

# Copy the WAR file to the Tomcat webapps directory
COPY abc_tech.war /usr/local/tomcat/webapps/

# Expose the default Tomcat port
EXPOSE 8080

# Starting the tomacat server
CMD ["catalina.sh", "run"]
```

The first line specifies the base image for the Docker container. The image used is tomcat:9-jdk8-openjdk, which includes Tomcat 9 and OpenJDK 8. This means the container will come pre-configured with Tomcat 9 and the necessary Java Development Kit (JDK) to run Java applications.

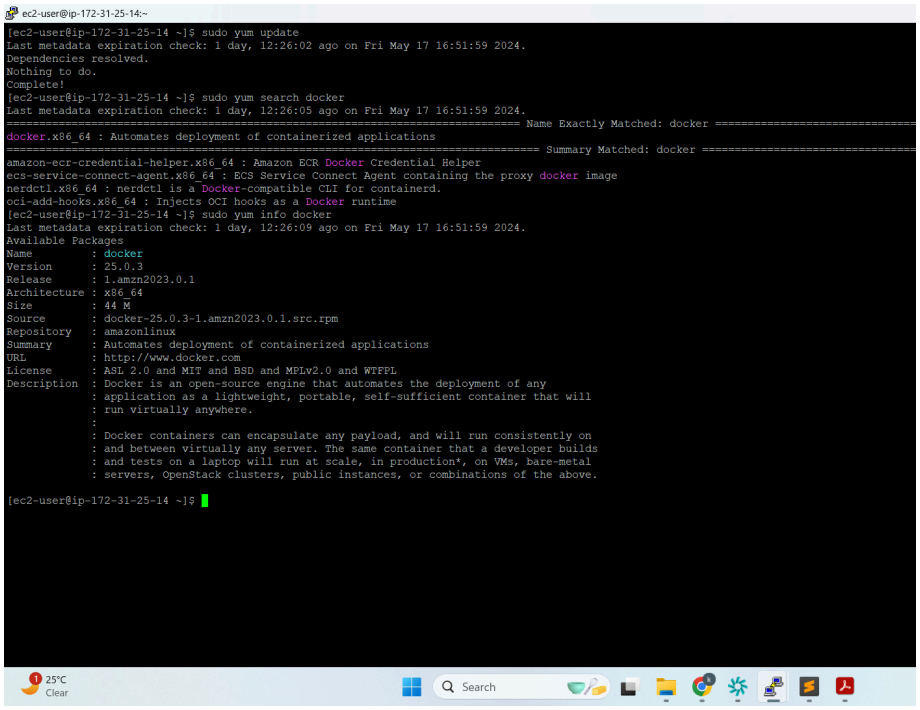
The second line copies a WAR file named abc_tech.war(which is build from the code we pushed into git) from your local machine(Jenkins workspace) to the webapps directory inside the Tomcat server in the container. The webapps directory is where Tomcat looks for applications to deploy.

The third line tells Docker to expose port 8080 on the container. Tomcat's default port for serving HTTP requests is 8080, so this line makes the port accessible from the outside, allowing usto reach the web application running inside the container.

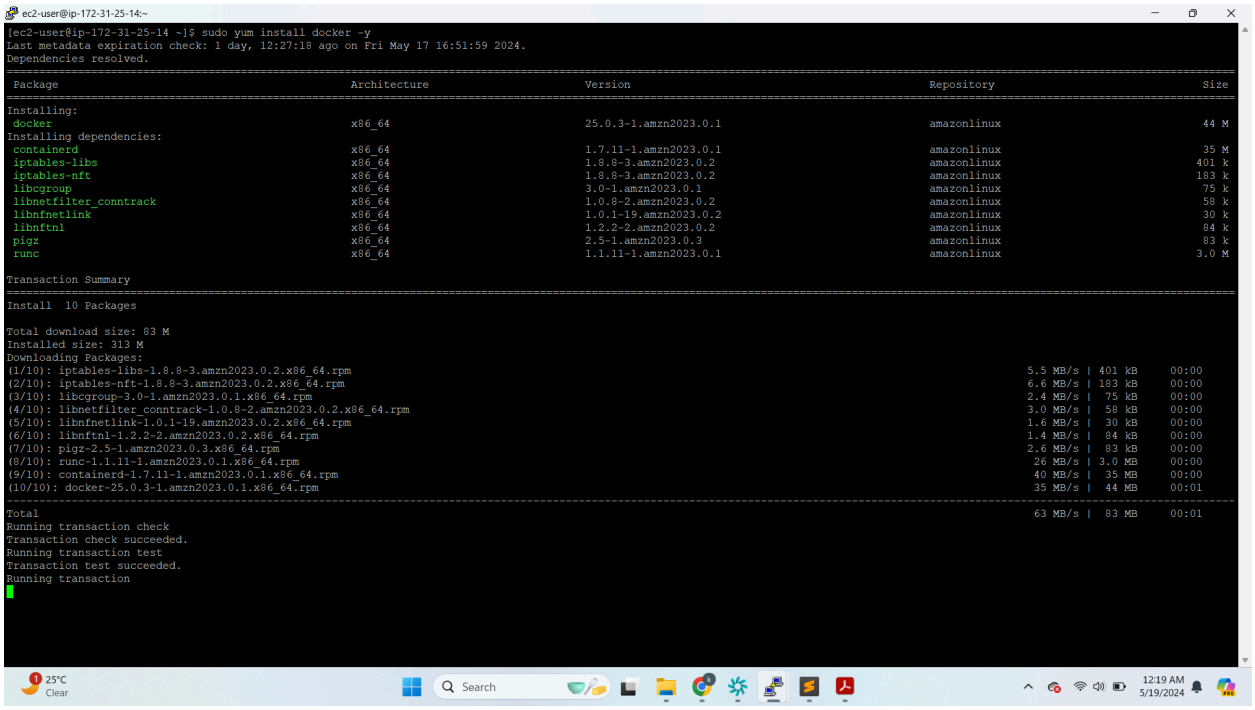
The fourth line specifies the command to run when the container starts. *catalina.sh* is a script used to control the Tomcat server, and the run argument tells it to start the Tomcat server in the foreground.

Step 2: Install docker on Linux machine

1. Start by updating your package index to ensure you have the latest information about available packages.
2. Search for docker package in the yum repository to make sure it is available.



3. Then install docker using yum command *sudo yum install docker -y*



4. Once the installation is complete, start the Docker service. And to ensure Docker starts when your system boots, enable the Docker service using these commands.

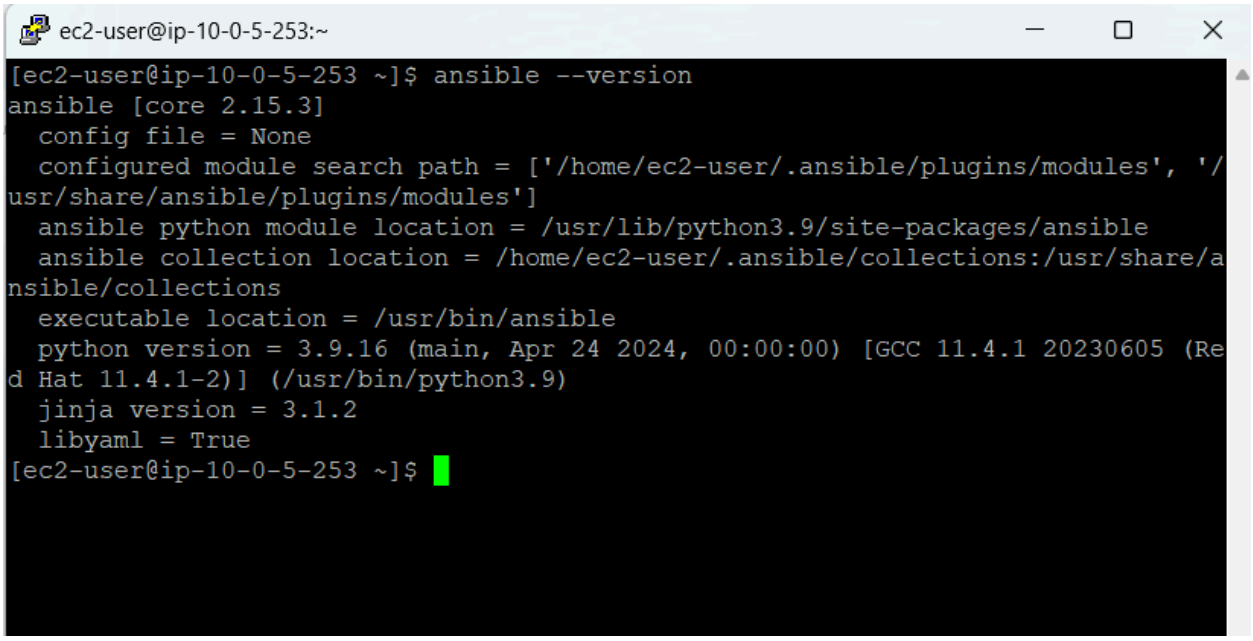
```
sudo systemctl start docker
sudo systemctl enable docker
```

5. Adding user jenkins to the Docker group allows you to run Docker commands.
6. Restart the Jenkins service to apply the group changes.

```
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
groups jenkins
```

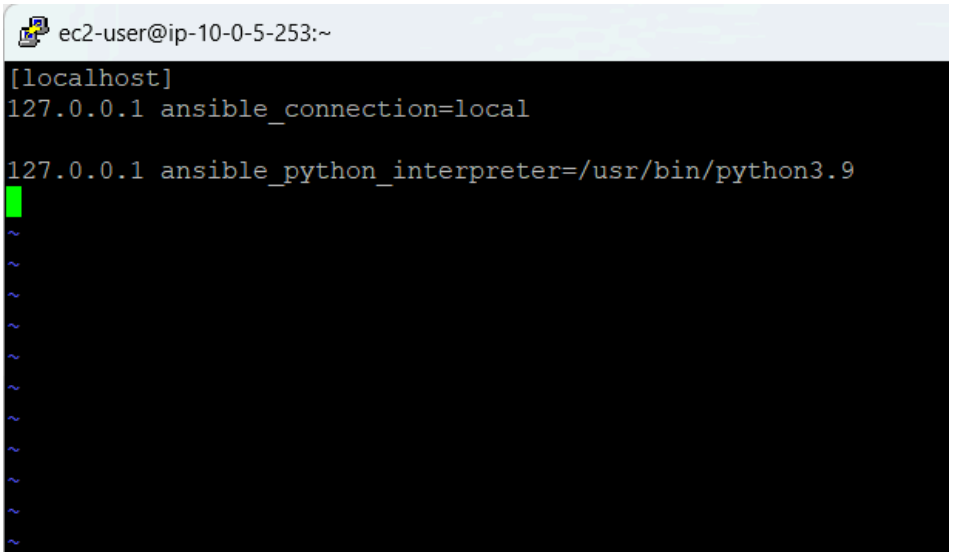
Step 3: Install ansible on ec2 machine

Install ansible using yum repo and Check the installed version of Ansible to confirm it is correctly installed.

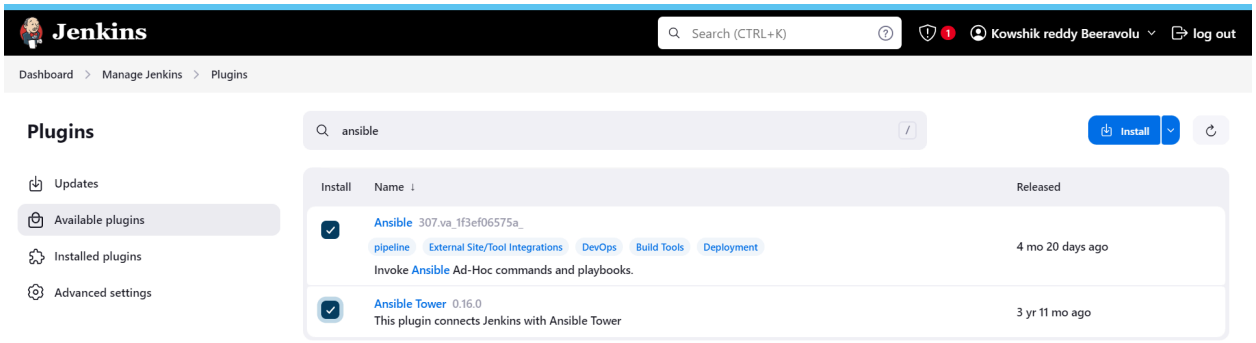


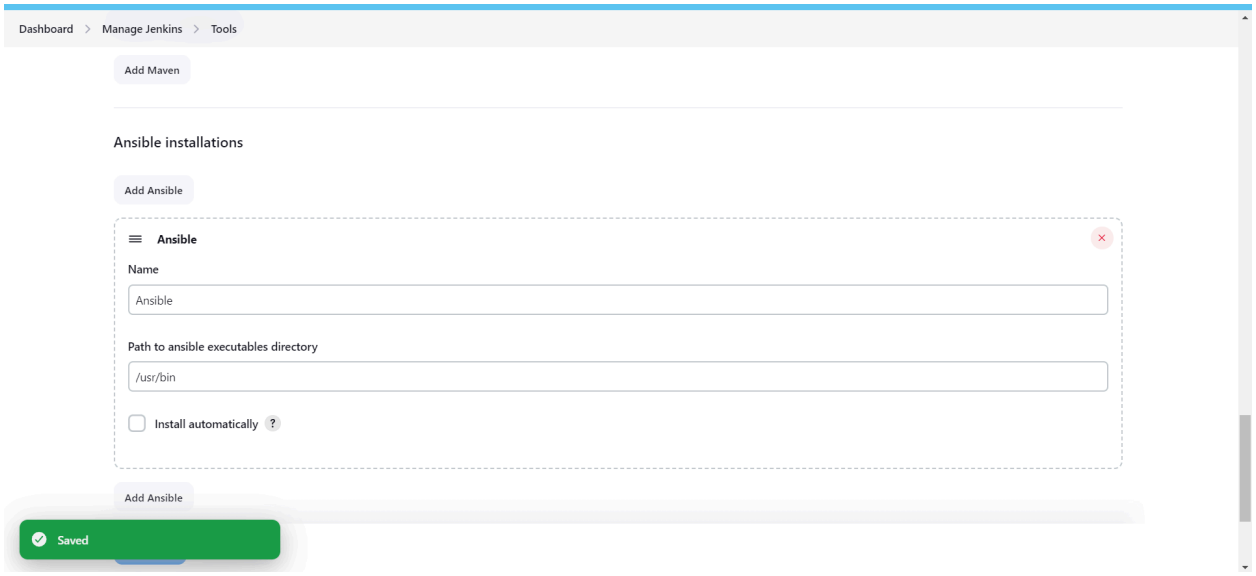
The Ansible hosts file, also known as the inventory file, is where you define the systems that Ansible will manage. Generally we can find the hosts file in /etc/ansible/hosts or we can give a different inventory file while running a playbook.

Below is the ansible hosts file for running the docker playbook.



Step 4: Install Ansible plugin and configure ansible installaton





Step 5: Ansible playbook to Automate

I have written an ansible playbook that automates the process of building, pushing, and deploying a Docker container.

Let's breakdown the playbook:

1. playbook is named "Build, Push, and Deploy Docker Container" and it runs on the `localhost` host, meaning it will execute on the machine where Ansible is running.
2. In the vars section there are couple of placeholders for the Docker Hub username and password. They will be provided when the playbook is executed, allowing for secure login to Docker Hub.

3. The first task copies the WAR file from a specified Jenkins workspace location to the Docker build context directory. The WAR file is renamed to abc_tech.war in the destination directory.
4. The second task builds a Docker image from the Dockerfile in the specified directory. The image is tagged with **abc_tech:<build_number>**, where **<build_number>** is a dynamic variable.
5. This third task tags the newly built Docker image with a tag suitable for pushing to Docker Hub. The image is tagged as **kowshikreddy137/abc_tech:<build_number>**.
6. The Fourth task logs into Docker Hub using the provided username and password.
7. The fifth task pushes the tagged Docker image to Docker Hub, making it available for download and deployment.
8. The last task runs the Docker container in detached mode (-d) with the name **ABC_Tech_<build_number>**. It maps port 8081 on the host to port 8080 on the container, making the application accessible on port 8081.

