**DevOps Project Report**

**CI/CD Pipeline Using Git, Jenkins, Ansible & Kubernetes**

# Overview

The primary objective of this project was to automate the entire software delivery pipeline. This includes fetching code from a GitHub repository, building it with Jenkins, deploying it using Ansible, and orchestrating containers in a Kubernetes cluster. By integrating these tools, the project ensures seamless and error-free deployments, which are repeatable and scalable.

This project is not just a technical implementation; it represents a paradigm shift in how software is developed, delivered, and maintained. Through this hands-on exercise, I was able to understand the practical importance of DevOps in bridging the gap between development and operations teams DevOps tools. Each strategy progressively introduces additional tools to automate and streamline deployment, improve scalability, and enable modern container orchestration:

# Technologies & Tools:

**1. Git & GitHub**

* Git: A distributed version control system used for tracking code changes.
* GitHub: A cloud platform for hosting repositories and triggering webhooks for automated build processes.

**2. Jenkins**

* Jenkins: An open-source automation server used to build and test code continuously. It integrates with GitHub to trigger jobs automatically.

**3. Maven**

* Apache Maven: A Java-based build automation tool used to manage dependencies and build the project.

**4. Docker**

* Docker: Used to containerize the application so it can run consistently across different environments.

**5. Ansible**

* Ansible: An automation tool used to configure systems, deploy applications, and orchestrate tasks across multiple servers.

**6. Kubernetes**

* Kubernetes (Mini Kube): A container orchestration platform used to manage and scale containerized applications.

**7. AWS**

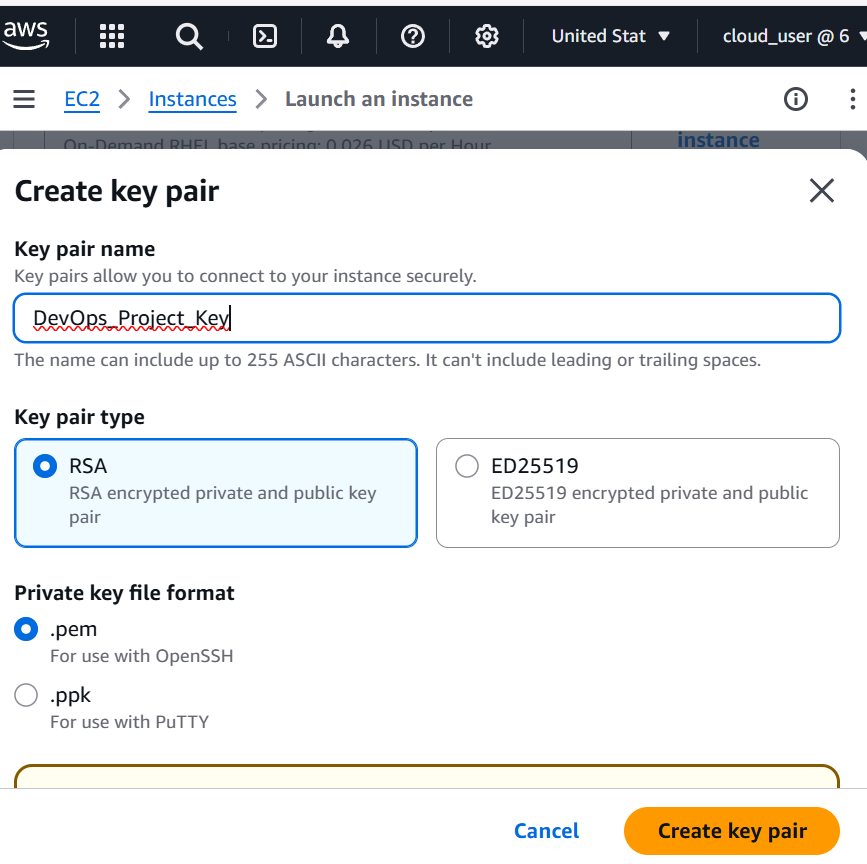
* A cloud platform for setting up, managing, and scaling infrastructure effortlessly.

# Infrastructure Setup and Common Prerequisites

Jenkins Installation on AWS EC2 Instance. Here we are using **Amazon Linux 2** for the configuration. These are the initial steps for setting up the Git, Java 17and Jenkins on OS.

A screenshot of a computer

AI-generated content may be incorrect.



