Case Study Report:

DevOps CI/CD Pipeline Implementation Using Git, Jenkins, Maven, Docker, Ansible, and Kubernetes

1. Introduction

1.1 Overview of DevOps, Continuous Integration (CI), and Continuous Delivery/Deployment (CD)

DevOps
DevOps is a modern software engineering approach that emphasizes collaboration between development and operations teams. Its primary objective is to streamline the software development lifecycle by promoting automation, continuous monitoring, and faster delivery of high-quality software. DevOps fosters a culture of shared responsibility, iterative improvement, and operational excellence.

Continuous Integration (CI)
Continuous Integration refers to the practice where developers frequently commit code changes to a shared repository. Each commit triggers an automated build and testing process, enabling early detection of integration issues. CI helps maintain code stability and ensures that new changes work well with the existing codebase.

Continuous Delivery and Continuous Deployment (CD)

- Continuous Delivery ensures that applications are always in a deployable state. Validated code changes can be released to production at any time with minimal manual intervention.
- Continuous Deployment extends this concept by automating the entire release process. Every successful code change that passes all tests is automatically deployed to production, eliminating manual steps entirely.

1.2 Importance of CI/CD in Modern Software Development

In today's fast-paced and highly competitive digital landscape, rapid innovation and frequent feature releases are essential. CI/CD plays a vital role in enabling these capabilities by:

- Automating repetitive tasks such as building, testing, and deployment.
- Reducing integration challenges and shortening release cycles.
- Improving software quality through early bug detection and resolution.
- Enabling faster feedback loops for continuous improvement.

CI/CD is a fundamental component of Agile and DevOps methodologies and is critical for delivering scalable, reliable, and maintainable software solutions

1.3 Objective of the Case Study

This case study aims to:

- Design and implement a fully automated DevOps pipeline.
- Integrate industry-standard tools for source control, build automation, configuration management, containerization, and orchestration.
- Demonstrate end-to-end automation of the software delivery process—from code commit to deployment on a Kubernetes cluster.

1.4 Tools and Technologies Overview

Tool	Purpose
Git/GitHub	Source code version control and collaboration
Jenkins	Automation server for CI/CD pipeline execution
Maven	Build automation tool for Java-based projects
Tomcat	Web application server for deploying Java apps
Docker	Containerization platform for consistent environments
Ansible	Configuration management and automation
Kubernetes	Container orchestration platform for deployment and scaling

2. Scope of the Case Study

This case study serves as a comprehensive guide to understanding and implementing CI/CD pipelines within the DevOps lifecycle. Key areas of focus include:

2.1 Fundamentals of CI/CD in the DevOps Lifecycle

- **Continuous Integration (CI):** Explains how developers integrate their code frequently into a shared repository, triggering automated builds and tests to ensure smooth integration with the existing system.
- **Continuous Delivery and Deployment (CD):** Provides insights into the processes that automatically deliver or deploy validated code to staging or

production environments. The distinction between delivery (manual trigger) and deployment (fully automated) highlights different levels of pipeline automation.