```
---
- name: Build, Push, and Deploy Docker Container
  hosts: localhost

  vars:
    dockerhub_username: "{{ dockerhub_username }}"
    dockerhub_password: "{{ dockerhub_password }}"

  tasks:
    - name: Copy WAR file to Docker build context
      copy:
        src: /var/lib/jenkins/workspace/{{ job_name
}}/target/ABCtechnologies-1.0.war
        dest: /var/lib/jenkins/workspace/Purdue_IGP/abc_tech.war

    - name: Build Docker image
      command: docker build -t abc_tech:{{ build_number }} .
      args:
        chdir: /var/lib/jenkins/workspace/Purdue_IGP

    - name: Tag Docker image
      command: docker tag abc_tech:{{ build_number }}
kowshikreddy137/abc_tech:{{ build_number }}

    - name: Log in to Docker Hub
      docker_login:
```

```
        username: "{{ dockerhub_username }}"
        password: "{{ dockerhub_password }}"
        registry_url: "https://index.docker.io/v1/"

    - name: Push Docker image to Docker Hub
      command: docker push kowshikreddy137/abc_tech:{{ build_number }}

    - name: Run Docker container
      command: docker run -d --name ABC_Tech_{{ build_number }} -p
8081:8080 kowshikreddy137/abc_tech:{{ build_number }}
```

This snippet is a part of a Jenkins pipeline script which is set up to run an Ansible playbook.

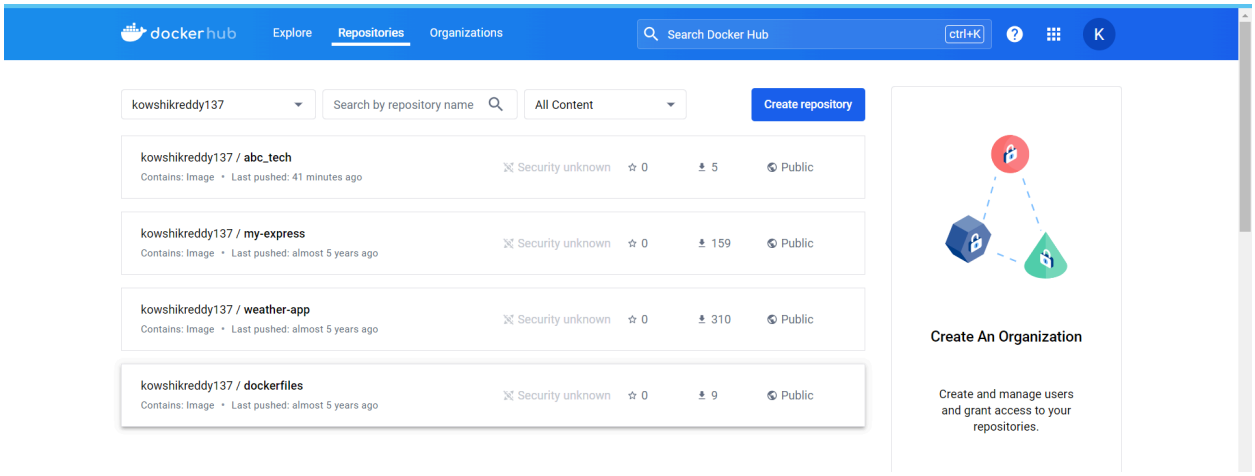
```
 stage('Run Ansible Playbook') {
            steps {
                // Run the Ansible playbook
                ansiblePlaybook(
                    playbook:
'/var/lib/jenkins/workspace/Purdue_IGP/playbook.yml',
                    inventory: '/etc/ansible/hosts',
                    extras: "-e job_name=${JOB_NAME} -e
build_number=${BUILD_NUMBER} -e dockerhub_username=${DOCKER_HUB_USR} -e
dockerhub_password=${DOCKER_HUB_PSW}"
                )
            }
        }
```

playbook: The path to the Ansible playbook that should be run. Here, the playbook is located at /var/lib/jenkins/workspace/Purdue_IGP/playbook.yml.
inventory: The path to the Ansible inventory file. This file lists the hosts that Ansible can manage. Here, the inventory file is located at /etc/ansible/hosts.
extras: Additional command-line options for the ansible-playbook command. This includes extra variables passed to the playbook using the -e option. These variables are dynamically set using Jenkins environment variables:

- job_name=${JOB_NAME}: Passes the name of the Jenkins job to the playbook.
- build_number=${BUILD_NUMBER}: Passes the current build number to the playbook.
- dockerhub_username=${DOCKER_HUB_USR}: Passes the Docker Hub username to the playbook.

- dockerhub_password=${DOCKER_HUB_PSW}: Passes the Docker Hub password to the playbook.

We need to have a docker hub account to able to push the docker image into the repo and use it. Below is a screenshot of my account and the abc_tech image repository.



Task 5: Kubernetes Integration

I have utilized Terraform to provision an Amazon EKS (Elastic Kubernetes Service) cluster on AWS. This setup is designed to deploy and manage our application efficiently within a scalable and robust Kubernetes environment. By leveraging Terraform, the infrastructure is defined as code, ensuring consistency and repeatability in the deployment process. The EKS cluster provides a managed Kubernetes service, offering simplified operations, automated scaling, and integrated AWS services, which enhance the overall performance and reliability of our application

This brings me to install Terraform on my local machine, and here is how I have done it.

Download Terraform:

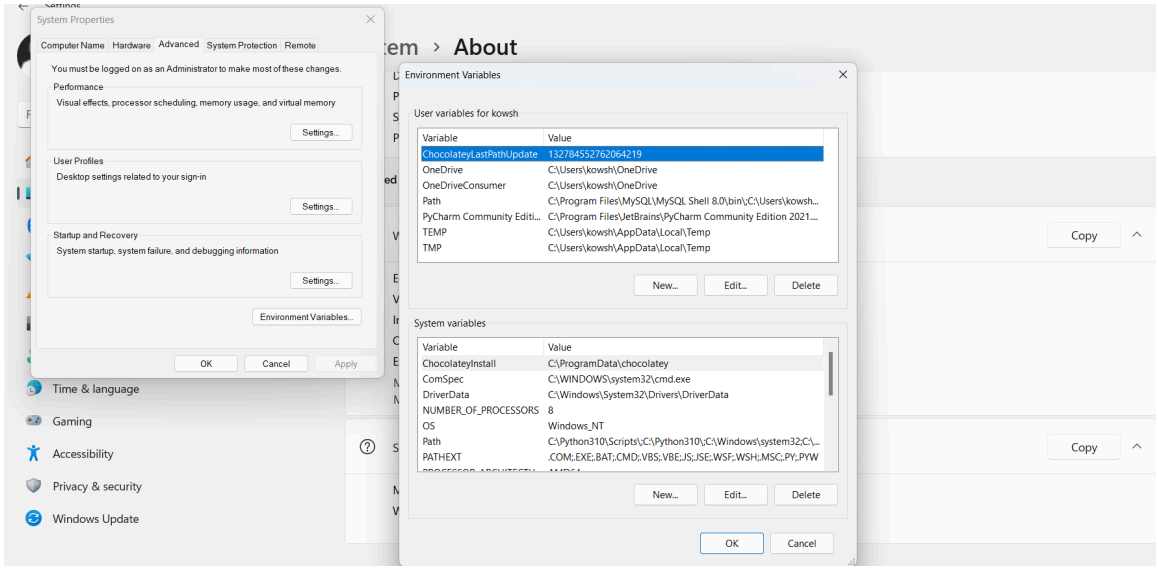
- Visit the Terraform download page.
- Select the appropriate Windows version and download the ZIP file.

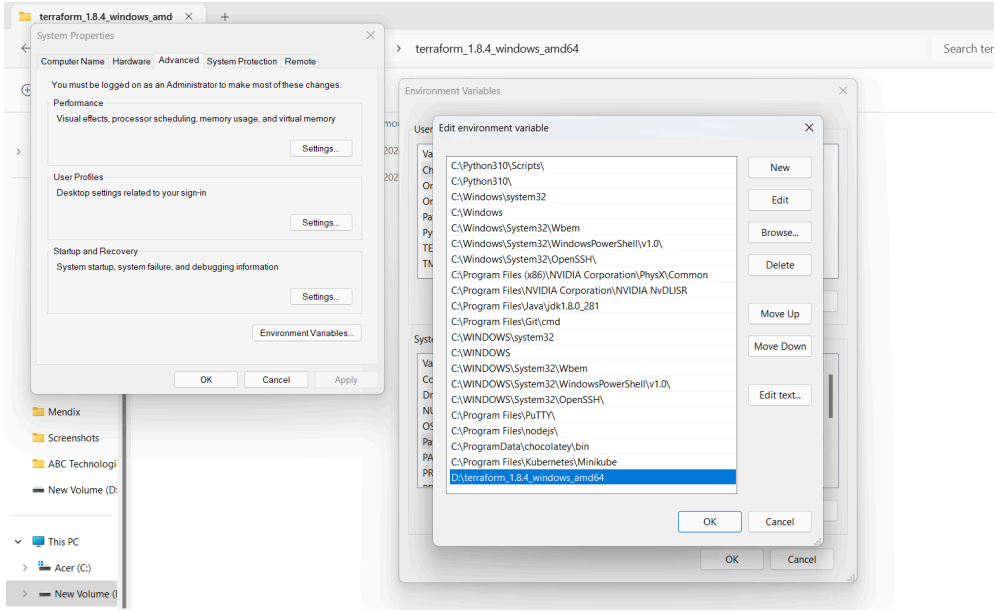
Extract the ZIP File:

- Right-click on the downloaded ZIP file and select "Extract All..." to extract the files

Add Terraform to PATH:

- Open the Start Menu, search for "Environment Variables", and select "Edit the system environment variables".
- In the System Properties window, click on the "Environment Variables..." button.
- In the Environment Variables window, under "System variables", find and select the `Path` variable, then click "Edit".
- In the Edit Environment Variable window, click "New" and add the path to the directory where you extracted Terraform.





We can verify the terraform installation by opening the command prompt and Type terraform -v and press Enter. Weshould see the version of Terraform that we installed.

Now we can start writing terraform file and execute them to provision our infrastructure (eks cluster)

This is how i have achieved this:

Create a directory for your Terraform configuration files. Within this directory, create the following files:

1. versions.tf
2. Variables.tf
3. vpc.tf
4. Security_groups.tf
5. eks_cluster.tf
6. outputs.tf

Versions.tf

```
terraform {
  required_version = ">= 0.12"
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "~> 3.1.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = ">=2.7.1"
    }
    aws = {
      source  = "hashicorp/aws"
      version = ">= 3.68.0"
    }
```

```
  }
}
```

terraform block: This defines settings related to Terraform itself.

required_version = ">= 0.12": Specifies that the Terraform version must be 0.12 or higher.

The required_providers block specifies the providers that Terraform will use, along with their source and version constraints.

kubernetes provider: Manages Kubernetes resources.

version: Must be version 2.7.1 or higher.

aws provider: Manages AWS resources.

version: Must be version 3.68.0 or higher.

Variables.tf

```
variable "kubernetes_version" {
  default     = 1.29
  description = "kubernetes version"
}

variable "vpc_cidr" {
  default     = "10.0.0.0/16"
  description = "default CIDR range of the VPC"
}
variable "aws_region" {
  default = "us-east-1"
  description = "aws region"
}
```

- **kubernetes_version**: A variable to specify the version of Kubernetes, with a default value of 1.29.
- **vpc_cidr**: A variable to specify the CIDR range for an AWS VPC, with a default value of "10.0.0.0/16".
- **aws_region**: A variable to specify the AWS region for resource deployment, with a default value of "us-east-1".

These variables allow you to customize the configuration of your Terraform setup without directly modifying the code. By changing the input values, you can adjust the Kubernetes version, the CIDR block for the VPC, and the AWS region as needed.

vpc.tf

```
provider "aws" {
  region = var.aws_region
}

data "aws_availability_zones" "available" {}

locals {
  cluster_name = "Kowshik-eks-${random_string.suffix.result}"
}

resource "random_string" "suffix" {
  length  = 8
  special = false
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.7.0"

  name                 = "Kowshik-eks-vpc"
  cidr                 = var.vpc_cidr
  azs                  = data.aws_availability_zones.available.names
  private_subnets      = ["10.0.1.0/24", "10.0.2.0/24"]
  public_subnets       = ["10.0.4.0/24", "10.0.5.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }

  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
```

```
    "kubernetes.io/role/elb"                      = "1"
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = "1"
  }
}
```

provider "aws": Configures the AWS provider.

region = var.aws_region: Sets the AWS region using the aws_region variable declared earlier. This determines the region where the resources will be created.

data "aws_availability_zones" "available": Fetches the list of available AWS availability zones in the specified region. This data source can be used to distribute resources across multiple availability zones for high availability.

locals block: Defines local variables.

cluster_name: Constructs a unique cluster name by appending a random string suffix to "Kowshik-eks-".

resource "random_string" "suffix": Generates a random string of 8 characters without special characters. This is used to ensure unique naming for resources, avoiding naming conflicts.

module "vpc": Uses a publicly available VPC module from the Terraform registry (terraform-aws-modules/vpc/aws), version 5.7.0.

name = "Kowshik-eks-vpc": Names the VPC.

cidr = var.vpc_cidr: Sets the CIDR block for the VPC using the vpc_cidr variable.

azs = data.aws_availability_zones.available.names: Uses the availability zone names fetched by the aws_availability_zones data source.

private_subnets and public_subnets: Specifies the CIDR blocks for private and public subnets, respectively.

enable_nat_gateway = true and single_nat_gateway = true: Enables and configures a single NAT gateway for outbound internet access from private subnets.

enable_dns_hostnames and enable_dns_support: Enables DNS support and hostnames within the VPC.

tags: Adds tags to the VPC, marking it as shared by the Kubernetes cluster.

public_subnet_tags and private_subnet_tags: Tags public and private subnets respectively for Kubernetes to recognize their roles. Public subnets are tagged for load balancers (elb), and private subnets for internal load balancers (internal-elb).

Security_groups.tf

```
resource "aws_security_group" "all_worker_mgmt" {
  name_prefix = "all_worker_management"
  vpc_id      = module.vpc.vpc_id
}

resource "aws_security_group_rule" "all_worker_mgmt_ingress" {
  description       = "allow inbound traffic from eks"
  from_port         = 0
  protocol          = "-1"
  to_port           = 0
  security_group_id = aws_security_group.all_worker_mgmt.id
  type              = "ingress"
  cidr_blocks = [
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16",
  ]
}

resource "aws_security_group_rule" "all_worker_mgmt_egress" {
  description       = "allow outbound traffic to anywhere"
  from_port         = 0
  protocol          = "-1"
  security_group_id = aws_security_group.all_worker_mgmt.id
  to_port           = 0
  type              = "egress"
  cidr_blocks       = ["0.0.0.0/0"]
}
```

resource "aws_security_group" "all_worker_mgmt": Creates a new security group named all_worker_mgmt.

name_prefix = "all_worker_management": Sets the prefix for the name of the security group, which will have an additional suffix automatically added by AWS to ensure uniqueness.

vpc_id = module.vpc.vpc_id: Associates the security group with the VPC created by the vpc module. This links the security group to the correct VPC.

resource "aws_security_group_rule" "all_worker_mgmt_ingress": Defines an ingress rule for the all_worker_mgmt security group.

description = "allow inbound traffic from eks": Provides a description for the rule.

from_port = 0, to_port = 0: Specifies the port range. A value of 0 indicates that all ports are included.

protocol = "-1": Allows all protocols.

security_group_id = aws_security_group.all_worker_mgmt.id: Associates this rule with the all_worker_mgmt security group.

type = "ingress": Specifies that this is an ingress rule, allowing inbound traffic.

cidr_blocks: Specifies the CIDR blocks from which inbound traffic is allowed. In this case, traffic is allowed from the private IP ranges:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

resource "aws_security_group_rule" "all_worker_mgmt_egress": Defines an egress rule for the all_worker_mgmt security group.

description = "allow outbound traffic to anywhere": Provides a description for the rule.

from_port = 0, to_port = 0: Specifies the port range. A value of 0 indicates that all ports are included.

protocol = "-1": Allows all protocols.

security_group_id = aws_security_group.all_worker_mgmt.id: Associates this rule with the all_worker_mgmt security group.

type = "egress": Specifies that this is an egress rule, allowing outbound traffic.

cidr_blocks = ["0.0.0.0/0"]: Allows outbound traffic to any IP address (0.0.0.0/0).

- **aws_security_group**: Defines a security group named all_worker_mgmt within the VPC.

- **aws_security_group_rule (ingress)**: Allows inbound traffic from private IP ranges, covering typical internal network ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16). This is useful for communication within the cluster or other private networks.
- **aws_security_group_rule (egress)**: Allows all outbound traffic to any destination. This ensures that the worker nodes can communicate outwards to the internet or other services.

These configurations ensure that the EKS worker nodes have the necessary network permissions to function correctly within the VPC, with secure inbound and unrestricted outbound traffic.

Eks_cluster.tf