• **Update the OS and install Git:**

* sudo yum update -y
* sudo yum install git -y
* git --version

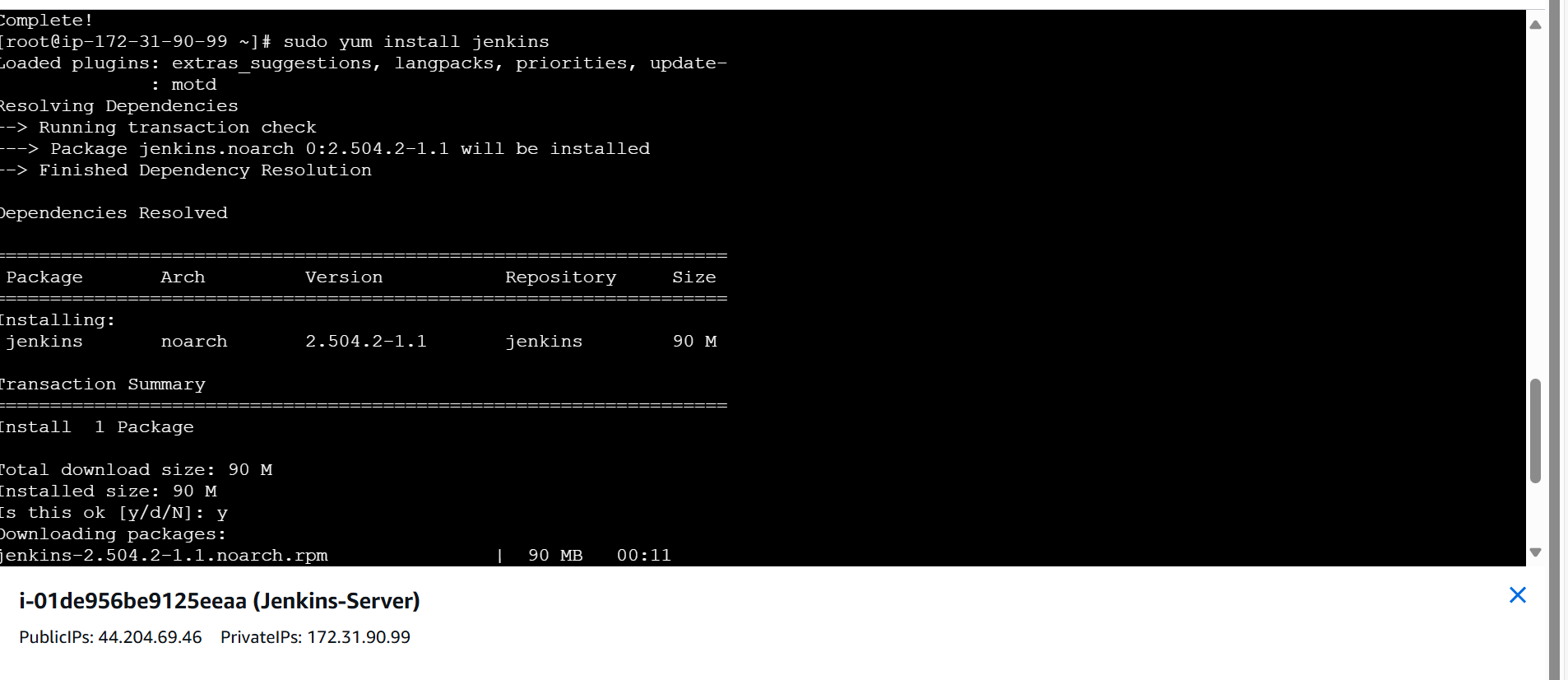
**• Install Java:**

* sudo yum install java-17-amazon-corretto -y
* java –version

**• Configure AWS security group to allow port 8080 for Jenkins access or it can be done when creating the EC2 instance.**

**• Install Jenkins:**

* 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-023.key>
* sudo yum upgrade
* sudo yum install jenkins -y
* sudo systemctl enable Jenkins
* sudo systemctl start Jenkins
* sudo systemctl status Jenkins

****

**• Access Jenkins:**

* URL: EC2\_Instance\_public\_IP:8080

**• Jenkins password:**

* sudo cat /var/lib/jenkins/secrets/initialAdminPassword

• **Install Git plugin in Jenkins & Configure it:**

* Manage Jenkins → Plugins → Available Plugins → GitHub
* Manage Jenkins → Global Tool Configuration → Git

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Maven and Jenkins Deployment

In this deployment, **Maven** is used to build the Java project, ensuring efficient dependency management and compilation. A **CI/CD pipeline** is then set up to automatically fetch the latest code from **GitHub**, store it in the **Jenkins server** or **Jenkins workspace**, and prepare it for further processing or deployment.

Here are the steps to install and configure the maven in Jenkins as well as in OS:

* **Maven Plugin on Jenkins & Configuration of Java and Maven:**
  + Manage Jenkins → Plugins → Available Plugins → Maven Integration
  + Manage Jenkins → Global Tool Configuration →Maven
  + Manage Jenkins → Global Tool Configuration →Java
* **Installation of Maven:**
  + **cd /opt/**
  + sudo wget <https://downloads.apache.org/maven/maven-3/3.9.10/binaries/apache-maven-3.9.10-bin.tar.gz>
  + sudo tar -xvzf apache-maven-3.9.10-bin.tar.gz
  + sudo mv apache-maven-3.9.10-bin.tar.gz maven

(rename the filename to maven)

* **Find Java Path:**
  + find / -name jvm
  + find / -name java-17\*
* **Add Path into the file named .bash\_profile :**
  + Sudo nano .bash\_profile

//Write below code in .bash\_profile file

M2\_HOME=/opt/maven

M2=/opt/maven/bin

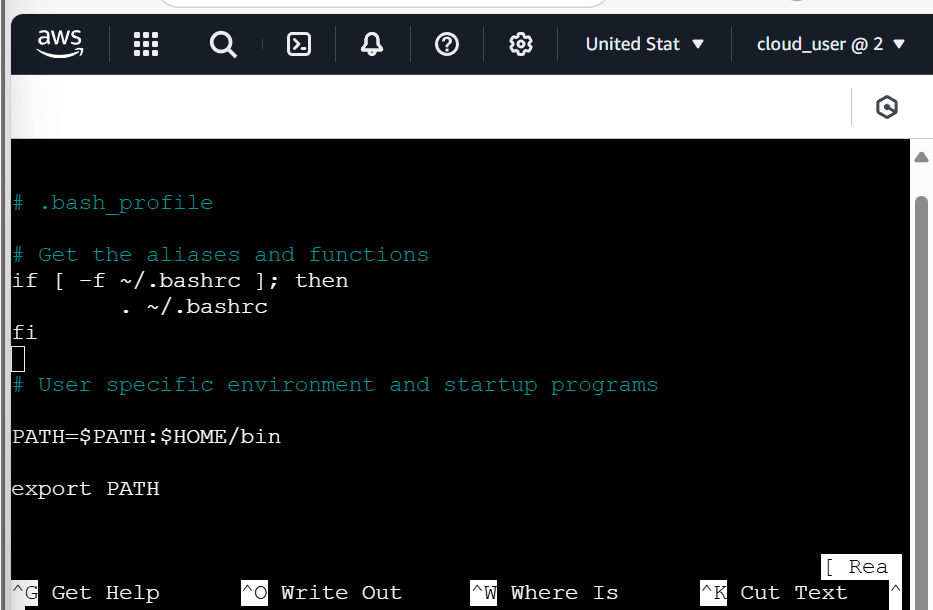
JAVA\_HOME=/usr/lib/jvm/java-17-amazon-corretto.x86\_64