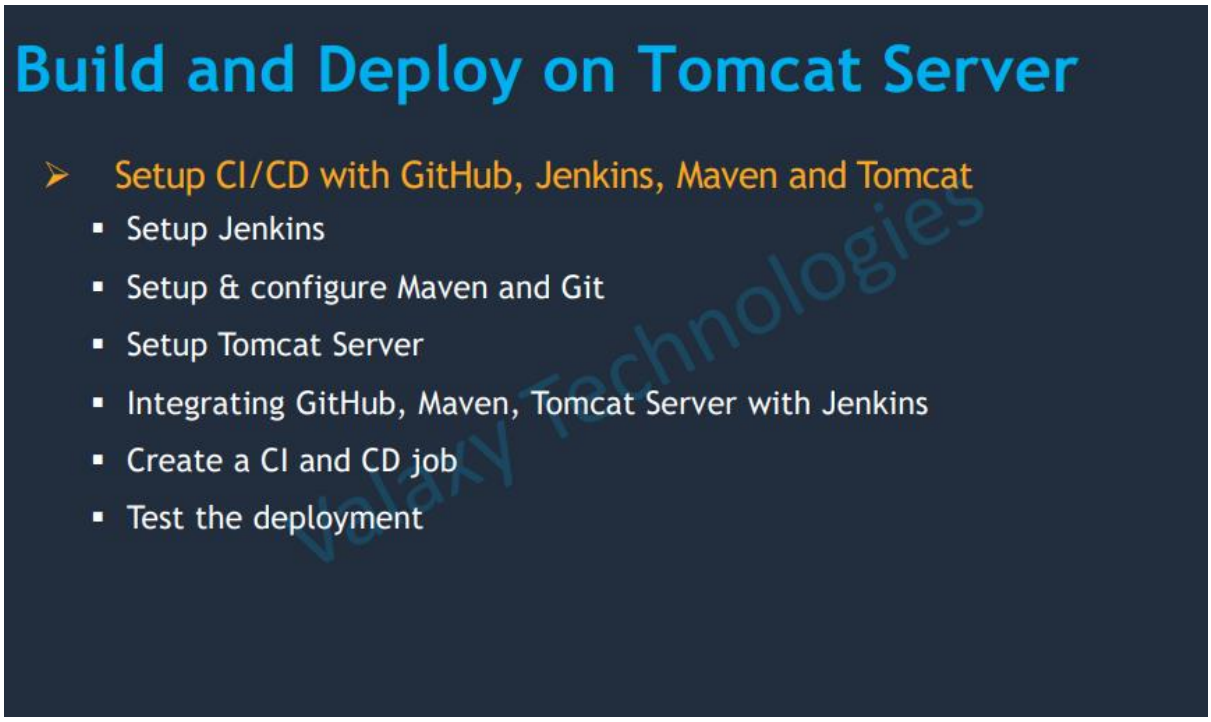
- **DevOps Lifecycle:** Describes the broader DevOps lifecycle, which includes planning, coding, building, testing, releasing, deploying, operating, and monitoring software systems.

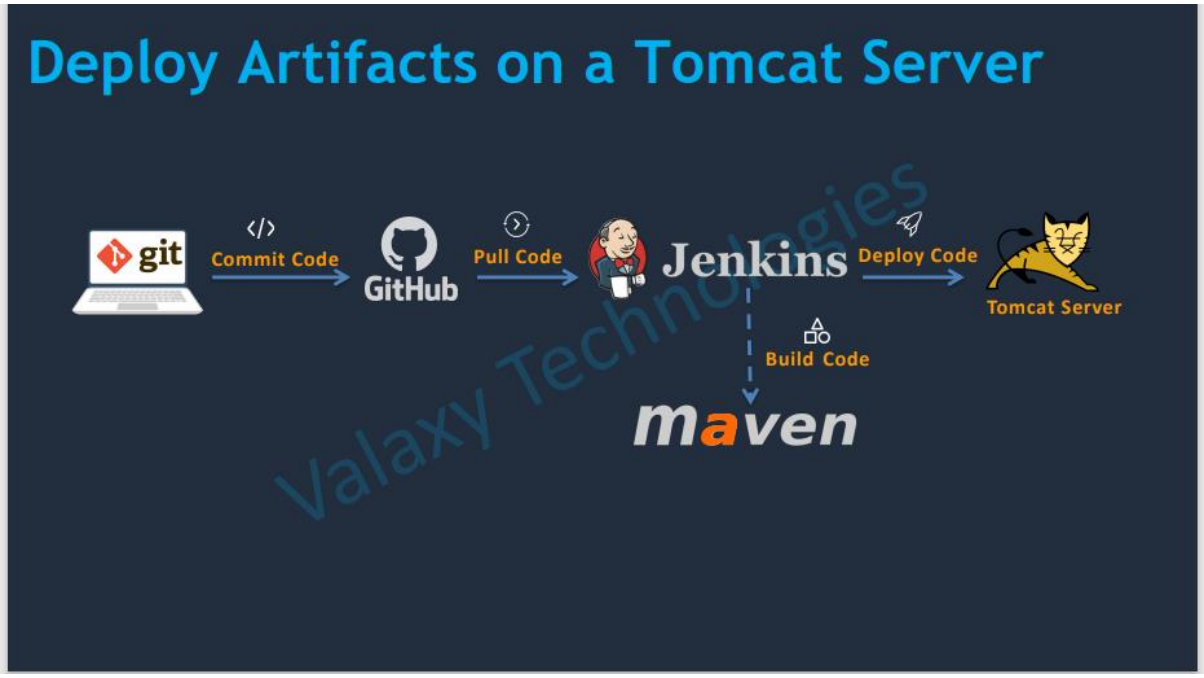
2.2 Required Tools and Resources

To implement an efficient CI/CD pipeline, the following tools and resources are introduced:

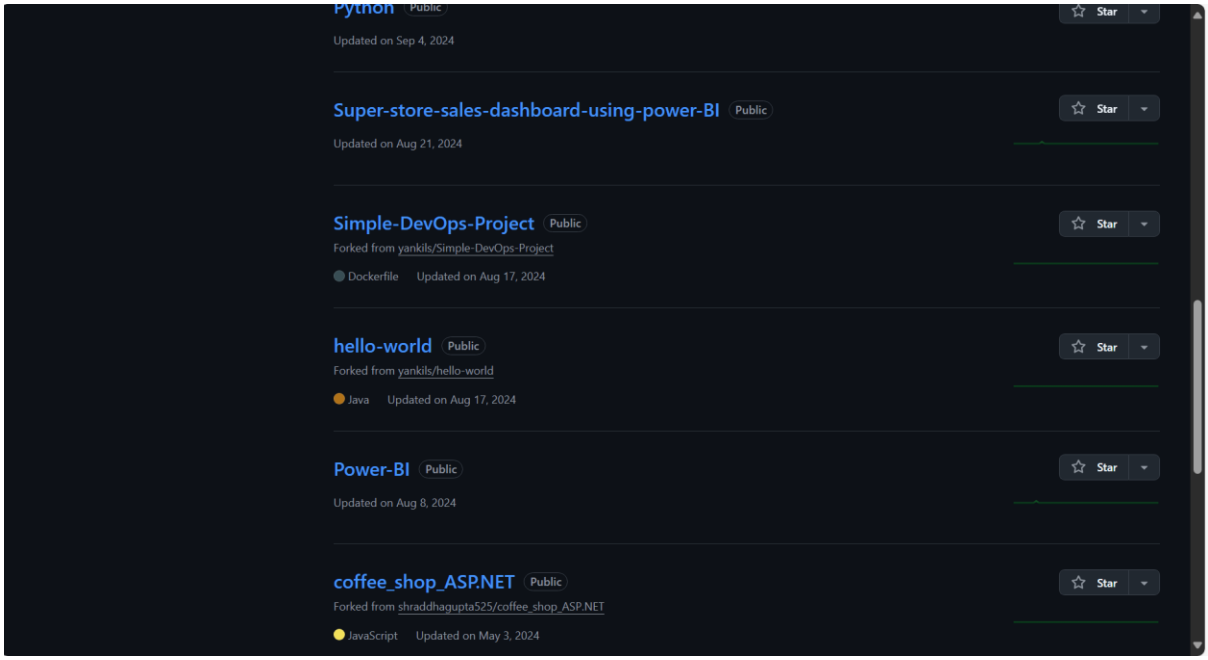
- **Version Control Systems:** Tools such as Git and platforms like GitHub for managing source code and collaboration.
- **Build Automation Tools:** Tools like Maven to automate compiling, testing, and packaging of software.
- **CI/CD Orchestration Tools:** Jenkins or similar tools to automate build, test, and deployment processes.
- **Containerization Tools:** Docker to create portable and consistent environments for development, testing, and production.
- **Configuration Management Tools:** Ansible to automate the setup and configuration of infrastructure and applications.
- **Container Orchestration Platforms:** Kubernetes to deploy, manage, and scale containerized applications efficiently.
- **Cloud Platforms:** AWS, Azure, or Google Cloud for hosting infrastructure and enabling dynamic scalability.