```
module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  version         = "20.8.4"
  cluster_name    = local.cluster_name
  cluster_version = var.kubernetes_version
  subnet_ids      = module.vpc.private_subnets

  enable_irsa = true

  tags = {
    cluster = "demo"
  }

  vpc_id = module.vpc.vpc_id

  eks_managed_node_group_defaults = {
    ami_type               = "AL2_x86_64"
    instance_types         = ["t3.medium"]
    vpc_security_group_ids = [aws_security_group.all_worker_mgmt.id]
  }

  eks_managed_node_groups = {
    node_group = {
      min_size     = 2
      max_size     = 6
      desired_size = 2
    }
  }
}
```

Source and Version

- **source = "terraform-aws-modules/eks/aws"**: Specifies the source of the EKS module, which is a publicly available module from the Terraform Registry.
- **version = "20.8.4"**: Specifies the version of the module to use, ensuring consistency and compatibility.

Basic Cluster Configuration

- **cluster_name = local.cluster_name**: Sets the name of the EKS cluster using the local variable cluster_name defined earlier, which includes a unique random suffix.
- **cluster_version = var.kubernetes_version**: Specifies the Kubernetes version for the cluster, using the kubernetes_version variable defined earlier.
- **subnet_ids = module.vpc.private_subnets**: Specifies the subnets where the EKS cluster nodes will be deployed, using the private subnets from the VPC module.

Enabling IRSA

- **enable_irsa = true**: Enables IAM Roles for Service Accounts (IRSA), which allows Kubernetes service accounts to assume IAM roles.

Tags

- **tags**: Adds a tag to the cluster with a key cluster and value demo.

VPC Configuration

- **vpc_id = module.vpc.vpc_id**: Associates the EKS cluster with the VPC created by the VPC module.

Managed Node Group Defaults

- **eks_managed_node_group_defaults**: Sets default configuration for EKS managed node groups.
    - **ami_type = "AL2_x86_64"**: Specifies the Amazon Machine Image (AMI) type, in this case, Amazon Linux 2 for x86_64 architecture.
    - **instance_types = ["t3.medium"]**: Specifies the EC2 instance types for the nodes, here t3.medium.
    - **vpc_security_group_ids = [aws_security_group.all_worker_mgmt.id]**: Associates the all_worker_mgmt security group with the node groups for network configuration.

Managed Node Groups

- **eks_managed_node_groups**: Defines specific managed node groups for the EKS cluster.
    - **node_group**: Defines a node group with the following settings:
        - **min_size = 2**: Minimum number of nodes in the group.
        - **max_size = 6**: Maximum number of nodes in the group.
        - **desired_size = 2**: Desired number of nodes in the group.

Outputs.tf

```
output "cluster_id" {
  description = "EKS cluster ID."
  value       = module.eks.cluster_id
}

output "cluster_endpoint" {
  description = "Endpoint for EKS control plane."
  value       = module.eks.cluster_endpoint
}

output "cluster_security_group_id" {
  description = "Security group ids attached to the cluster control plane."
  value       = module.eks.cluster_security_group_id
}

output "region" {
  description = "AWS region"
  value       = var.aws_region
}

output "oidc_provider_arn" {
  value = module.eks.oidc_provider_arn
}
}
```

Cluster ID Output

description = "EKS cluster ID.": Provides a description for the output.

value = module.eks.cluster_id: Specifies the value of the output, which is the ID of the EKS cluster obtained from the eks module.

Cluster Endpoint Output

- **description = "Endpoint for EKS control plane."**: Provides a description for the output.
- **value = module.eks.cluster_endpoint**: Specifies the value of the output, which is the endpoint for the EKS control plane obtained from the eks module.

Cluster Security Group ID Output

description = "Security group ids attached to the cluster control plane.": Provides a description for the output.

value = module.eks.cluster_security_group_id: Specifies the value of the output, which is the security group IDs attached to the EKS cluster control plane obtained from the eks module.

AWS Region Output

- **description = "AWS region": Provides a description for the output.**
- **value = var.aws_region: Specifies the value of the output, which is the AWS region obtained from the aws_region variable.**

OIDC Provider ARN Output

value = module.eks.oidc_provider_arn: Specifies the value of the output, which is the OIDC provider ARN obtained from the eks module.

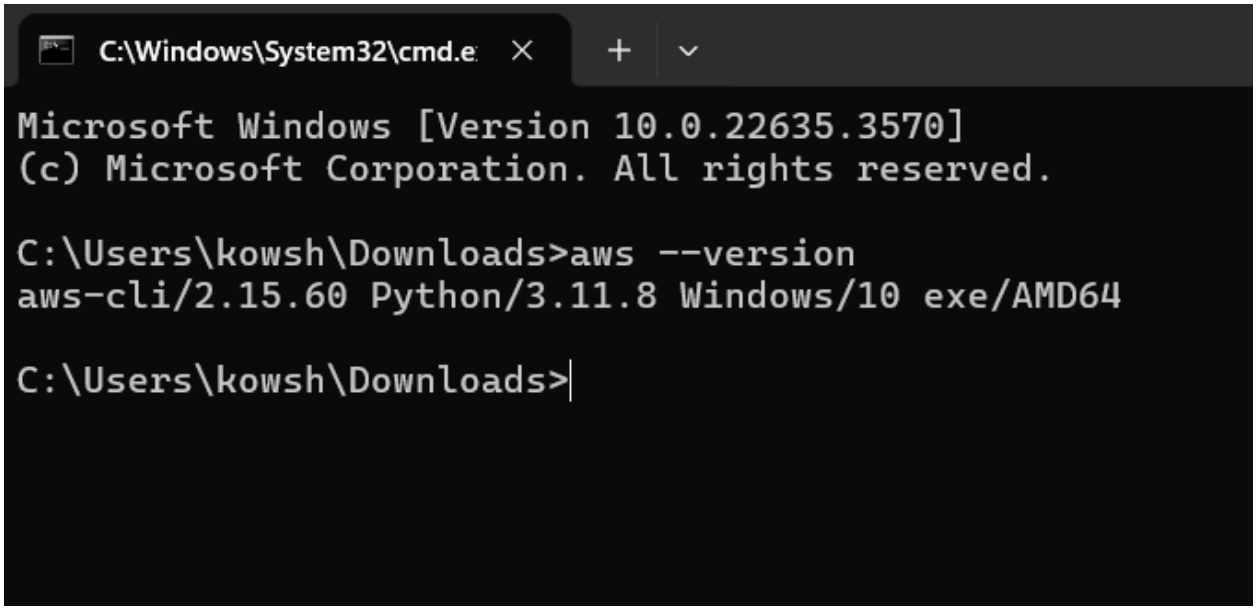
Summary

These files orchestrate the deployment of an EKS (Elastic Kubernetes Service) cluster within an AWS VPC (Virtual Private Cloud). It sets up necessary networking components like security groups and subnets, as well as provisions the EKS cluster itself, specifying its configuration, managed node groups, and associated security groups. Outputs are defined to retrieve essential information post-deployment. This setup creates a scalable and secure Kubernetes environment ready for application deployment and management.

Now we have come to a part where we can run these files and provision the required infrastructure. To do that we have to install AWS CLI and configure it. This will make sure terraform creates resources on the required account.

After the installation is complete, you can verify it by checking the installed version of the AWS CLI:

```
aws --version
```



Then we have to run `aws configure`

Then it will promp to enter your AWS Access Key ID, Secret Access Key, default region, and default output format.

- **Access Key ID** and **Secret Access Key**: These are the credentials for your AWS IAM user. Enter the Access Key ID and Secret Access Key generated when you created the IAM user.
- **Default region**: This is the AWS region where you want the AWS CLI to operate by default. Enter the code for your preferred AWS region (e.g., `us-west-2`).
- **Default output format**: This is the default format for command output. You can choose from `json`, `text`, or `table`. JSON is a common choice for scripting. You can press Enter to accept the default (`json`) or type your preferred output format.

Then we can start running the terraform commands, we have to start by initiating the folder where all the files are located.

```
terraform init
```

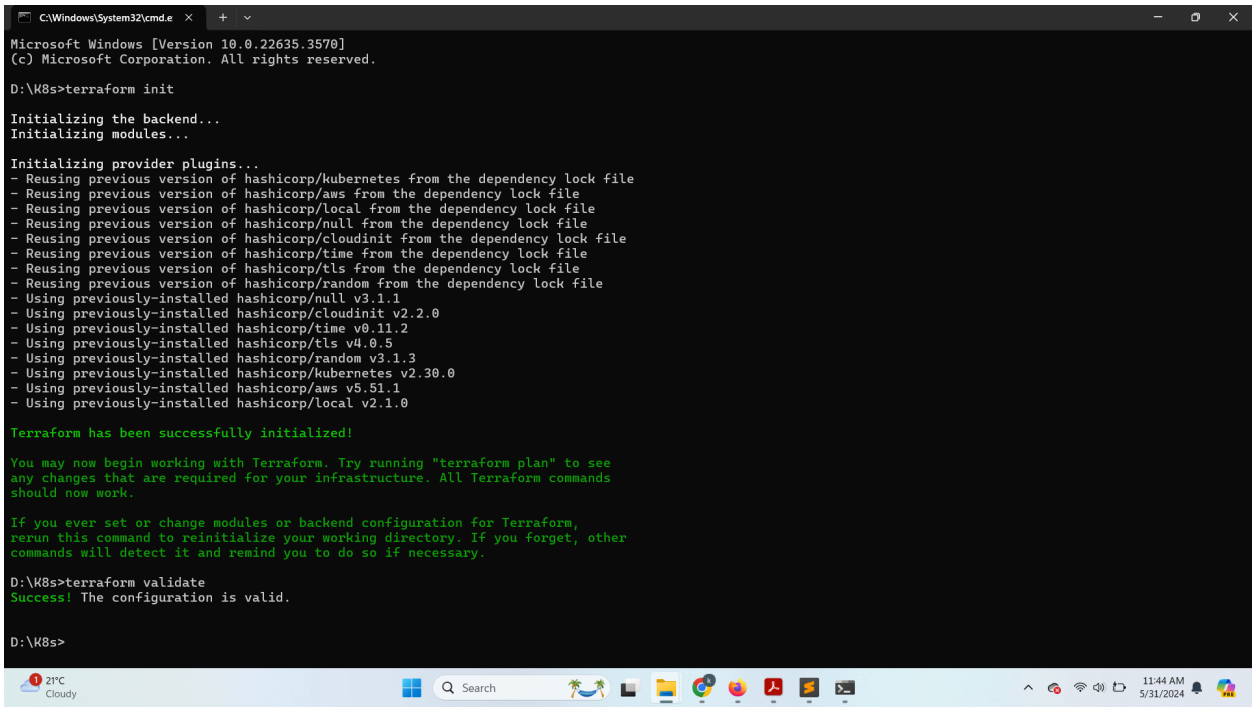
This command initializes the directory and downloads any necessary providers.

```
terraform validate
```

Terraform will validate the syntax and configuration of your `.tf` files.

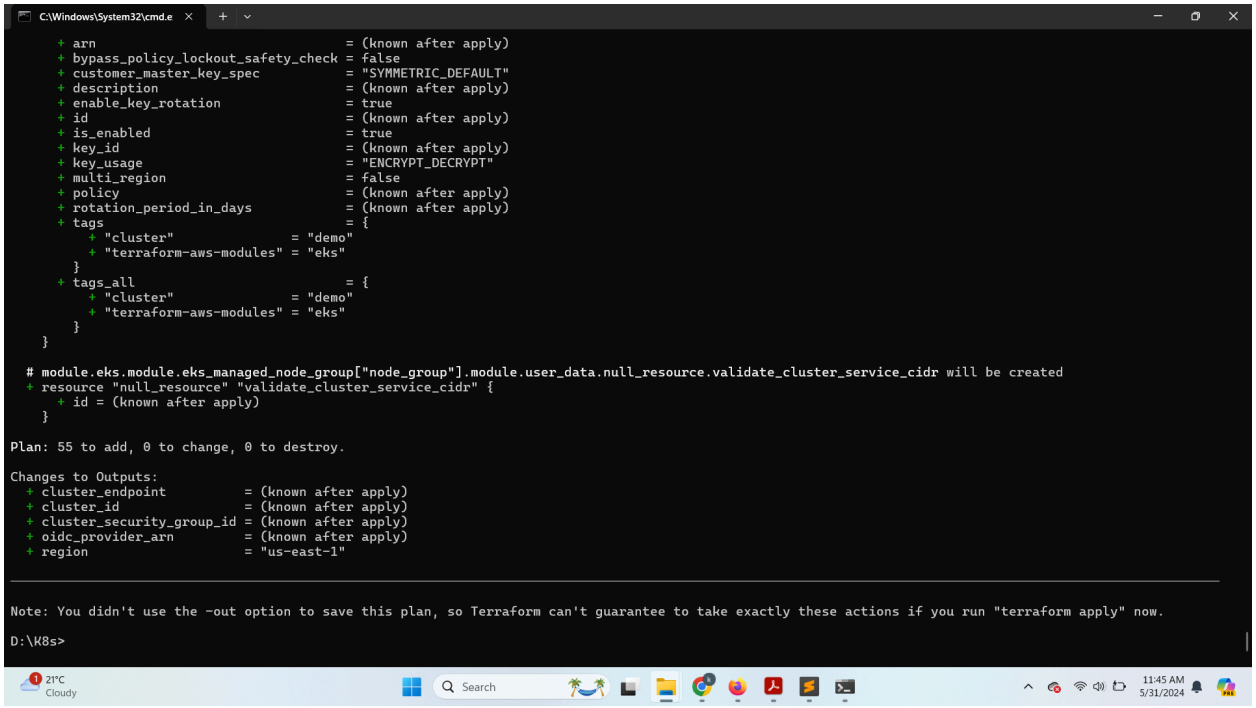
If there are any syntax errors or other issues, Terraform will display error messages indicating what needs to be corrected.

If your configuration is valid, Terraform will output a message indicating that the configuration is valid.



```
terraform plan
```

With this command, Terraform will analyze your configuration files, query the current state of your infrastructure, and then generate an execution plan based on any changes required to achieve the desired state described in your configuration files.

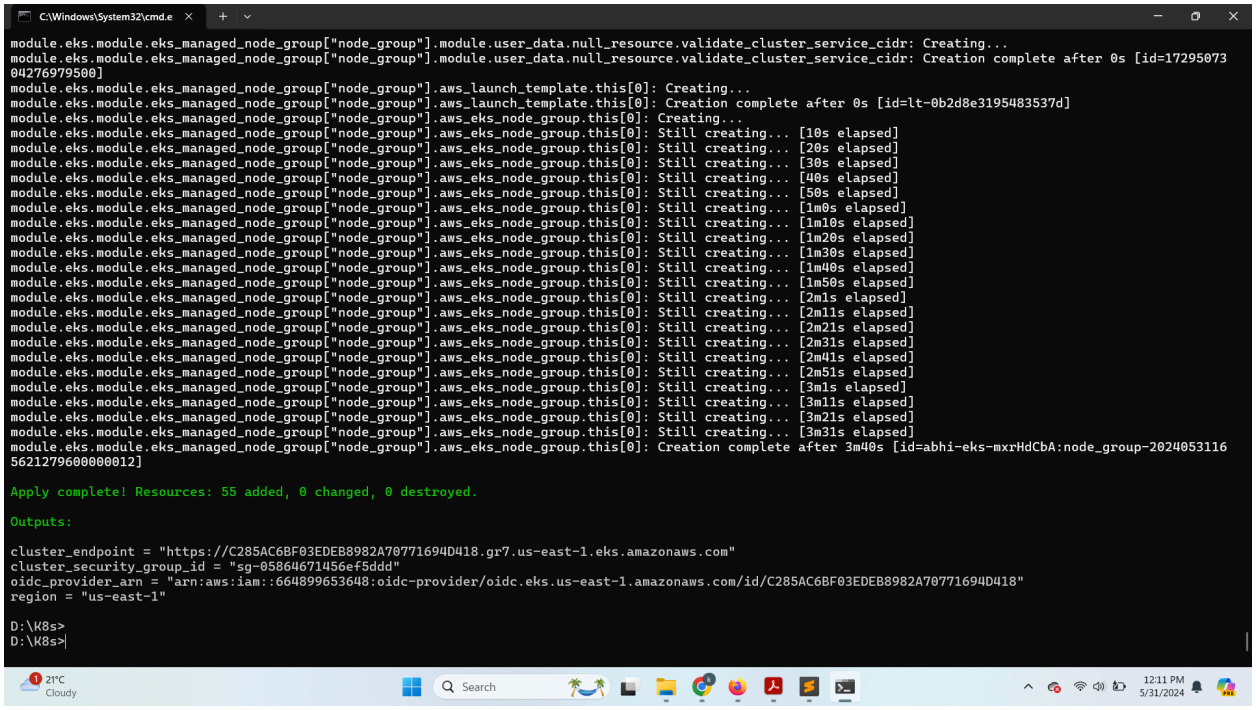