PATH=$PATH:$HOME/bin:$JAVA\_HOME:$M2\_HOME:$M2

* **To Apply changes:**
  + Sudo source .bash\_profile
* **After configuring Java and Maven paths in Jenkins, we proceed with setting up the pipeline:**
* **Create a new Jenkins job:**  
  Start by making a new job in Jenkins. You can give it any name you like. Choose **"Maven project"** as the type of job.
* **Connect to GitHub:**  
  Add the link to your GitHub repository where your project files (especially the pom.xml file) are stored. This tells Jenkins where to get your code from.
* **Set the Maven goal:**  
  In the build settings, enter clean install as the Maven goal. This makes sure Jenkins deletes any old files and builds everything fresh from the start.
* **Run the build:**  
  When you click **Build**, Jenkins will pull the code from GitHub, run the Maven commands, and create the necessary files to run your Java application.

A screenshot of a computer

AI-generated content may be incorrect.

 A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Maven, Tomcat, and Jenkins Deployment

In this deployment, we first set up a **Tomcat Server** to host and run our Java application. Then, we use **Maven** to build the project, which helps manage all the required files and dependencies. To make the process automatic, we create a **CI/CD pipeline** using **Jenkins**. Jenkins pulls the latest code from **GitHub**, builds it using Maven, and then deploys it directly to the **Tomcat Server**—all without needing to do it manually.

Here are the steps to install and configure the Tomcat in Jenkins as well as in OS

* **Installing Tomcat Server in OS:**
  + sudo cd /opt/
  + sudo wget <https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.41/bin/apache-tomcat-10.1.41.tar.gz>
  + sudo tar xzvf apache-tomcat-10.1.41.tar.gz
  + sudo mv apache-tomcat-10.1.41.tar.gz tomcat (rename the file to tomcat)
* **To Start the Tomcat server:**
  + cd /opt/tomcat/bin
  + sudo ./startup.sh
* **To Access Tomcat we need to provide access to instance ports:**
  + cd ~
  + find / -name context.xml

// To allow access to all ports on the instance, **comment out** the command restricting access to localhost. This ensures the server can be reached externally. In below files

* + Sudo nano /opt/tomcat/webapp/host-manager/META-INF/context.xml
  + Sudo nano /opt/tomcat/webapp/manager/META-INF/context.xml
* **After Providing the port access we need to add the user credentials in tomcat-users.xml file :**
  + cd /opt/tomcat/conf/
  + sudo nano tomcat-users.xml
  + Then add user at the bottom of the file  
    <role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