1. Deploying Java Application in Tomcat server

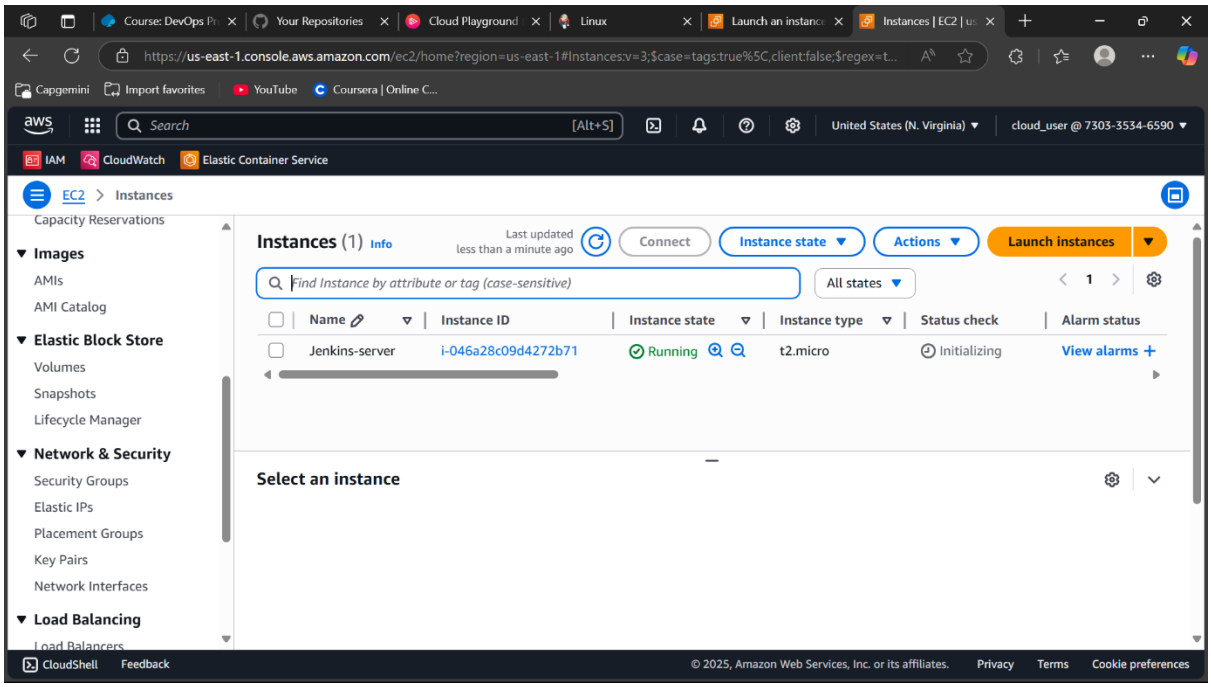




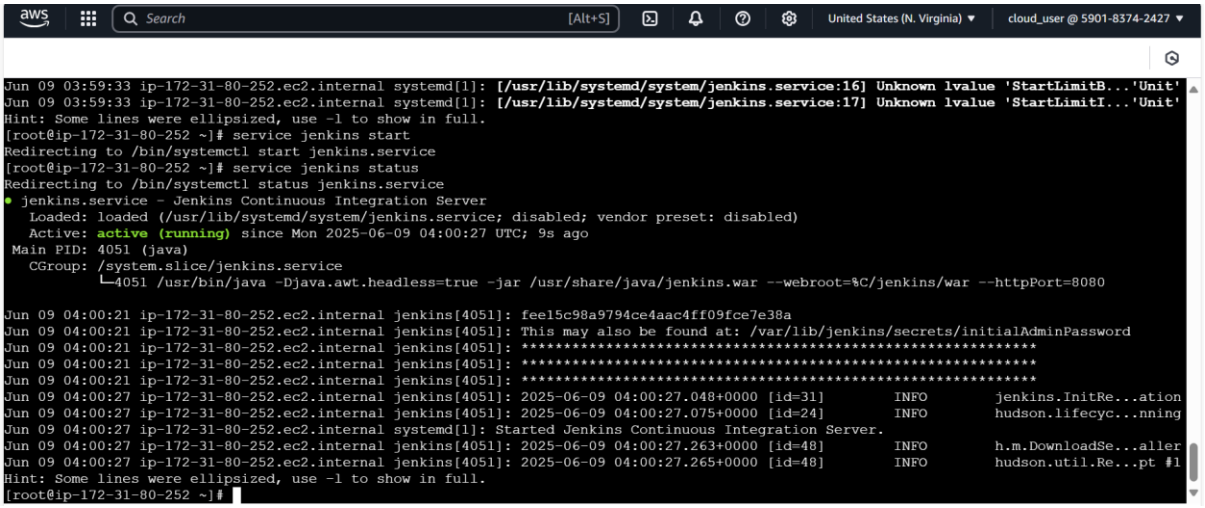
- Technologies Used - Git - Jenkins - Maven - Tomcat Server
- Forked the java application named hello-world in my github account