```
terraform apply
```

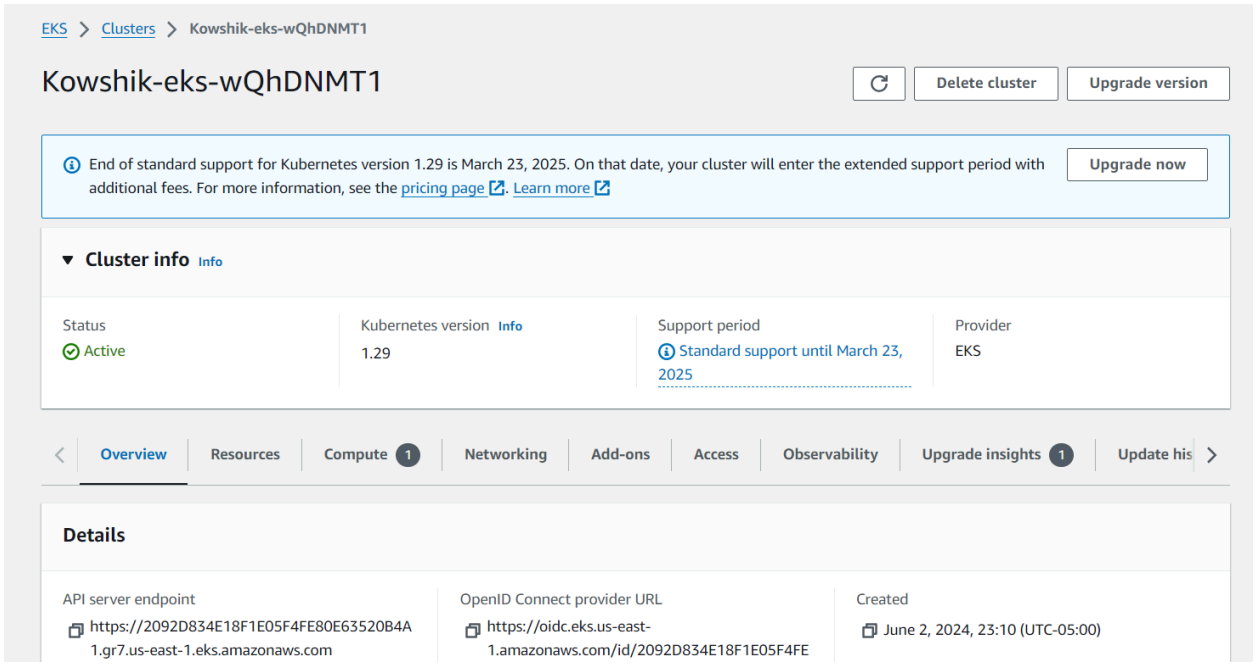
Terraform will prompt you to confirm that you want to apply the changes described in the execution plan. Review the changes carefully before confirming. You can type yes and press Enter to proceed.

Terraform will then begin applying the changes to your infrastructure. During this process, it will create, modify, or destroy resources as necessary to achieve the desired state described in your configuration files.

Once the changes have been applied, Terraform will output a summary of the actions taken and any relevant information about the state of your infrastructure.



Here are the resources created by the Terraform files



Now all i had to do was install kubectl on my linux machine and connect the eks cluster. So that I can access the eks cluster.

Kubectl Installation:

Use curl to download the **kubectl** binary.

Make the **kubectl** binary executable and move it to a directory included in your PATH.

Check that **kubectl** is installed and its version.

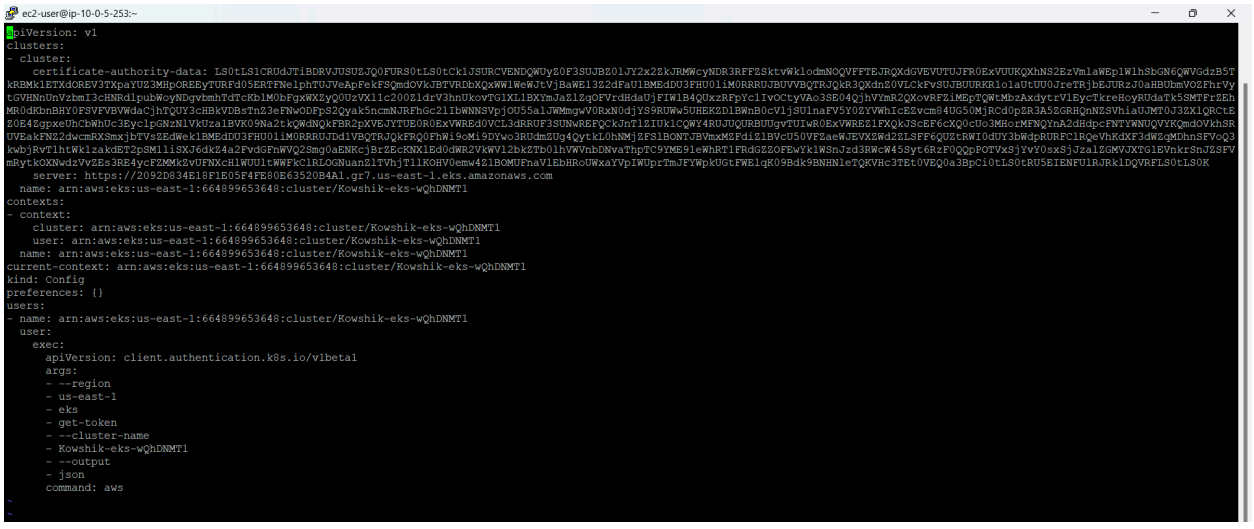
```
curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
    chmod +x ./kubectl
    sudo mv ./kubectl /usr/local/bin/kubectl
```

These commands will install kubectl and configure it.

Now we have to register the created cluster to able to access it.

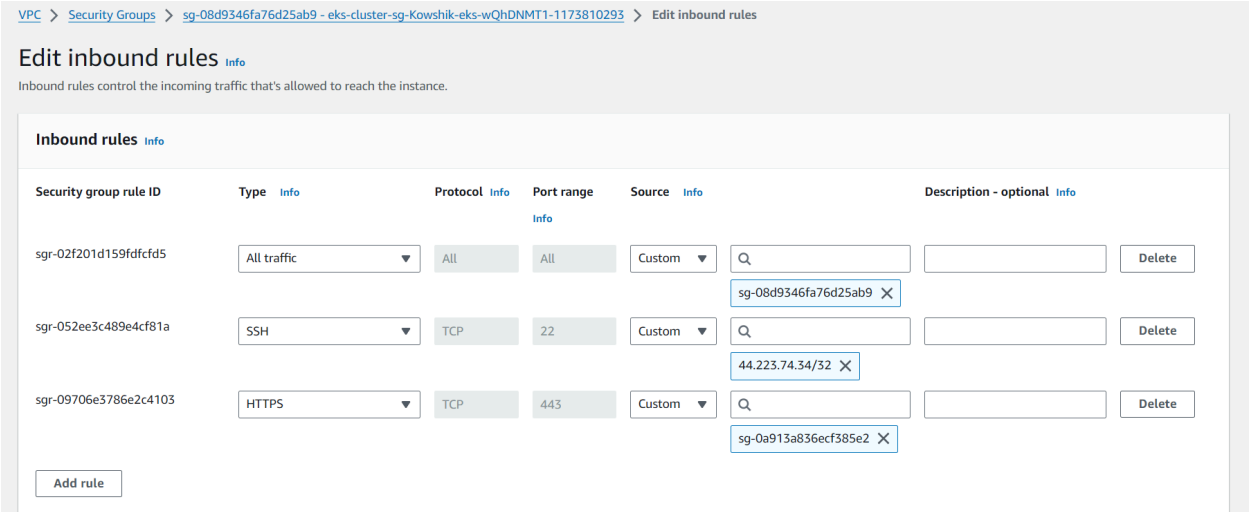
```
aws eks register-cluster     --name Kowshik-eks-wQhDNMT1
--connector-config
roleArn=arn:aws:iam::664899653648:role/your-eks-connector-role,provider="OTHER"     --region us-east-1
```

This is how my kubeconfig looks like after registering it.

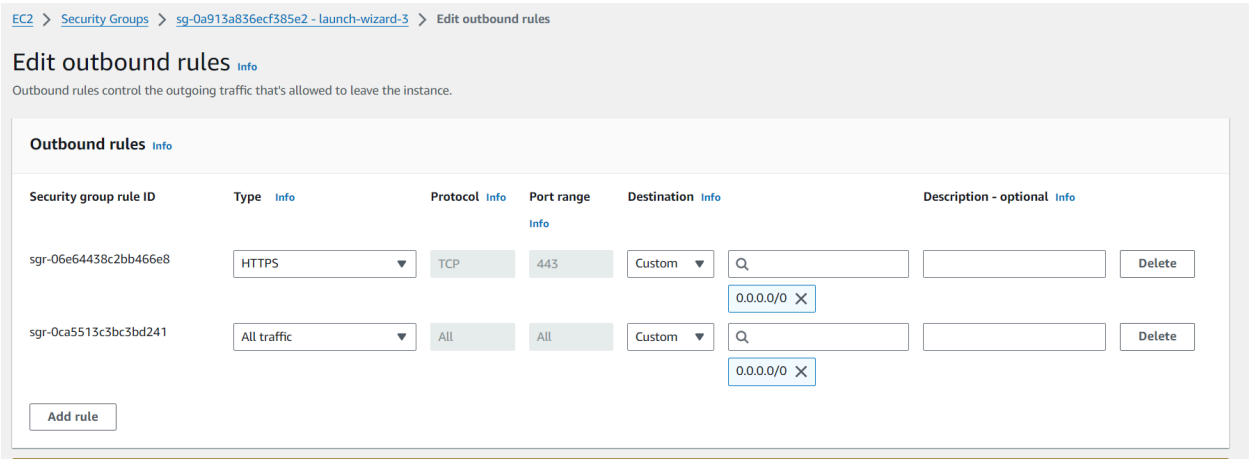


- Before being able to connect, we have to make some little adjustments to security groups, so that we can access the eks cluster.

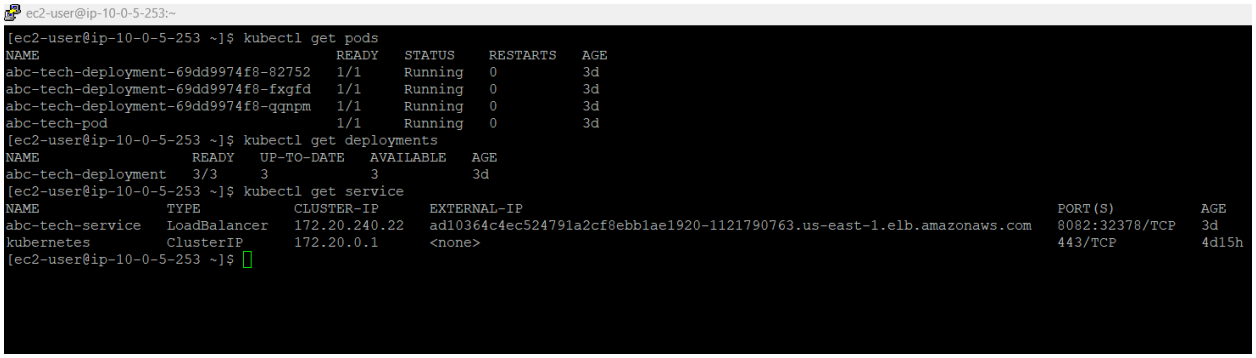
- First we have to add the security group of our ec2 machine in the inbound rules of the eks cluster security group to all all traffic from the ec2 machine (linux machine.



- Now we have to edit the outbound rules of the ec2 machine so that it can access the ec2-cluster.



- Now we will be able to run kubectl commands on the eks cluster we have created. This will make our job easier because we can create deployments, pods, services, check logs, and other things on the pods.
- Below is an screenshot of running those commands.



Now we reach to the part where we automate the process from jenkins pipeline.

```
  stage('Deploy k8s') {
            steps {
                    sh 'chmod +x
/var/lib/jenkins/workspace/Purdue_IGP/apply_resources.sh'
                    sh './apply_resources.sh ${BUILD_NUMBER}'
            }
        }
```

1. **stage('Deploy k8s'):** This defines a stage in your Jenkins pipeline named "Deploy k8s". Stages are used to organize and visualize the different steps in your pipeline.
2. **steps**: Within the stage, you define the steps that Jenkins will execute.
3. sh **'chmod +x /var/lib/jenkins/workspace/Purdue_IGP/apply_resources.sh**': This command grants execute permissions (chmod +x) to the shell script apply_resources.sh located in the specified workspace directory.
4. sh '**./apply_resources.sh ${BUILD_NUMBER}**': This command executes the shell script apply_resources.sh, passing the Jenkins build number (BUILD_NUMBER) as an argument. The build number can be used within your shell script for versioning or tracking purposes.

Kubernetes Implementation

Now i would like to covers the creation and configuration of a Deployment, Service, and Pod for the ABC Tech application.

Deployment.yml

This YAML file defines a Kubernetes Deployment named `abc-tech-deployment` that manages three replicas of a Pod. Each Pod runs a single container based on the `kowshikreddy137/abc_tech:49` Docker image, and the container exposes port `8080`. The Deployment ensures that there are always three replicas of the Pod running and can manage updates and scaling of the application.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: abc-tech-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: abc-tech
  template:
    metadata:
      labels:
        app: abc-tech
    spec:
      containers:
      - name: abc-tech-container
        image: kowshikreddy137/abc_tech:49
        ports:
        - containerPort: 8080
```

apiVersion: apps/v1:

- Specifies the API version of the Kubernetes object you're using. apps/v1 is the current stable version for Deployments.

kind: Deployment:

- Specifies the type of Kubernetes resource. In this case, it is a Deployment, which is used to manage a set of replica Pods for scaling and updating applications.

metadata:

- **name: abc-tech-deployment**: Sets the name of the Deployment to abc-tech-deployment.

Spec: Defines the desired state of the Deployment.

- **replicas: 3**: Specifies the number of Pod replicas to run. In this case, 3 Pods will be created and maintained.
- **Selector**: Defines how to identify the Pods that belong to this Deployment.
    - **matchLabels**:
        - **app: abc-tech**:
            - The Deployment will manage Pods that have the label app: abc-tech.
- **Template**: Defines the Pod template used to create new Pods. It includes the desired specifications for the Pods managed by the Deployment.
    - **metadata**:
        - **labels**:
            - **app: abc-tech**: Labels assigned to the Pods created by this template. These labels should match the selector labels to ensure that the Deployment can identify and manage its Pods.
    - **Spec**: Defines the containers that will be part of each Pod.
        - **containers**:
            - A list of containers to be included in each Pod.
            - **- name: abc-tech-container**: The name of the container.
            - **image: kowshikreddy137/abc_tech:49**: Specifies the Docker image to use for the container. In this case, it is kowshikreddy137/abc_tech:49, where kowshikreddy137 is the Docker Hub username, abc_tech is the repository name, and 49 is the tag/version of the image.

- **Ports**: Defines the ports that will be exposed by the container.
  - **containerPort: 8080**: Exposes port 8080 on the container.

Pods.yml

This YAML file defines a Kubernetes Pod named `abc-tech-pod` that runs a single container based on the `kowshikreddy137/abc_tech:49` Docker image. The container will expose port `8080`. This Pod is a standalone instance and is typically used for running single instances of an application.

apiVersion: v1

```
kind: Pod
metadata:
  name: abc-tech-pod
  annotations:
spec:
  containers:
  - name: abc-tech-container
    image: kowshikreddy137/abc_tech:49
    ports:
    - containerPort: 8080
```

kind: Pod:

- Specifies the type of Kubernetes resource. In this case, it is a Pod, which is a single unit of deployment in Kubernetes.

metadata:

- Metadata provides information about the Pod.
- **name: abc-tech-pod**:
  - Sets the name of the Pod to `abc-tech-pod`.
- **Annotations**: Annotations are key-value pairs that can be used to attach arbitrary non-identifying metadata to the Pod. This field is empty in your YAML, but it can be used for various purposes like adding notes, specifying how a Pod should be treated

by a controller, etc.

Spec: Defines the desired state of the Pod, including the containers that run inside it.

- **Containers**: A list of containers that will run in this Pod. In this case, there is only one container.
    - **- name: abc-tech-container**:
        - The name of the container.
    - **image: kowshikreddy137/abc_tech:49**:
        - Specifies the Docker image to use for the container. In this case, it is `kowshikreddy137/abc_tech:49`, where `kowshikreddy137` is the Docker Hub username, `abc_tech` is the repository name, and `49` is the tag/version of the image.
    - **ports**:
        - Defines the ports that will be exposed by the container.
        - **- containerPort: 8080**: Exposes port `8080` on the container. This means that the container will listen for traffic on port `8080`.