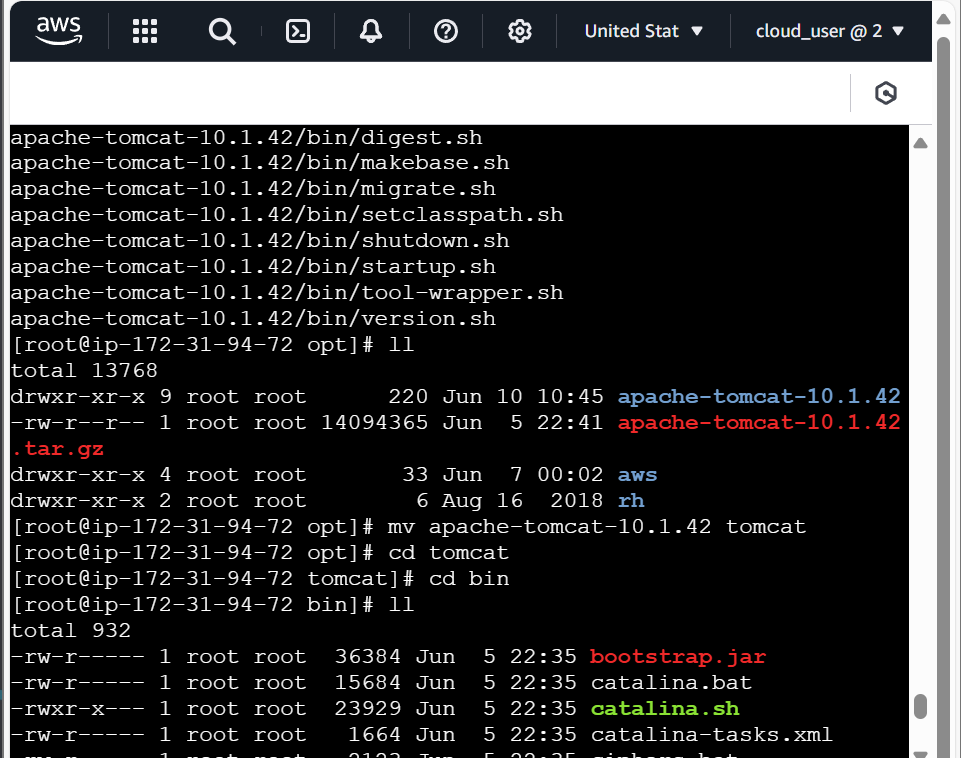
<role rolename="manager-status"/>

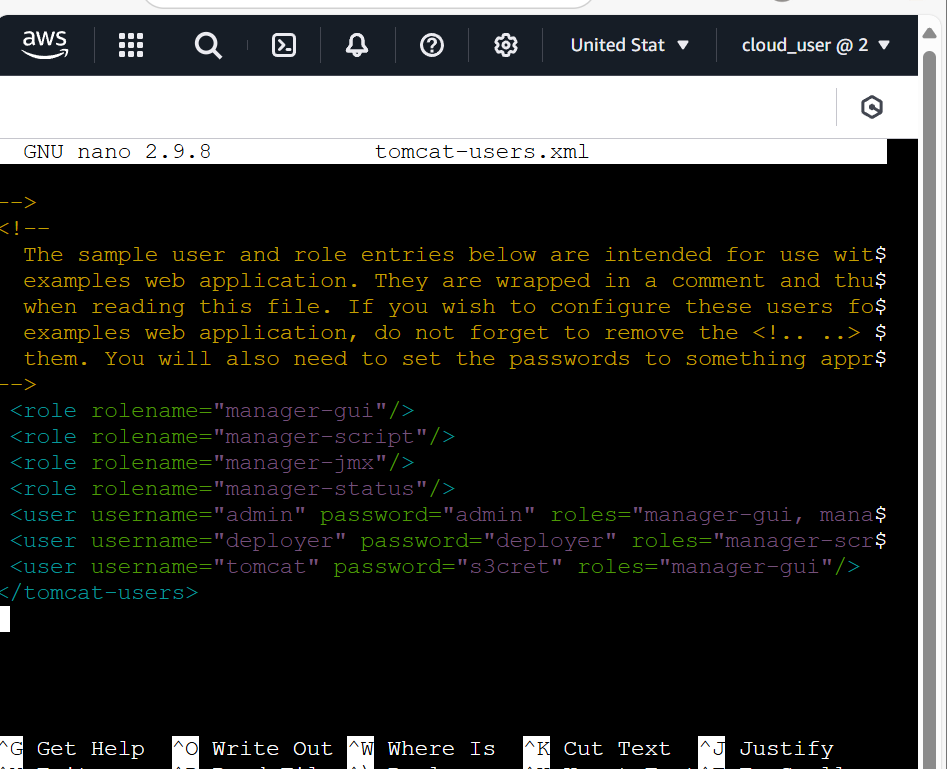
<user username="admin" password="admin" roles="manager-gui, manager-script, manager-jmx, manager-status"/>

<user username="deployer" password="deployer" roles="manager-script"/>

<user username="tomcat" password="s3cret" roles="manager-gui"/>

* **To start or shutdown tomcat from anywhere update the command:**
  + ln -s /opt/tomcat/ bin/startup.sh /usr/local/bin/tomcatup
  + ln -s /opt/tomcat/bin/shutdown.sh/usr/local/bin/tomcatdown
  + tomcatup // To start the tomcat
  + tomcatdown // To stop the tomcat
* **Configure Tomcat with Jenkins:**
  + Manage Jenkins → Plugins → Available Plugins → Deploy to Container
  + Manage Jenkins → Credentials → System → Global credentials
  + Add the username: deployer
  + Add the password: deployer
* **After configuring Tomcat in Jenkins Build the pipeline:**
* **Create a New Jenkins Job**  
  Give your job a name and select "Maven project" when setting it up.
* **Connect to GitHub**  
  Add the GitHub repository URL where your pom.xml file is stored. This is where Jenkins will pull the code from.
* **Set Maven Build Goals**  
  Use clean install as the Maven goal. This will clean up old files and build everything fresh.
* **Configure Post-Build Steps**  
  Add any necessary credentials and specify the path to the .war file that gets created after the build.
* **Run the Build**  
  Start the build process. Jenkins will:
  + Fetch your code from GitHub
  + Use Maven to build the project
  + Deploy the generated .war file to your Tomcat server
  + Let you access your web application in the browser





A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Deploying with Docker, Tomcat, and Jenkins

In this setup, we use **Docker** to run **Tomcat** inside a container. This makes our Tomcat server easy to move and run on any system without worrying about setup issues.

**By putting Tomcat in a Docker container:**

* Everything it needs is packaged together (like Java, config files, etc.)
* It runs the same way everywhere — on your laptop, a test server, or in production
* There are fewer errors caused by missing software or version mismatches

**When combined with Jenkins, we can automate the entire process:**

1. Jenkins builds the project.
2. The .war file is created.
3. Jenkins copies the .war file into the Dockerized Tomcat container.
4. Your app is automatically deployed and ready to use.

* **Installation of plugins on Jenkins:**

Ensure Jenkins has the following plugins installed to support Docker-based deployment:

* + SSH Server
  + Maven Integration
  + Deploy to container
  + Publish Over SSH
* **Docker Installation on EC2:**

Start by updating the Amazon Linux OS and installing Docker:

• sudo yum update -y

• sudo amazon-linux-extras install docker

• sudo yum install -y docker

Start Docker and verify it's running:

• sudo service docker start

• docker ps

* **Add a user in Docker instance and provide all the permissions:**
  + sudo useradd dockeradmin
  + sudo passwd dockeradmin

Adding to wheel group so that we can access sudo commands

* + usermod -aG wheel dockeradmin

Adding to dokcer to access the docker command

* + sudo usermod -aG docker dockeradmin
* **Enabling PasswordAuthotication:**

This will help us to access dockeradmin through Jenkins without SSH keys

* + nano /etc/ssh/sshd\_config
  + service sshd reload
* **Dockerfile Creation:**

This Dockerfile uses the official Tomcat image, copies the default webapps directory, and adds the compiled WAR file from your Maven project.