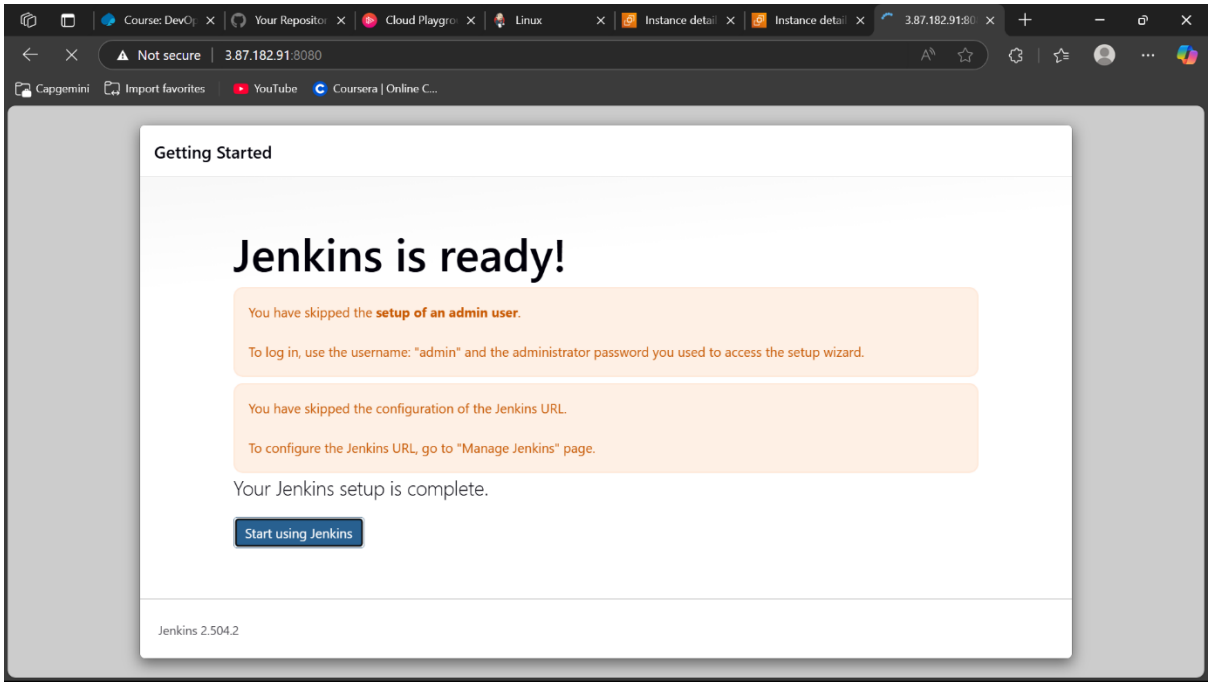


- Created an ec2 instance named Jenkins server with the configuration and security settings