Service.yml

This YAML file defines a Kubernetes Service. A Service is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable loose coupling between dependent Pods.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: abc-tech-service
spec:
  selector:
    app: abc-tech
  ports:
  - protocol: TCP
    port: 8082
    targetPort: 8082
  type: LoadBalancer
```

kind: Service:

- Specifies the type of Kubernetes resource. In this case, it is a Service, which provides a stable endpoint (IP and port) to access one or more Pods.

metadata:

- Metadata provides information about the Service.
- **name: abc-tech-service**:
    - Sets the name of the Service to `abc-tech-service`.

Spec: Defines the desired state of the Service.

- **Selector**: Defines how the Service identifies the Pods it should route traffic to.
    - **app: abc-tech**:
        - The Service will route traffic to Pods that have the label `app: abc-tech`.
- **Ports**: Defines the ports that the Service will expose and the corresponding ports on the Pods to which it will route traffic.
    - **- protocol: TCP**:
        - Specifies the protocol used by the Service. In this case, it is TCP.
    - **port: 8082**:
        - The port on which the Service will be exposed. Clients can connect to this port.
    - **targetPort: 8082**:
        - The port on the Pods to which the Service will forward traffic. In this case, it is also port 8082.
- **type: LoadBalancer**:
    - Specifies the type of Service. A LoadBalancer service exposes the Service externally using a cloud provider's load balancer. The cloud provider will provision a load balancer for the Service, and traffic to the load balancer will be forwarded to the Pods.

apply_resource.sh

I have written a script to run the Kubectl commands on the Linux machine through Jenkins

```
#!/bin/bash

# Get the build number from the Jenkins environment variable
BUILD_NUMBER=${BUILD_NUMBER}

# Switch to the appropriate directory
cd /var/lib/jenkins/workspace/Purdue_IGP/

# Apply the deployment YAML with build number
sudo -u ec2-user kubectl apply -f deployment.yml --namespace default

# Apply the pod YAML with build number
sudo -u ec2-user kubectl apply -f pod.yml --namespace default

# Apply the service YAML with build number
sudo -u ec2-user kubectl apply -f service.yml --namespace default
```

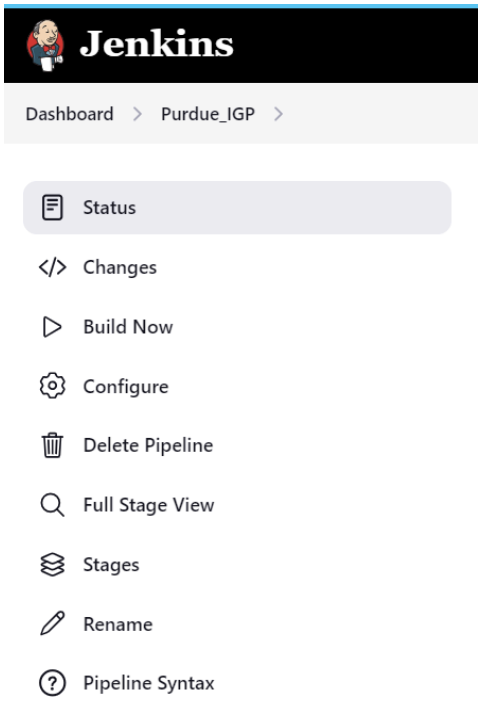
1. The first line retrieves the **BUILD_NUMBER** from the Jenkins environment variable.
2. Second line changes the current working directory to the specified workspace.
3. These commands apply the specified Kubernetes YAML files (**deployment.yml, pod.yml, and service.yml**) to the default namespace. The sudo -u ec2-user ensures the commands are run as the ec2-user.

Task 6: Running Pipeline

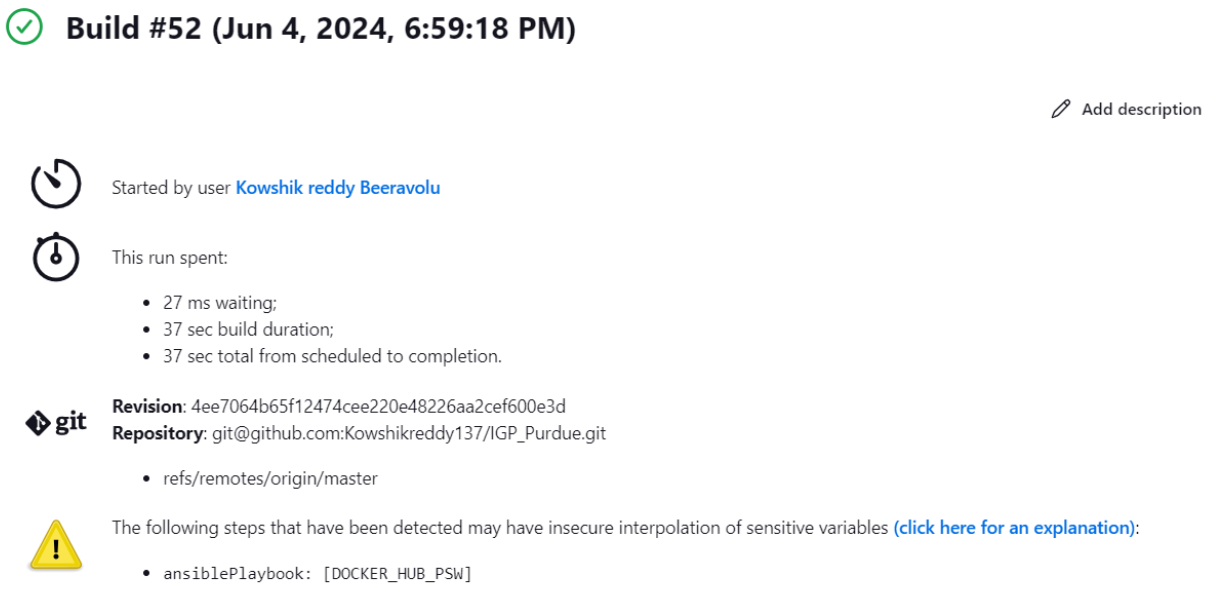
Now as we have all setup we can run the pipeline and check whether the application is being built , tested, create a package and then build a docker image and then deploy it on to kubernetes cluster.

To build the pipeline we have to login into jenkins dashboard and select the pipeline and in the left corner we will see different option regarding the pipeline.

Click on build now and pipeline starts building. And we can see the progress on the dashboard and detailed output in the console output section.



This is how it looks when we open the particular build, from here we can open the console output and do a detailed check.



This is the output for the code checkout stage of the pipeline

```
[Pipeline] git
The recommended git tool is: NONE
using credential Jenkins_slave_GIT
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Purdue_IGP/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@github.com:Kowshikreddy137/IGP_Purdue.git # timeout=10
Fetching upstream changes from git@github.com:Kowshikreddy137/IGP_Purdue.git
 > git --version # timeout=10
 > git --version # 'git version 2.40.1'
using GIT_SSH to set credentials
Verifying host key using known hosts file
 > git fetch --tags --force --progress -- git@github.com:Kowshikreddy137/IGP_Purdue.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 4ee7064b65f12474cee220e48226aa2cef600e3d (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 4ee7064b65f12474cee220e48226aa2cef600e3d # timeout=10
 > git branch -a -v --no-abbrev # timeout=10
 > git branch -D master # timeout=10
 > git checkout -b master 4ee7064b65f12474cee220e48226aa2cef600e3d # timeout=10
Commit message: "added inventory file for ansible k8s integration"
```

This is the output for the maven compile stage of the pipeline

```
+ mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.abc:ABCtechnologies >-----------------------
[INFO] Building RetailModule 1.0
[INFO] --------------------------------[ war ]---------------------------------
[WARNING] The artifact xml-apis:xml-apis:jar:2.0.2 has been relocated to xml-apis:xml-apis:jar:1.0.b2
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:prepare-agent (jacoco-initialize) @ ABCtechnologies ---
[INFO] argLine set to -javaagent:/var/lib/jenkins/.m2/repository/org/jacoco/org.jacoco.agent/0.8.6/org.jacoco.agent-0.8.6-
runtime.jar=destfile=/var/lib/jenkins/workspace/Purdue_IGP/target/jacoco.exec
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ ABCtechnologies ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Purdue_IGP/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ ABCtechnologies ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.245 s
[INFO] Finished at: 2024-06-04T18:59:23Z
[INFO] ------------------------------------------------------------------------
```

This is the output for the code test stage of the pipeline

```
+ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.abc:ABCtechnologies >-----------------------
[INFO] Building RetailModule 1.0
[INFO] --------------------------------[ war ]---------------------------------
[WARNING] The artifact xml-apis:xml-apis:jar:2.0.2 has been relocated to xml-apis:xml-apis:jar:1.0.b2
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:prepare-agent (jacoco-initialize) @ ABCtechnologies ---
[INFO] argLine set to -javaagent:/var/lib/jenkins/.m2/repository/org/jacoco/org.jacoco.agent/0.8.6/org.jacoco.agent-0.8.6-runtime.jar=destfi
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ ABCtechnologies ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Purdue_IGP/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ ABCtechnologies ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ ABCtechnologies ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Purdue_IGP/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ ABCtechnologies ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ ABCtechnologies ---
[INFO] Surefire report directory: /var/lib/jenkins/workspace/Purdue_IGP/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.abc.dataAccessObject.ProductImpTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.138 sec
```

```
Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.293 s
[INFO] Finished at: 2024-06-04T18:59:29Z
[INFO] ------------------------------------------------------------------------
[Pipeline] }
[Pipeline] // stage
```

This the output of the build stage of the pipeline

```
+ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.abc:ABCtechnologies >-----------------------
[INFO] Building RetailModule 1.0
[INFO] --------------------------------[ war ]---------------------------------
[WARNING] The artifact xml-apis:xml-apis:jar:2.0.2 has been relocated to xml-apis:xml-apis:jar:1.0.b2
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:prepare-agent (jacoco-initialize) @ ABCtechnologies ---
[INFO] argLine set to -javaagent:/var/lib/jenkins/.m2/repository/org/jacoco/org.jacoco.agent/0.8.6/org.jacoco.agent-0.8.6-runtim
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ ABCtechnologies ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Purdue_IGP/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ ABCtechnologies ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ ABCtechnologies ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Purdue_IGP/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ ABCtechnologies ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ ABCtechnologies ---
[INFO] Surefire report directory: /var/lib/jenkins/workspace/Purdue_IGP/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.abc.dataAccessObject.ProductImpTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.125 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
```

```
[INFO] --- maven-war-plugin:3.2.2:war (default-war) @ ABCtechnologies ---
[INFO] Packaging webapp
[INFO] Assembling webapp [ABCtechnologies] in [/var/lib/jenkins/workspace/Purdue_IGP/target/ABCtechnologies-1.0]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/Purdue_IGP/src/main/webapp]
[INFO] Webapp assembled in [93 msecs]
[INFO] Building war: /var/lib/jenkins/workspace/Purdue_IGP/target/ABCtechnologies-1.0.war
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:report (jacoco-site) @ ABCtechnologies ---
[INFO] Loading execution data file /var/lib/jenkins/workspace/Purdue_IGP/target/jacoco.exec
[INFO] Analyzed bundle 'RetailModule' with 2 classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.293 s
[INFO] Finished at: 2024-06-04T18:59:37Z
[INFO] ------------------------------------------------------------------------
```

This is the output of the ansible-playbook stage

```
Warning: A secret was passed to "ansiblePlaybook" using Groovy String interpolation, which is insecure.
                 Affected argument(s) used the following variable(s): [DOCKER_HUB_PSW]
                 See https://jenkins.io/redirect/groovy-string-interpolation for details.
[Purdue_IGP] $ ansible-playbook /var/lib/jenkins/workspace/Purdue_IGP/playbook.yml -i /etc/ansible/hosts -e
dockerhub_username=kowshikreddy137 -e dockerhub_password=****

PLAY [Build, Push, and Deploy Docker Container] *********************************

TASK [Gathering Facts] **********************************************************
ok: [127.0.0.1]

TASK [Stop and remove existing container] ****************************************
changed: [127.0.0.1]

TASK [Copy WAR file to Docker build context] *************************************
changed: [127.0.0.1]

TASK [Build Docker image] *******************************************************
changed: [127.0.0.1]

TASK [Tag Docker image] **********************************************************
changed: [127.0.0.1]

TASK [Log in to Docker Hub] ******************************************************
ok: [127.0.0.1]
```

```
TASK [Debug login result] ********************************************************
ok: [127.0.0.1] => {
    "login_result": {
        "changed": false,
        "failed": false,
        "login_result": {
            "email": null,
            "serveraddress": "https://index.docker.io/v1/",
            "username": "kowshikreddy137"
        }
    }
}

TASK [Push Docker image to Docker Hub] *******************************************
changed: [127.0.0.1]

TASK [Run Docker container] ******************************************************
changed: [127.0.0.1]

PLAY RECAP ***********************************************************************
127.0.0.1                  : ok=9    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

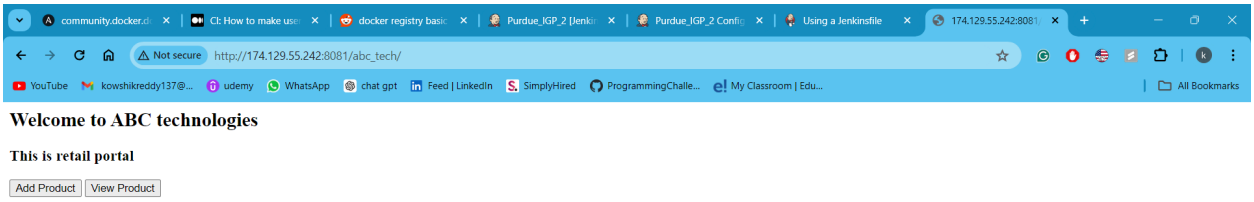
This is the output of the deploy k8s stage

```
[Pipeline] { (Deploy k8s)
[Pipeline] sh
+ chmod +x /var/lib/jenkins/workspace/Purdue_IGP/apply_resources.sh
[Pipeline] sh
+ ./apply_resources.sh 52
deployment.apps/abc-tech-deployment created
pod/abc-tech-pod created
service/abc-tech-service created
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline completed.
[Pipeline] echo
Pipeline succeeded.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

This is the stage view of the pipeline running

Stage View

	Code Checkout	Code Compile	Test	Build	Run Ansible Playbook	Deploy k8s	Declarative: Post Actions
Average stage times: (Average full run time: ~38s)	607ms	4s	5s	7s	11s	4s	123ms
#52 Jun 04 13:59 No Changes	470ms	4s	5s	7s	11s	6s	126ms
#51 Jun 04 13:58 No Changes	668ms	4s	5s	7s	10s failed	92ms failed	96ms
#50 Jun 04 13:49 No Changes	547ms	5s	6s	7s	10s	6s	168ms
#49 Jun 04 13:43 No Changes	570ms	4s	5s	Success Logs	11s	9s	72ms
#48 Jun 04 13:36 No Changes	604ms	4s	5s	6s	11s	6s	121ms
#47 Jun 04 13:24 No Changes	596ms	5s	6s	8s	11s	6s	120ms



Task 7: Monitoring with Prometheus and Grafana

I have setup monitoring the application using Prometheus, Node Exporter, and Grafana.

By following these steps, we should have Prometheus collecting metrics from Node Exporter, and Grafana visualizing these metrics through a dashboard.

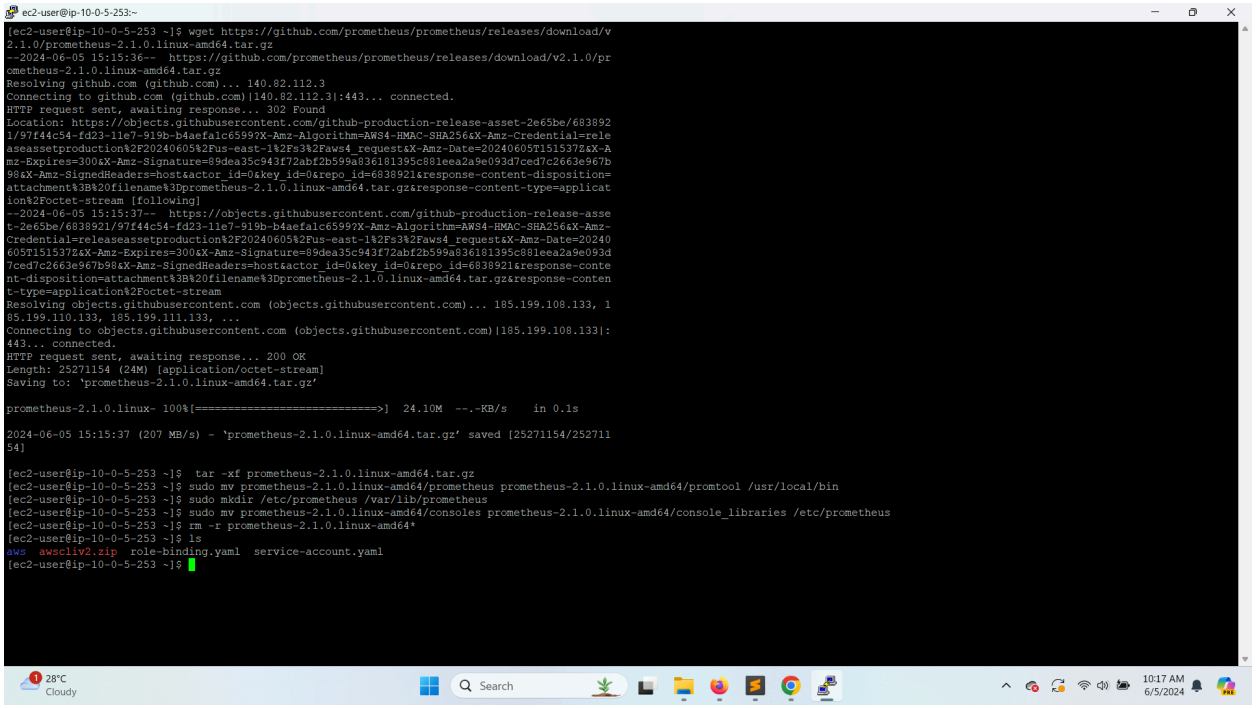
Install Prometheus

1. Download the Prometheus archive.
2. Extract the Prometheus archive.
3. Move the binaries to /usr/local/bin
4. Now, we need to create directories for configuration files and other Prometheus data.
5. Then, we move the configuration files to the directory we made previously:
6. Finally, we can delete the leftover files as we do not need them anymore:
7. We will use /etc/prometheus/prometheus.yml as our configuration file.