* cd /opt/
* mkdir docker
* cd docker/
  + sudo nano Dockerfile

// Content

FROM tomcat:latest

RUN cp -R /usr/local/tomcat/webapps.dist/\* /usr/local/tomcat/webapps

COPY ./\*.war /usr/local/tomcat/webapps

* **Provide the necessary permission to Docker directory and Dockerfile:**
  + chown -R dockeradmin:dockeradmin docker
  + chown -R dockeradmin:dockeradmin Dockerfile

// This will help in creating the docker container through Jenkins pipeline

* **Jenkins Configuration for docker:**
  + Manage Jenkins → Configure System → Publish Over SSH → SSH Server

// Pass the Credentials like   
// Name: dockerhost

// Hostname: Instance’s Public\_IP

//Username: dockeradmin

// In Advanced Section click on **Password** and give dockeradmin password

* **Build a new pipeline:**
  + **Create a New Job**Start by opening Jenkins and creating a new job. Give it any name you like and select "Maven project" as the job type. This tells Jenkins that the project will use Maven for building.
  + **Connect to GitHub**In the configuration, go to the Source Code Management section. Choose Git, and enter the GitHub repository URL where your project is hosted. Make sure this repository contains the pom.xml file, which Maven uses to build the project**.**
  + **Set Maven Build Goal**

Set the goal to **clean install**, ensuring a fresh build process.

* + **Post-Build Actions**

Select **Send Build artifacts over SSH** and provide all credentials like **Name, Source file, Remove Prefix, Remote Directory**.

* + **Execute Command:** In Exec Command section provide below code

cd /opt/docker;

docker build -t regapp:v1 .;

docker stop registerapp; (if running)

docker rm registerapp;

docker run -d --name registerapp -p 8085:8080 regapp:v1

**Here's what it does:**

Goes into the Docker folder.

Builds a Docker image named regapp:v1.

Stops and removes the old container if it exists.

Runs a new container exposing the app on port 8085

* + **Trigger the build**

Save the job and click **“Build Now”**. Jenkins will:

Clone your code from GitHub.

Build it using Maven.

Transfer the .war file to the remote server.

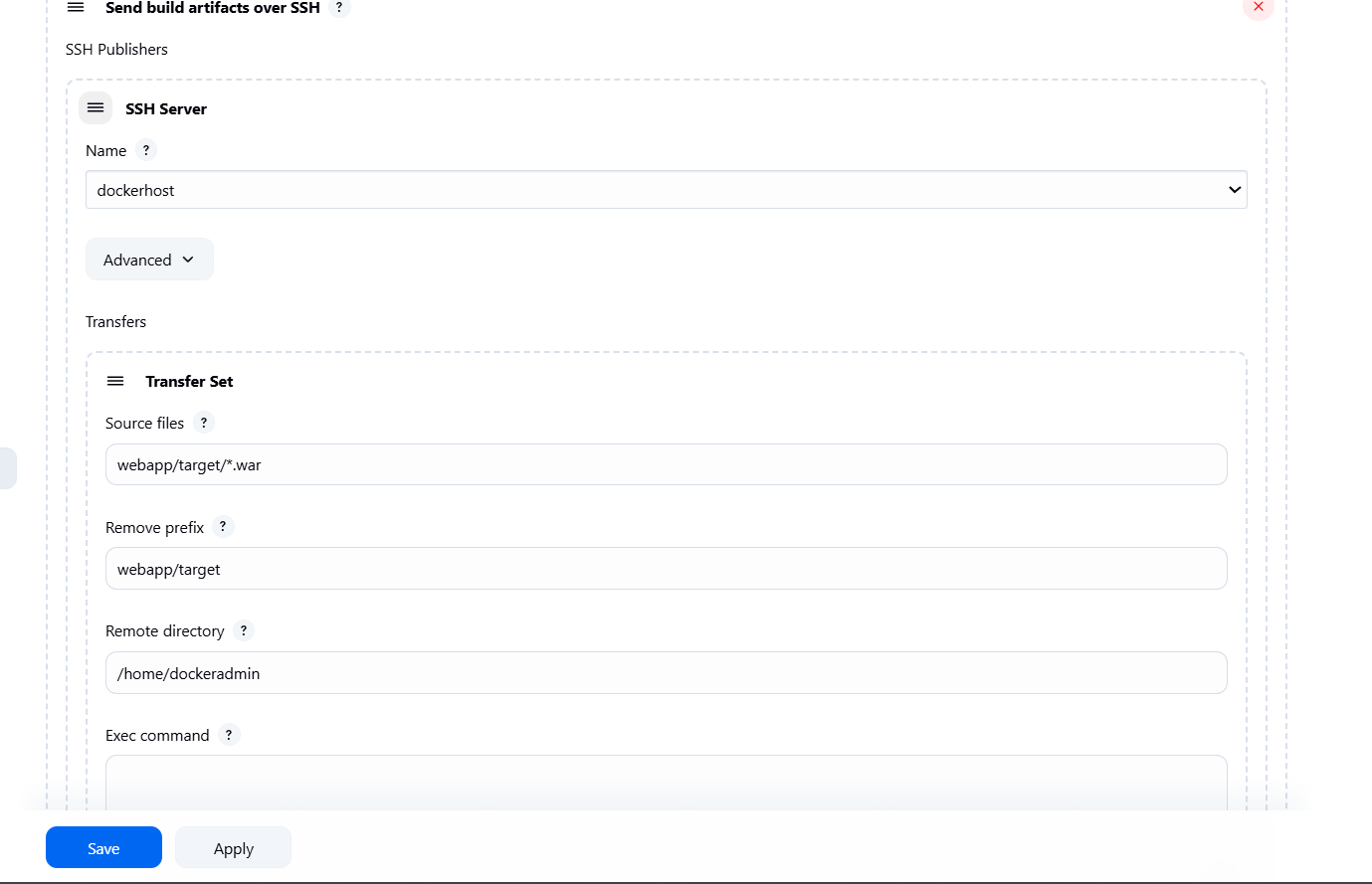
Use Docker to deploy your Java application inside a Tomcat container.

.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. 

## Ansible, Docker, and Jenkins Deployment

This strategy introduces configuration management using Ansible. Ansible automates the deployment process on remote Docker hosts, making it scalable, repeatable, and efficient. We use Jenkins to trigger Ansible playbooks that manage Docker image creation, tagging, and container deployment.

Below is step by step implementation of this:

* **Set Hostname and Create Ansible Admin User (on Ansible Server):**

**//** Here we are renaming the hostname to ansible-server because after that it’s easy to recognize which instance is doing what.

* + nano /etc/hostname.

// Set to:

// ansible-server

* + init 6 // restart the instance
* **Now adding a user in Ansible-server and providing all permissions:**
  + useradd ansadmin
  + passwd ansadmin
  + usermod -aG wheel ansadmin
  + id ansadmin
* **Configure SSH:**
  + nano /etc/ssh/sshd\_config

Enable PasswordAuthentication if needed

* + service sshd reload
* **Installation of Ansible in OS:**
  + sudo yum update -y
  + sudo amazon-linux-extras enable ansible2
  + sudo yum install -y ansible
  + ansible –version
* **Generate SSH Key for Ansible (ansadmin):**
  + ssh-keygen
* **Prepare Directory for Docker Files (on Ansible Server):**
  + sudo mkdir /opt/docker && cd /opt
  + sudo chown -R ansadmin:ansadmin docker

Providing admin permission to docker directory

* **Repeat Admin Setup on Docker Host for adding user (ansadmin):**
  + Creating new use ansadmin on Docker-server Instance
  + Providing all admin permissions like adding wheel and other groups
* **Add Docker Host IP in Ansible Inventory:**
  + sudo nano /etc/ansible/hosts

# Add this:

[dockerhost]

<DOCKER\_HOST\_IP> // provide private\_ip here

* **Copy SSH Key to Docker Host:**
  + ssh-copy-id ansadmin@<DOCKER\_HOST\_IP>

If SSH key doesn’t work, use:

* + sudo yum install -y sshpass
* **Check weather connection established:**
  + ansible all -m ping

if above doesn’t work try this:

ansible all -m ping -i /etc/ansible/hosts -u ansadmin -k

* + ansible all -m command -a "uptime"

if above doesn’t work try this:

ansible all -m command -a "uptime" -i /etc/ansible/hosts -u ansadmin -k

* **Now install Docker in ansible-server:**

// Start by updating the Amazon Linux OS and installing Docker:

* + • sudo yum update -y
  + • sudo amazon-linux-extras install docker
  + • sudo yum install -y docker

// Start Docker and verify it's running:

* + • sudo service docker start
  + • docker ps
* **Create a Docker directory And provide necessary permissions(on Ansible Server):**
  + cd /opt/
  + mkdir docker/
  + nano Dockerfile

// Content

FROM tomcat:latest

RUN cp -R /usr/local/tomcat/webapps.dist/\* /usr/local/tomcat/webapps

COPY ./\*.war /usr/local/tomcat/webapps

* sudo chmod 777 /var/run/docker.sock
* **Build the images from the Dockerfile:**
* docker build -t regapp:v1 .
* docker run -t --name regapp-server -p 8081:8080 regapp:v1
* **Update Ansible Hosts File:**
* sudo nano /etc/ansible/hosts
* [dockerhost]
* 172.31.87.87
* [ansible]
* 172.31.86.207
* **Create Ansible Playbook (regapp.yml):**
* Sudo nano regapp.yml

---

- hosts: ansible

become: true

tasks:

- name: Build Docker image

command: docker build -t regapp:latest .

args:

chdir: /opt/docker

- name: Tag Docker image

command: docker tag regapp:latest avinah16/regapp:latest

- name: Push Docker image

command: docker push avinah16/regapp:latest

* **Grant Ansible Admin Permission to Run Without Password:**

//This will help in running yml files from Jenkins

* + sudo visudo
  + # Add:
  + ansadmin ALL=(ALL) NOPASSWD: ALL
* **Ensure Ownership and Run Playbook:**
  + sudo chown -R ansadmin:ansadmin regapp.yml
  + ansible-playbook regapp.yml --check
  + ansible-playbook regapp.yml
  + ansible-playbook regapp.yml --limit

// if above doesn’t work then:

* + ansible-playbook regapp.yml --check -i /etc/ansible/hosts -u ansadmin -k
  + ansible-playbook regapp.yml -i /etc/ansible/hosts -u ansadmin -k
  + ansible-playbook regapp.yml --limit 172.31.86.207 -i /etc/ansible/hosts -u ansadmin -k
* **DockerHub Upload (Optional):**
  + docker login
  + # Enter credentials
  + docker tag regapp:latest avinah16/regapp:latest
  + docker push avinah16/regapp:latest
* **Create Second Playbook for Deployment (deploy\_regapp.yml):**
  + **nano deploy\_regapp.yml**

---

- hosts: dockerhost

become: true

tasks:

- name: Stop container

command: docker stop regapp-server

ignore\_errors: yes

- name: Remove container

command: docker rm regapp-server

ignore\_errors: yes

- name: Remove image

command: docker rmi avinah16/regapp

ignore\_errors: yes

- name: Deploy container

command: docker run -d --name regapp-server -p 8082:8080 avinah16/regapp:latest

* **Fix Socket Permission Issue (on Docker Host):**
  + chmod 777 /var/run/docker.sock
* **Now create a pipeline to automate the image creation process:**
  + **Create a new job** – Name it as desired and set it up as a **Maven project**
  + **Link to GitHub** – Provide the repository details where the pom.xml (Project Object Model) file is located.
  + **Define the Maven goal** – Set the goal to **clean install**, ensuring a fresh build process.
  + **Post-Build Actions –** Select **Transfer Set** and provide all credentials like **Source file, Remove Prefix, Remote Directory**.
  + **Execute Command –** In Exec Command section provide below code to push the docker image and run the container

**ansible-playbook /opt/docker/regapp.yml;**

**sleep 10;**

**ansible-playbook /opt/docker/deploy\_regapp.yml**

* **Trigger the build** – **Jenkins** will automatically fetch the latest code, trigger **Maven** to build the project, and then utilize **Ansible** to execute the required **YML** files. These files will create a **Docker image**, which will then be pushed to **DockerHub**. Finally, a **Docker container** will be generated from this image, allowing seamless access to the **web application**.

A screenshot of a computer

AI-generated content may be incorrect.

-;A screenshot of a computer

AI-generated content may be incorrect.A black screen with white lines

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**Ansible, Docker, Kubernetes, and Jenkins Deployment:**

This final strategy seamlessly integrates **Kubernetes** for powerful container orchestration, ensuring efficient scaling and resilience. With **AWS EKS**, you gain a managed Kubernetes environment that simplifies operations while maintaining flexibility. **Ansible** takes charge of automation, enforcing consistency and reducing manual overhead. **Jenkins** acts as the driving force behind CI/CD, orchestrating smooth deployments. Meanwhile, **Docker** encapsulates the application into lightweight, portable containers, streamlining distribution and execution.

* **Install AWS CLI:**
  + curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"
  + unzip awscliv2.zip
  + sudo ./aws/install
  + aws configure
* **Install kubectl:**
  + curl -LO "https://dl.k8s.io/release/v1.30.1/bin/linux/amd64/kubectl"
  + chmod +x kubectl
  + sudo mv kubectl /usr/local/bin/
  + kubectl version --client
  + kubectl get nodes
* **Install eksctl:**
* ARCH=amd64
* PLATFORM=$(uname -s)\_$ARCH
* curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$PLATFORM.tar.gz"
* curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_checksums.txt" | grep $PLATFORM | sha256sum --check
* tar -xzf eksctl\_$PLATFORM.tar.gz -C /tmp && rm eksctl\_$PLATFORM.tar.gz
* sudo mv /tmp/eksctl /usr/local/bin
* **Set Up IAM Role in AWS**
  + Go to AWS Console → IAM → Roles → Create new Role
  + Choose EC2 use case after select first option
  + Attach policies:
    - IAMFullAccess
    - AWSCloudFormationFullAccess
    - AmazonEC2FullAccess
    - AdministratorAccess (optional)
  + Name the role “eksctl\_role” and create role
  + Go to EC2 dashboard → Select server → Actions → Security → Modify Role
  + Attach eksctl\_role
* **Create EKS Cluster:**
  + eksctl create cluster \
  + --name xellian \
  + --region us-east-1
  + Check progress in AWS Console → CloudFormation
* **Interact with Cluster Using kubectl:**
  + kubectl get pods
  + kubectl get all # optional
* **Run Sample Web Application (Apache):**
  + kubectl run webapp --image=httpd
* **Run Nginx Deployment:**

Here we are creating the pod with 2 replicas and it on port 80

* + kubectl create deployment demo-nginx --image=nginx --port=80 --replicas=2
  + kubectl expose deployment demo-nginx --port=80 --type=LoadBalancer
  + kubectl get all
* **Go to EC2 dashboard → Scroll down to Load Balancers → Find external IP or DNS to access application**
* **Integration K8s with Ansible and Jenkins:**
* **Create Admin User on Kubernetes Node(ansadmin):**
  + useradd ansadmin
  + passwd ansadmin

provide all permissions

* **Configure SSH:**
  + nano /etc/ssh/sshd\_config

// Enable PasswordAuthentication if needed

* + service sshd reload
* **Rename Playbook Files (on Ansible Server):**
  + mv regapp.yml create\_image\_regapp.yml
  + mv deploy\_regapp.yml docker\_deployment.yml

// renaming existing yml files

* **Writing Ansible Host files:**
* sudo nano /etc/ansible/hosts/

// Add below content in file change the IP’s

[kubernetes]

172.31.88.173

[ansible]

172.31.80.130

* **Copy SSH Key to Kubernetes Node:**
  + ssh-copy-id root@<K8S\_PRIVATE\_IP>
* **Verify Access:**
  + ansible -i hosts all -a uptime
* **Create Kubernetes Playbooks (on Ansible Server):**
  + kube-deploy.yml
  + kube-service.yml