```
scrape_configs:
  - job_name: 'prometheus_metrics'
    scrape_interval: 5s
    static_configs:
     - targets: ['localhost:9090']
  - job_name: 'node_exporter'
    scrape_interval: 15s
    static_configs:
     - targets: ['localhost:9100']
```

8. We will also change the ownership of files that Prometheus will use
9. Then, we will create a systemd unit file in /etc/systemd/system/prometheus.service with the following contents :

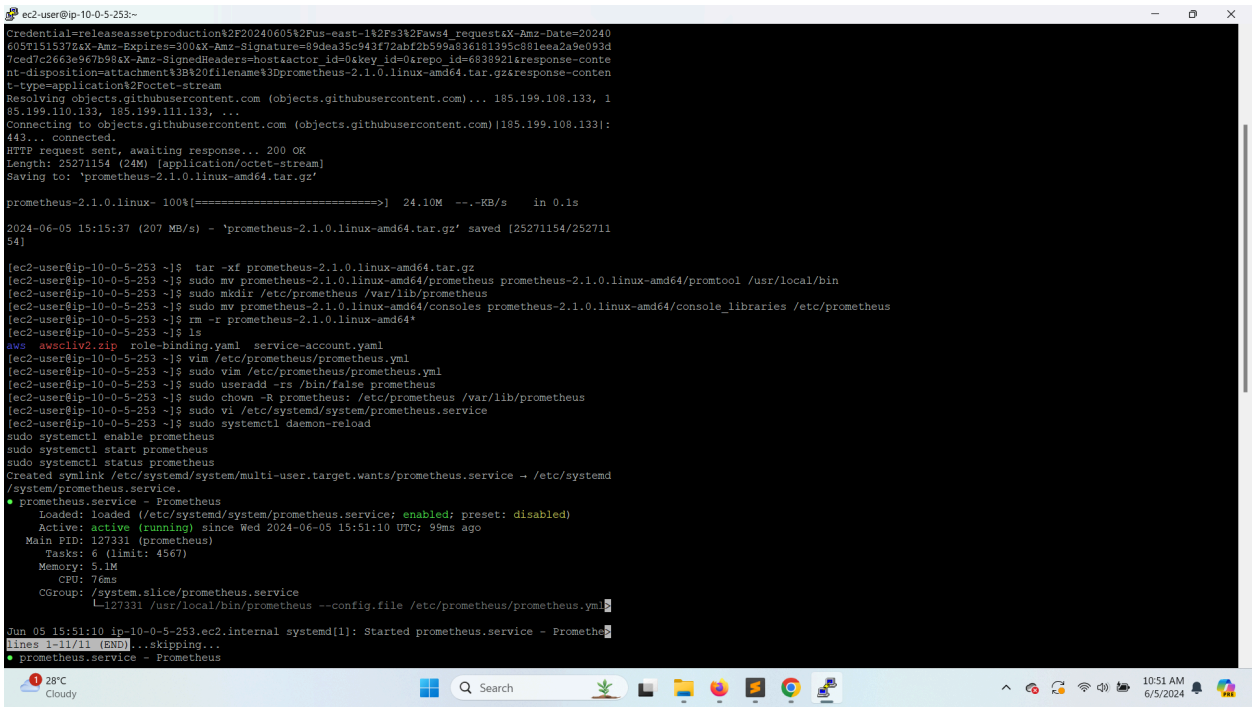
Screenshot of the above mentioned steps




```
[Unit]
Description=Prometheus
After=network.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
  --config.file=/etc/prometheus/prometheus.yml \
  --storage.tsdb.path=/var/lib/prometheus/ \
  --web.console.templates=/etc/prometheus/consoles \
  --web.console.libraries=/etc/prometheus/console_libraries
Restart=always

[Install]
WantedBy=multi-user.target
```

10. Then we can run the necessary commands to get prometheus running

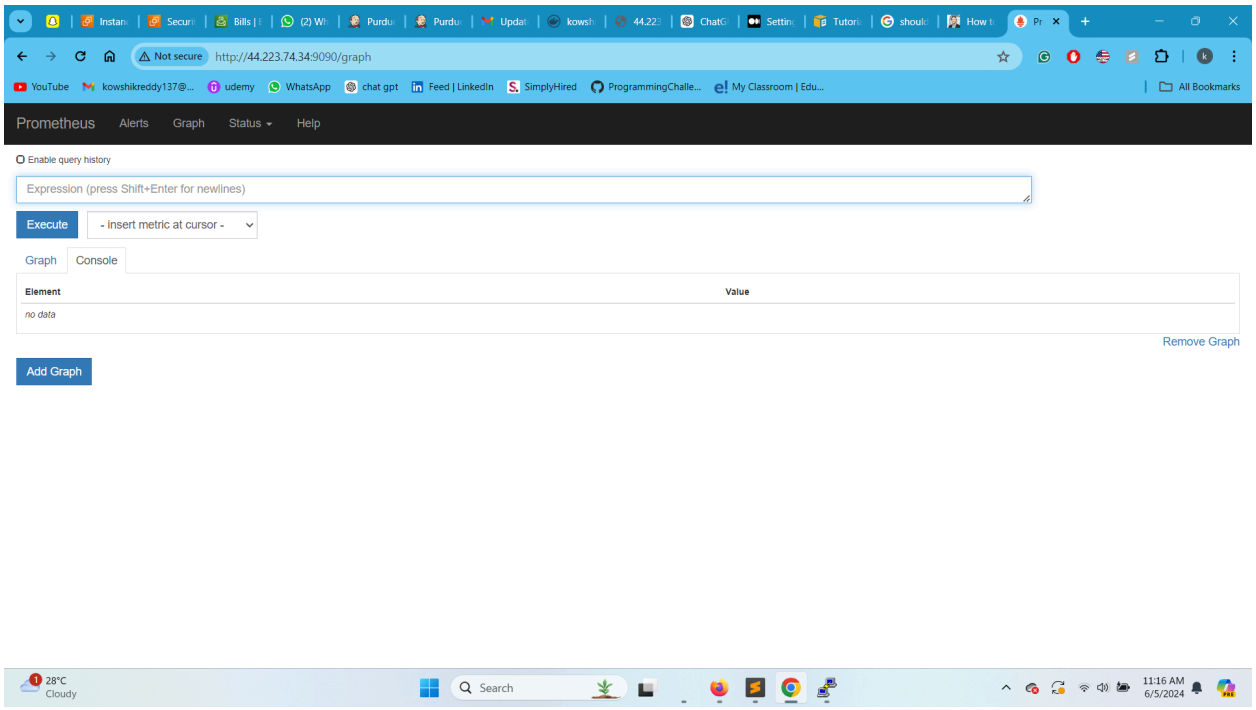
```
sudo systemctl daemon-reload
sudo systemctl enable prometheus
sudo systemctl start prometheus
sudo systemctl status prometheus
```



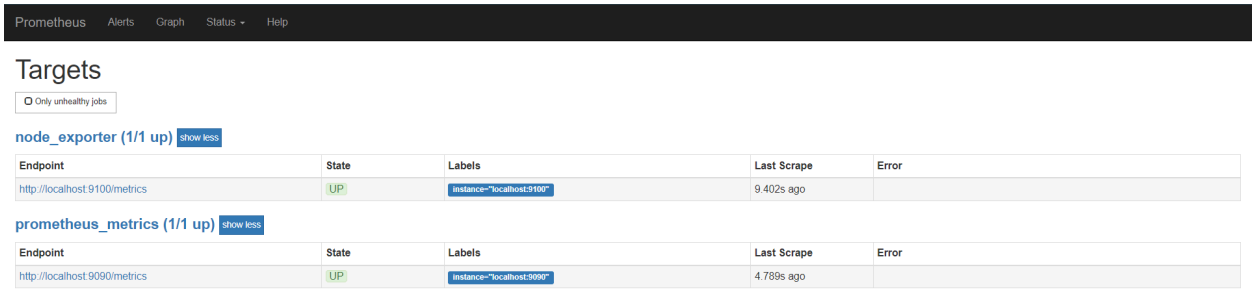
Prometheus provides a web UI to run the basic commands. We can access this website by using

`http://<ip-address>:9090`

To make sure we can access the website we have to add the port to the AWS security groups to allow traffic.

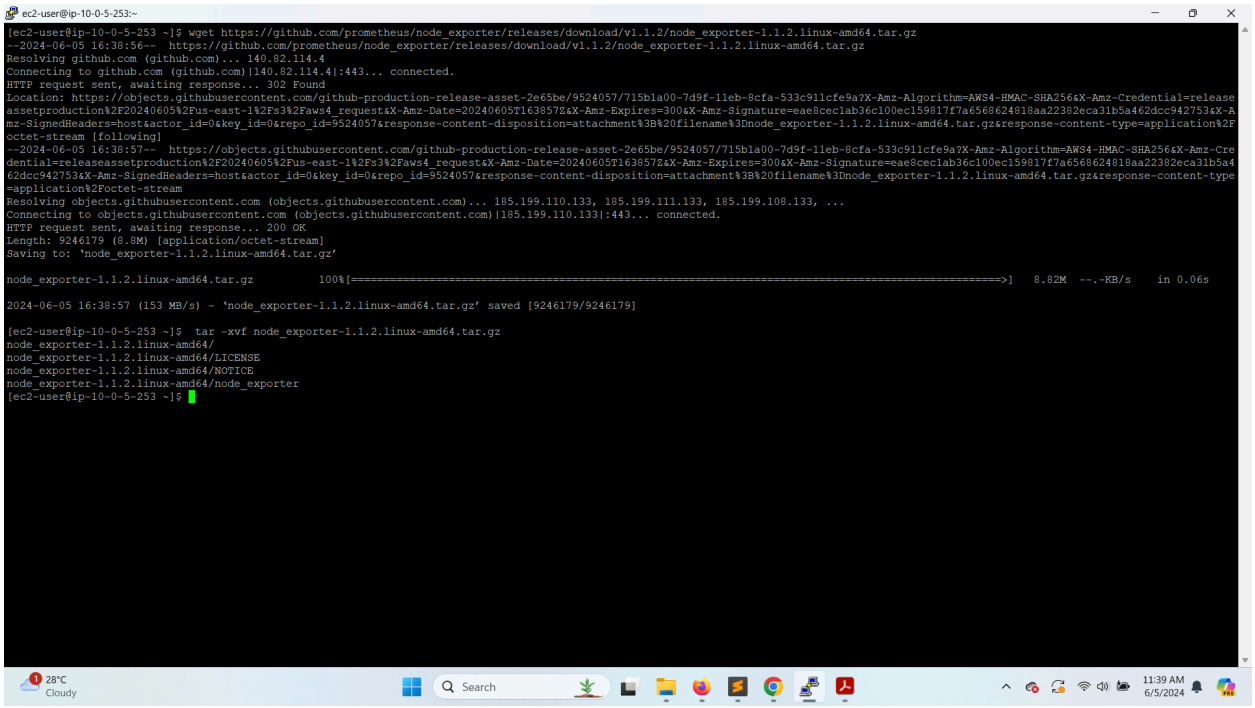


As we haven't set up the node exporter yet we will see the state of the node exporter as down. But in this case, I have taken the screenshot after setting up the noe exporter as well.



Node Exporter setup

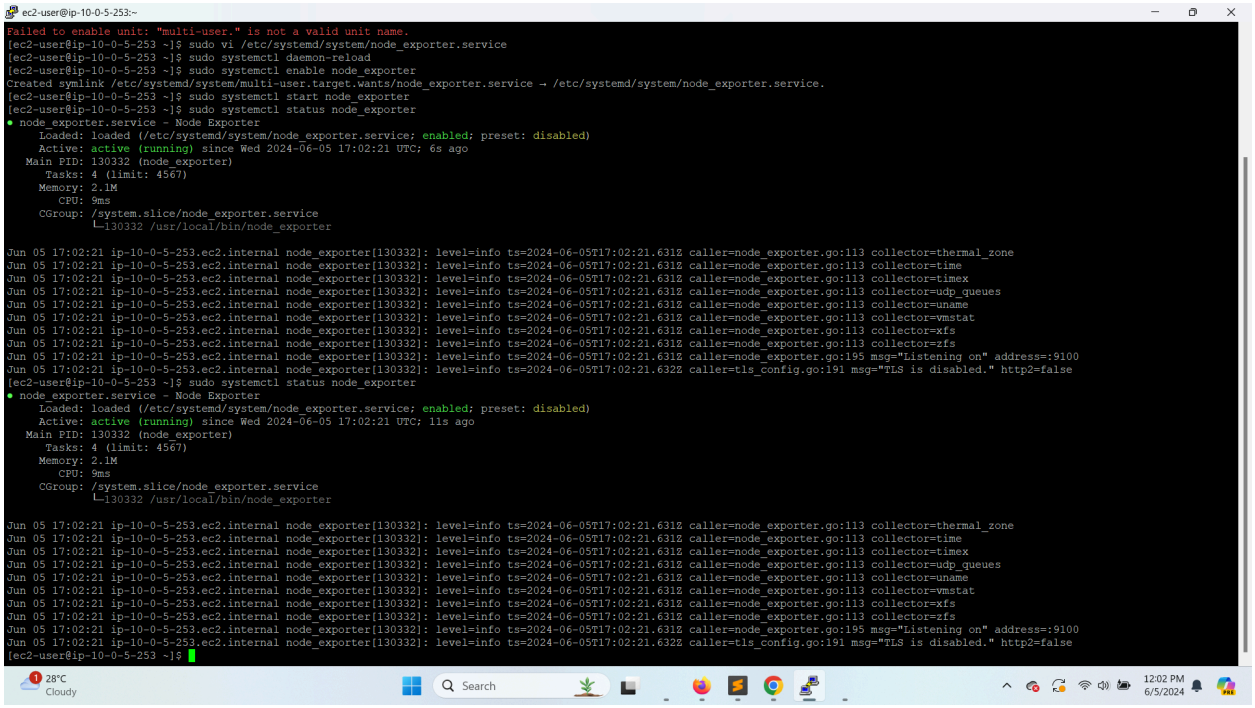
1. We will download the Node Exporter on our machine.
2. Extract the downloaded archive.
3. Move the node_exporter binary to /usr/local/bin
4. Remove the residual files.



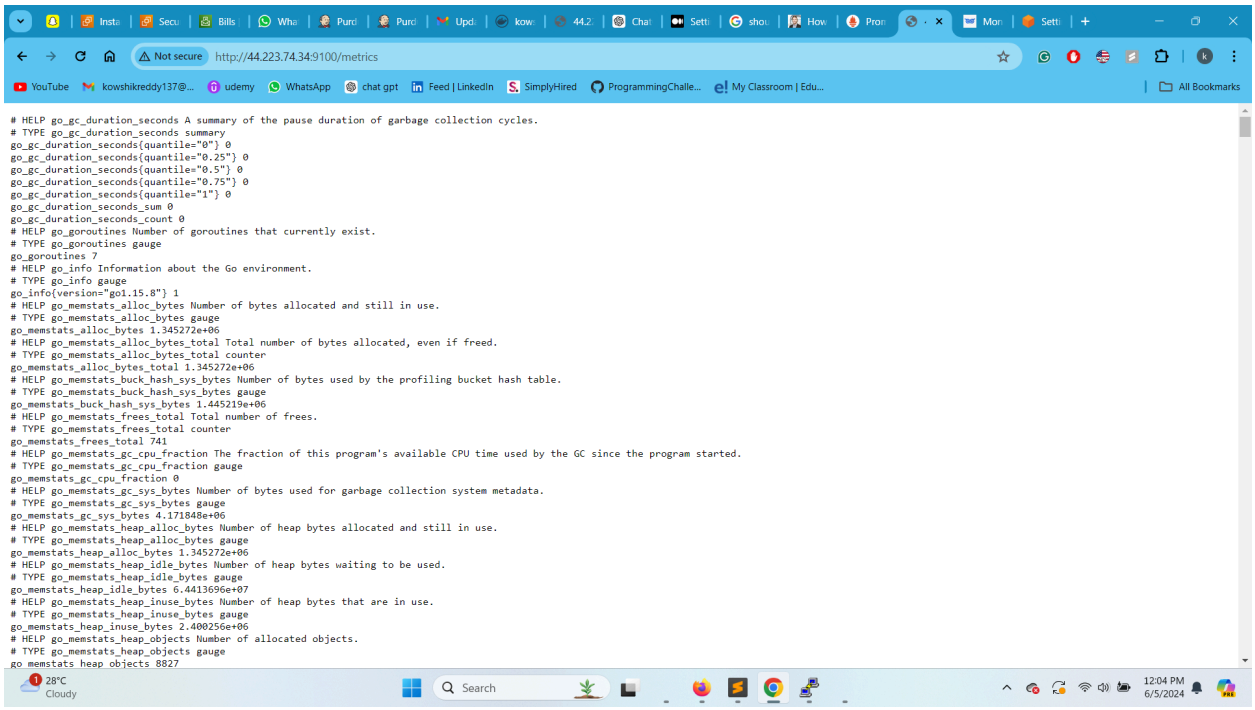
5. Next, we will create users and service files for node_exporter.
6. We will create users and service files for node_exporter.
7. Then, we will create a systemd unit file so that node_exporter can be started at boot.
8. Since we have created a new unit file, we must reload the systemd daemon, set the service to always run at boot, and start it using the below commands