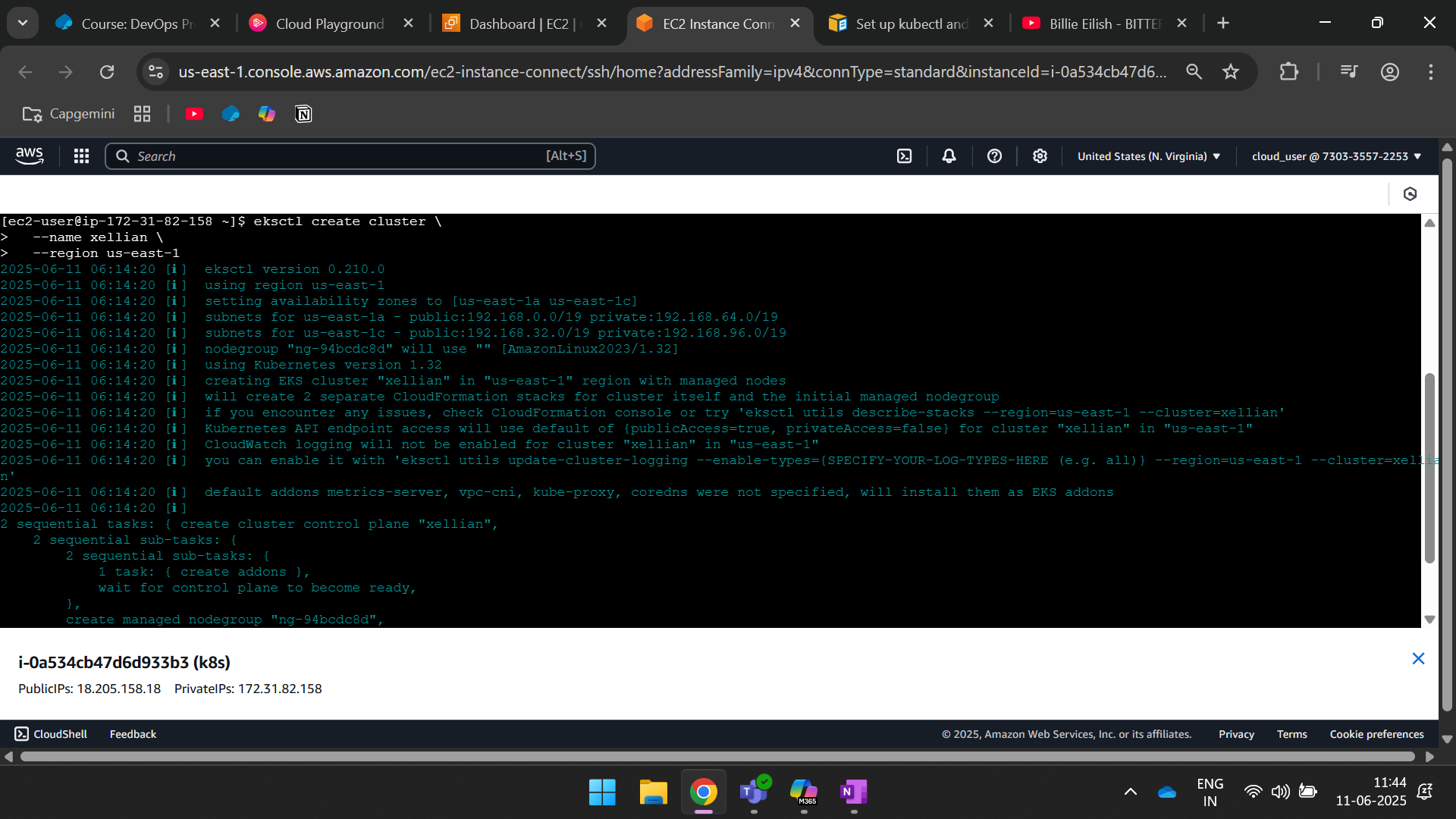
// Take deployment and service YAML from Git repo under k8s folder. Ensure the Kubernetes node has these YAMLs in its home directory.

* Run these files:
  + ansible-playbook -i /opt/docker/hosts kube\_deploy.yml
* **Jenkins Configuration:**
* **Create Freestyle project cd-job**
* **Execute shell:**
  + ansible-playbook -i /opt/docker/hosts /opt/docker/kube\_deploy.yml
* Set up a new **Freestyle project** named ci-job to manage the continuous integration process effectively.
* Incorporate a **build step** to execute the required artifact command, ensuring smooth automation and efficient workflow.
* Configure a **Post-Build Action** to trigger the cd-job using the **"Build other projects"** option, enabling seamless deployment

This completes a full CI/CD pipeline using Ansible, Docker, Kubernetes, and Jenkins with deployment into an AWS EKS cluster.

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion:**

This project gave us great hands-on experience with how to deploy Java applications using modern DevOps tools. Step by step, we moved from simple builds using Maven to more advanced automation with Ansible, Docker, Kubernetes, and Jenkins. Through this, we learned how to make the software development and deployment process faster, easier, and more reliable.

**What We Learned:**

* How to use **Maven** to build projects and manage software dependencies easily.
* How to set up **Jenkins** to automate the entire process of building and deploying code.
* How to create and manage **Docker containers**, which are used to run applications in a lightweight and consistent way.
* How to write and run **Ansible playbooks** to automatically manage and configure multiple servers.
* How to use **Kubernetes** to deploy and manage applications that need to scale and stay available.
* How to set up and use **AWS cloud services** as part of our deployment pipeline.

**Challenges Faced and Solutions:**

**1.Jenkins Installation Issues (Java Version & Low Storage)**One of the first problems we faced was while installing Jenkins. Jenkins requires a specific version of Java to run properly, and in our case, the system had an incompatible Java version, which caused Jenkins to crash or not start correctly. On top of that, the server had very low disk space, which caused installation failures and prevented Jenkins from running smoothly. We fixed this by upgrading Java to the correct version (Java 11 or 17) and increasing the disk space on the server.

**2. EKS Cluster Creation Issues**The next major issue was with creating the EKS (Elastic Kubernetes Service) cluster on AWS. The eksctl command would run, but the cluster was not being created properly, and the setup was not functional. We found that this was due to incorrect IAM permissions and missing configuration values in the YAML file. To solve it, we double-checked our AWS credentials, ensured all the necessary IAM roles and policies were set up, and reconfigured the eksctl YAML file with the correct settings.