```
sudo systemctl daemon-reload
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
sudo systemctl status node_expor
```

9. We can access the node exporter application by accessing http://<ip-address>:9100



This is how we will be able to see the node metrics.



Grafana Installation

1. we add a new YUM repository for the operating system to know where to download Grafana.
2. Add the lines below to grafana.repo. This setting will install to the Open Source version of Grafana

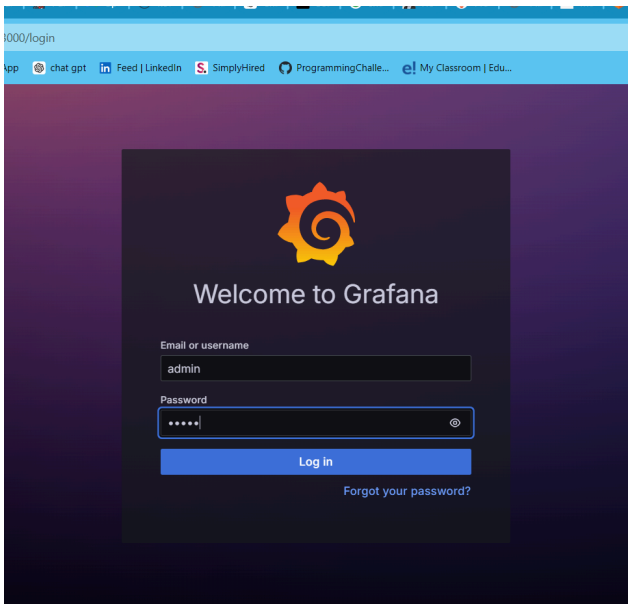
```
[grafana]
name=grafana
baseurl=https://packages.grafana.com/oss/rpm
repo_gpgcheck=1
enabled=1
gpgcheck=1
gpgkey=https://packages.grafana.com/gpg.key
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
```

3. Then we can install grafana by using *yum install grafana*
4. Reload the systemd to load the new settings. Start the Grafana Server, then check for its status.

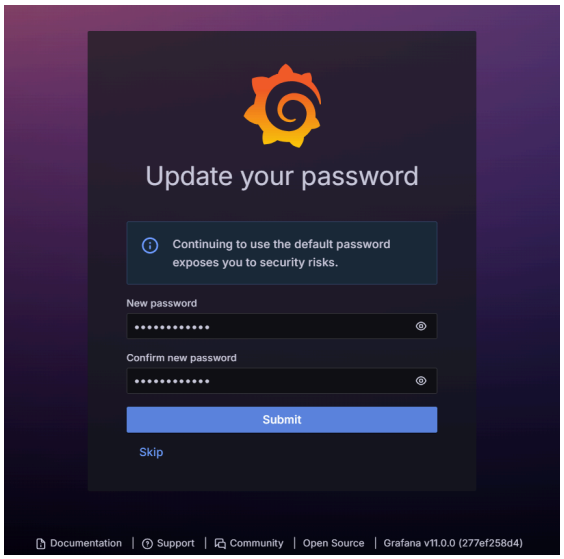
```
sudo systemctl daemon-reload
sudo systemctl start grafana-server
sudo systemctl enable grafana-server.service
sudo systemctl status grafana-server
```

5. Make sure to add the port 3000 in the inbound rules of security group.
6. We can connect to the grafana dashboard by accessing this link
   http://<ip_address>:3000
7. After accessing the website it will prompt us to enter username and password, for the first time we have to use admin and admin to login.
8. Then it will ask us to choose a new password, then we can set the new password.
9. Add Prometheus as a Data Source in Grafana:
   - Log in to Grafana.
   - Go to "Configuration" > "Data Sources".
   - Click on "Add data source".
   - Choose Prometheus.
   - Set the URL to http://localhost:9090 (assuming Prometheus is running on the same machine).
   - Save and test the data source.

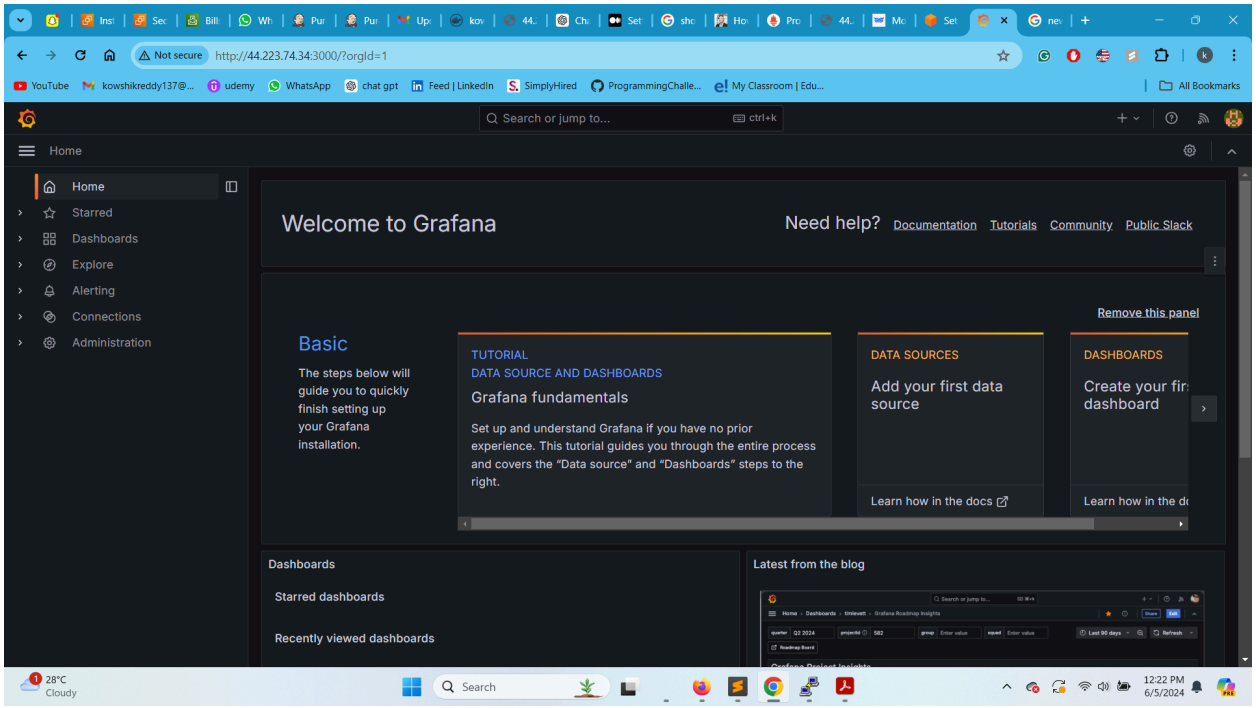
Logging in for the first time



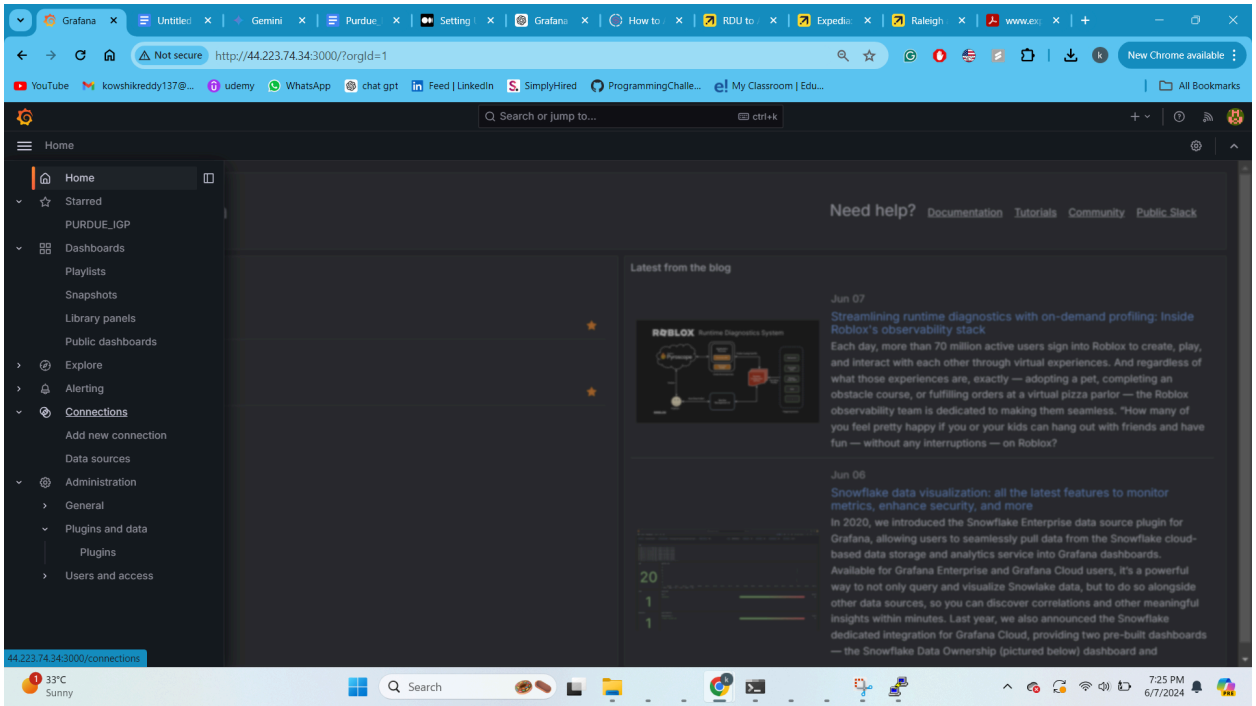
Setting a new password



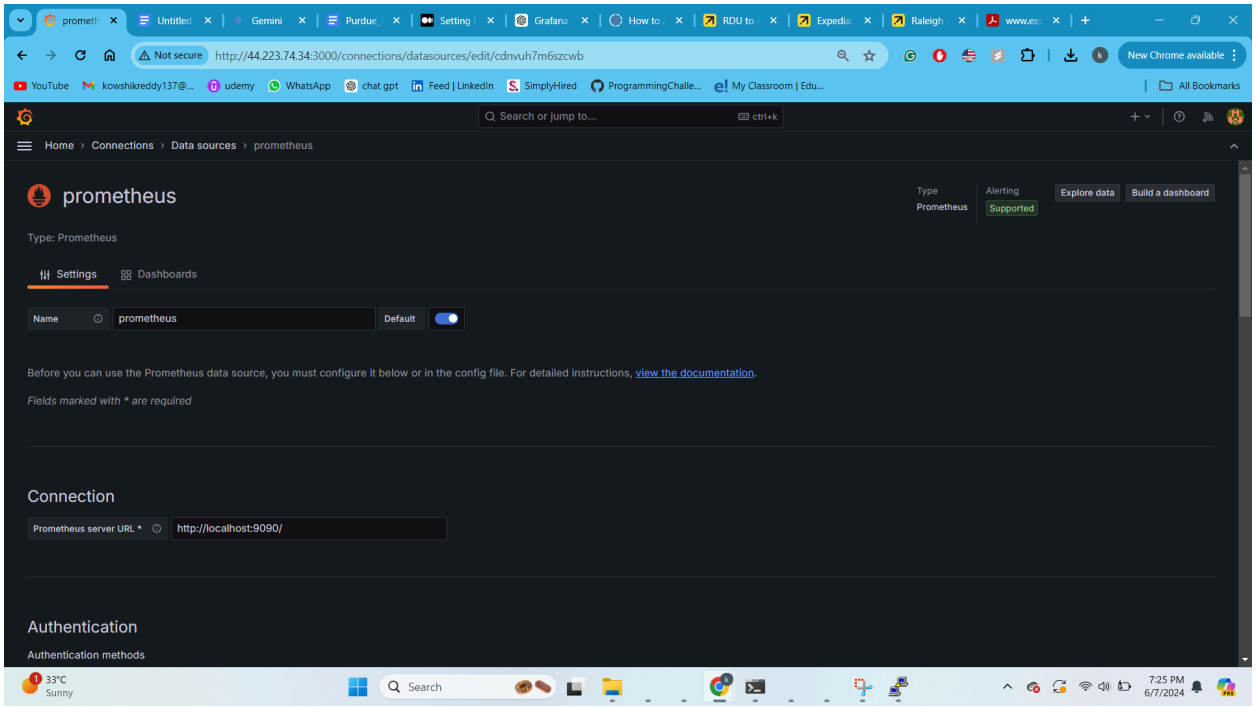
This is how the welcome page looks like

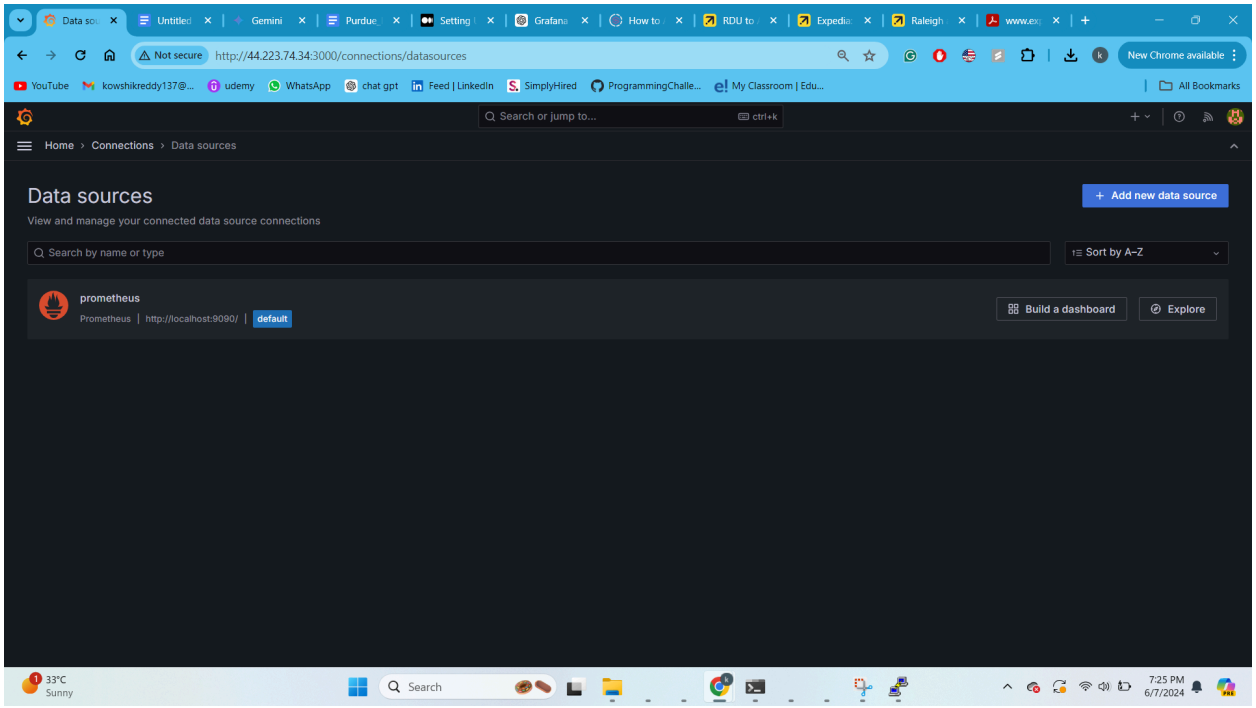


Configure a data source

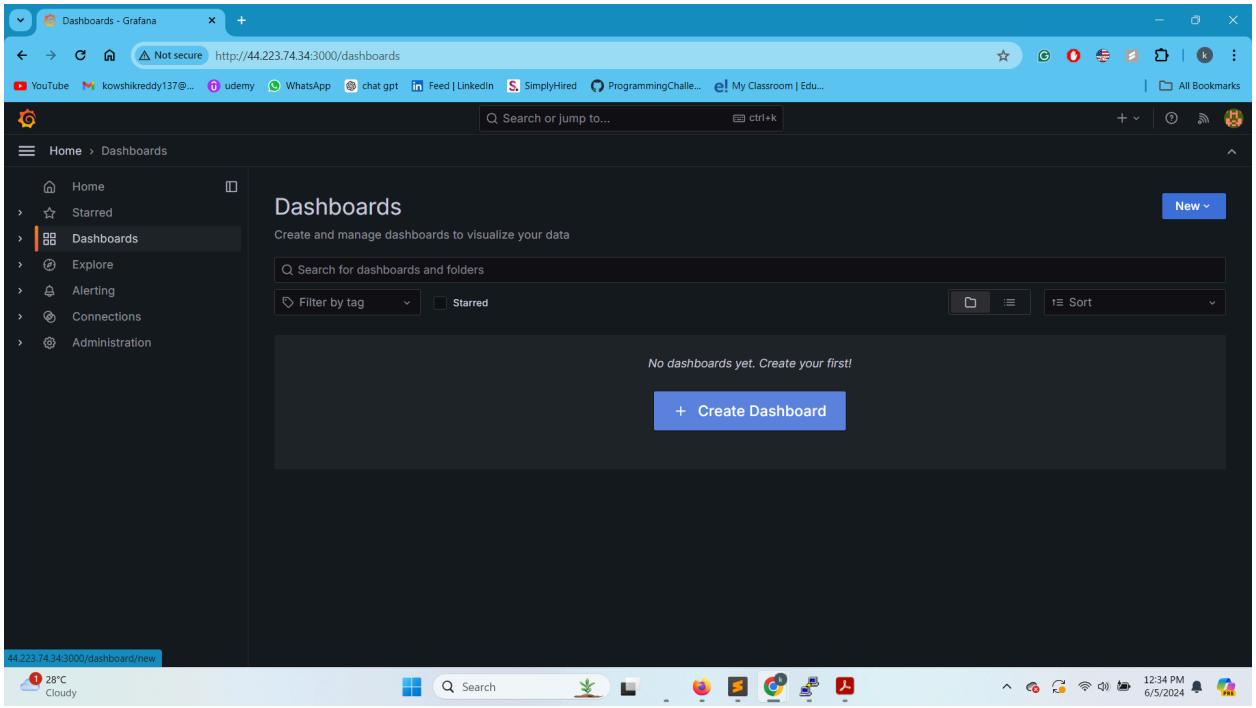


Configuring the data source by selecting Prometheus and giving the Url for pormetheus

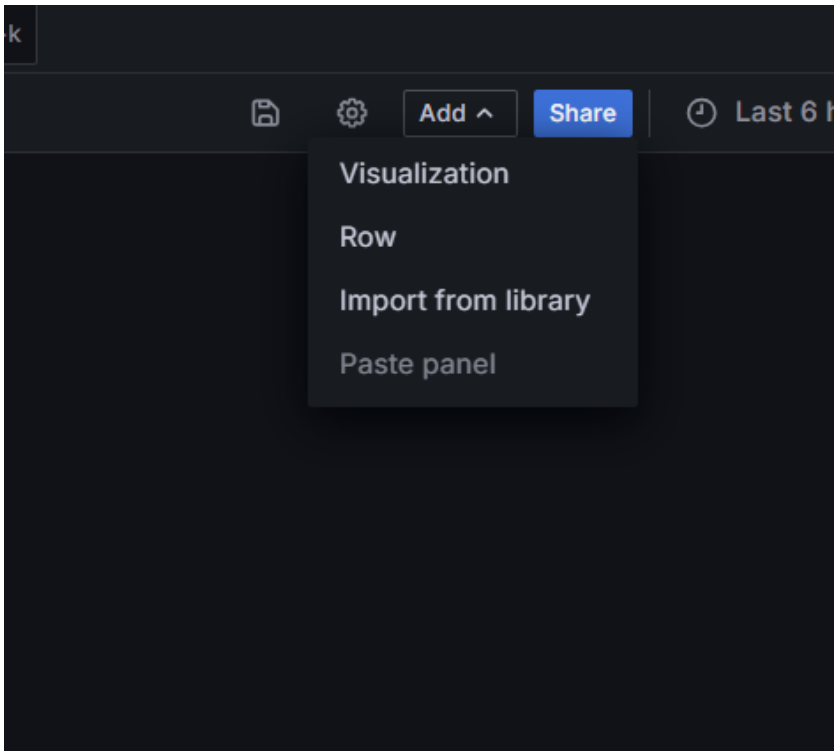

Not secure
http://44.223.74.34:3000/connections/datasources/edit/cdnvuh7m6szcwb
New Chrome available
Home › Connections › Data sources › prometheus
prometheus
Type: Prometheus
Type
Prometheus
Alerting
Supported
Explore data
Build a dashboard
Settings
Dashboards
Name
prometheus
Default
Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, view the documentation.
Fields marked with * are required
Connection
Prometheus server URL *
http://localhost:9090/
Authentication
Authentication methods


Not secure
http://44.223.74.34:3000/connections/datasources
New Chrome available
Home › Connections › Data sources
Data sources
View and manage your connected data source connections
+ Add new data source
Search by name or type
Sort by A-Z
prometheus
Prometheus | http://localhost:9090/ | default
Build a dashboard
Explore

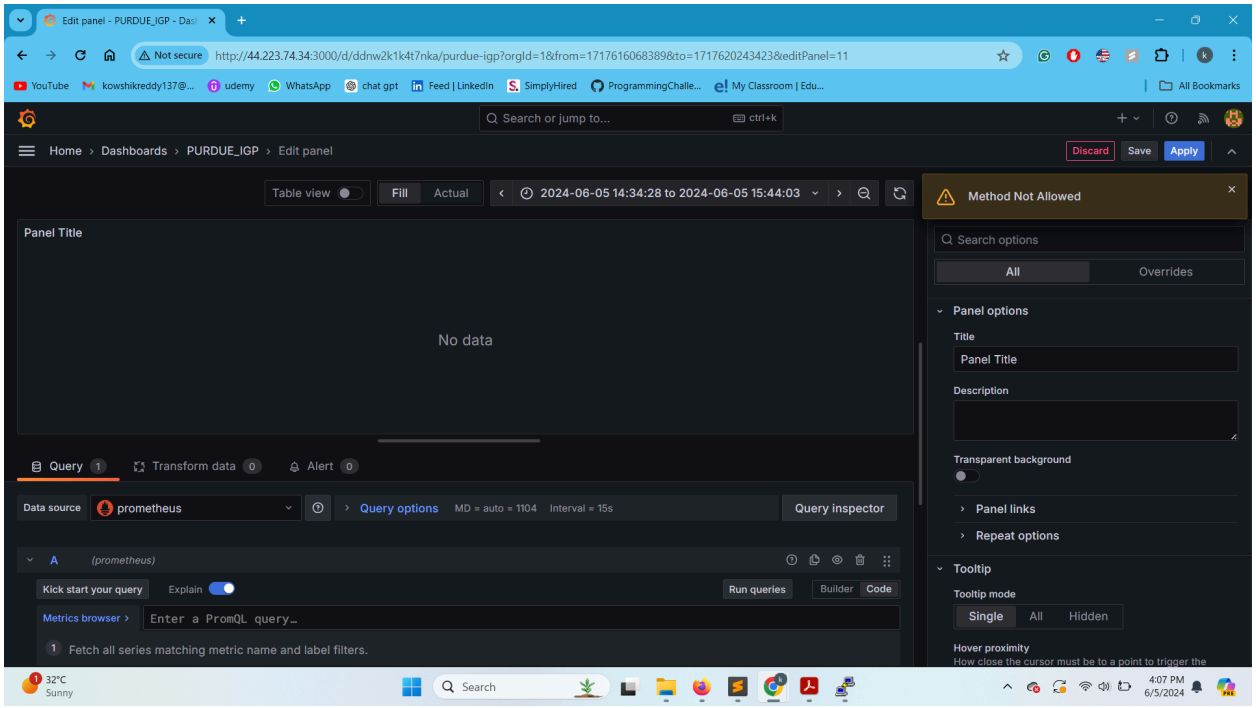
Create a new dashboard



Select visualization in the dropdown for add button.



We can enter the query in the PromQL text box and click on run query



Create a Dashboard in Grafana:

1. Go to "Create" > "Dashboard".
2. Click on "Add new panel".
3. Choose a visualization type (e.g., Graph, Singlestat, Table) depending on the metric you want to display.
4. In the query editor, write PromQL queries to retrieve the metrics you want to visualize. Here are some example queries:
    a. CPU total usage:

```
100 - avg by (cpu) (irate(node_cpu_seconds_total{mode="idle"}[1m])) *
100
```

    b. CPU usage per core:

```
100 - (avg by (instance, cpu)
(irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

    c. CPU Usage Breakdown

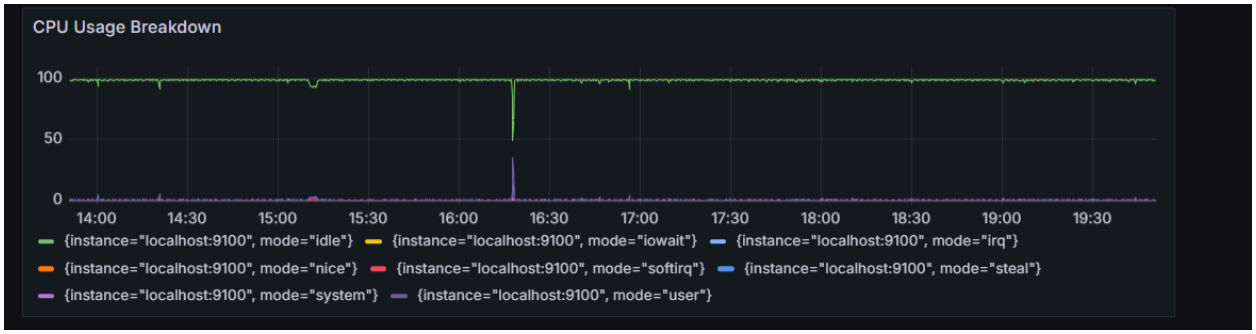
```
avg by (instance, mode) (irate(node_cpu_seconds_total[5m])) * 100
```

    d. Memory Usage:

```
100 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)
* 100
```


Memory
Memory Usage
65
60
55
50
14:00
14:30
15:00
15:30
16:00
16:30
17:00
17:30
18:00
18:30
19:00
19:30
{instance="localhost:9100", job="node_exporter"}
Network Metrics
Network Throughput
6000000
4000000
2000000
0
ens5


CPU
CPU Total Usage
40
30
20
10
0
0
1
CPU Usage Per Core
{cpu="0", instance="localhost:9100"}
{cpu="1", instance="localhost:9100"}



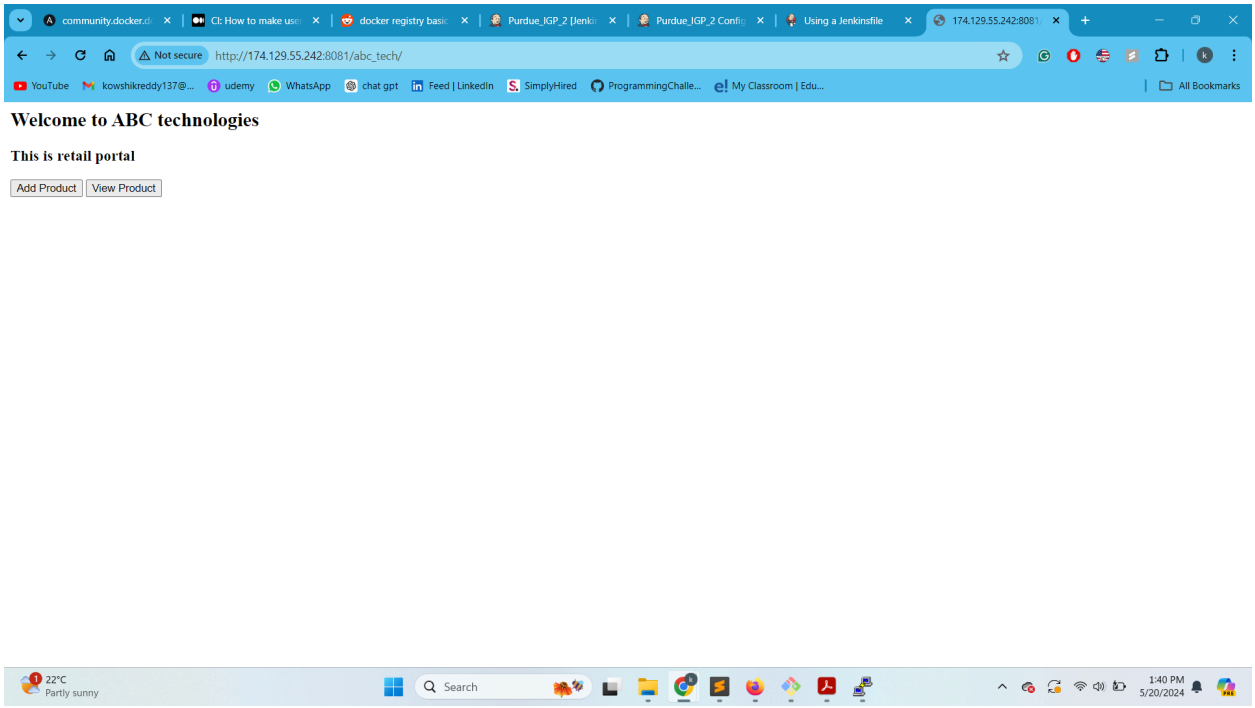
Conclusion

The implementation of the CI/CD pipeline for ABC Technologies marks a significant improvement in the company's software development and deployment processes. By leveraging modern DevOps tools and practices, we have addressed key challenges such as low availability, scalability, and performance. The integration of Git, Jenkins, Docker, Ansible, Kubernetes, Prometheus, and Grafana has resulted in a highly automated, efficient, and reliable system.

Key accomplishments of this project include:

1. **Automation of Development and Deployment**: The pipeline automates the entire process from code commit to deployment, significantly reducing manual intervention and the potential for errors.
2. **Enhanced Scalability and Performance**: With Kubernetes orchestration and Docker containerization, the application can scale seamlessly to handle increased loads, ensuring consistent performance.
3. **Improved Monitoring and Maintenance**: The integration of Prometheus and Grafana provides real-time monitoring and visualization of system metrics, allowing for proactive maintenance and quick resolution of issues.

Output after the final rundown



4. **Faster Time to Market**: The streamlined CI/CD pipeline accelerates the release cycles, enabling quicker delivery of new features and updates to customers.
5. **Better Collaboration and Code Quality**: The use of Git and Jenkins facilitates better collaboration among development teams, while continuous integration ensures that code changes are thoroughly tested and validated.

This project not only enhances the technical capabilities of ABC Technologies but also positions the company for future growth and innovation. The new CI/CD pipeline supports the company's strategic goals of becoming highly available, scalable, and performant, while also improving the overall customer experience. With this robust infrastructure in place, ABC Technologies is well-equipped to adapt to the evolving demands of the retail industry and maintain a competitive edge.