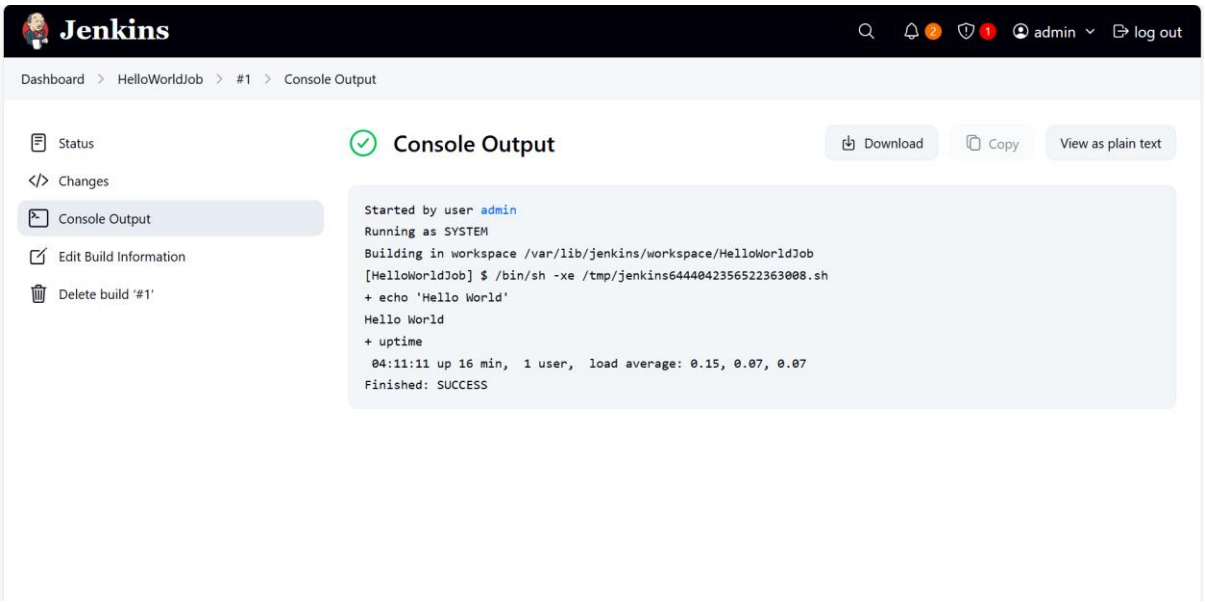


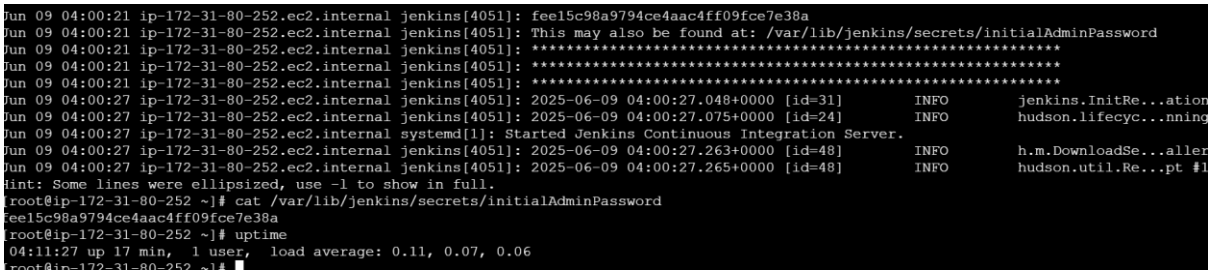
- Installed java and Jenkins in the Jenkins server instance



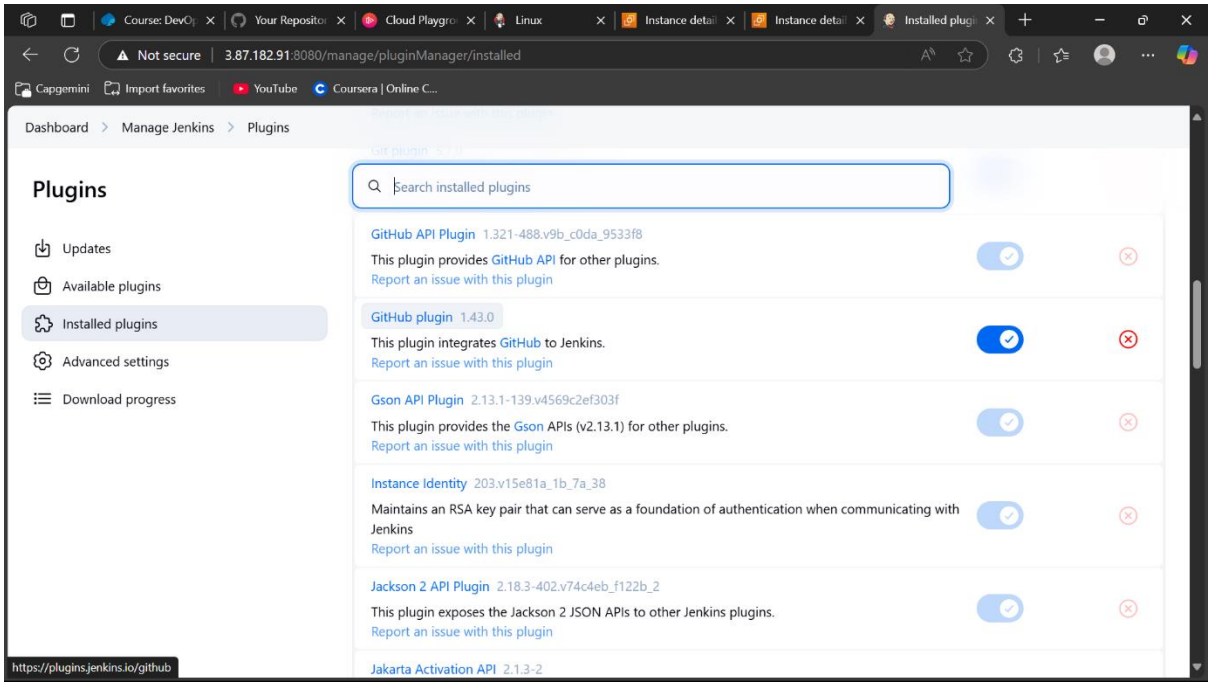


- Created my first Jenkins job named Helloworldjob that printed “Hello World”.Created this job to check whether the Jenkins setup was working properly or not

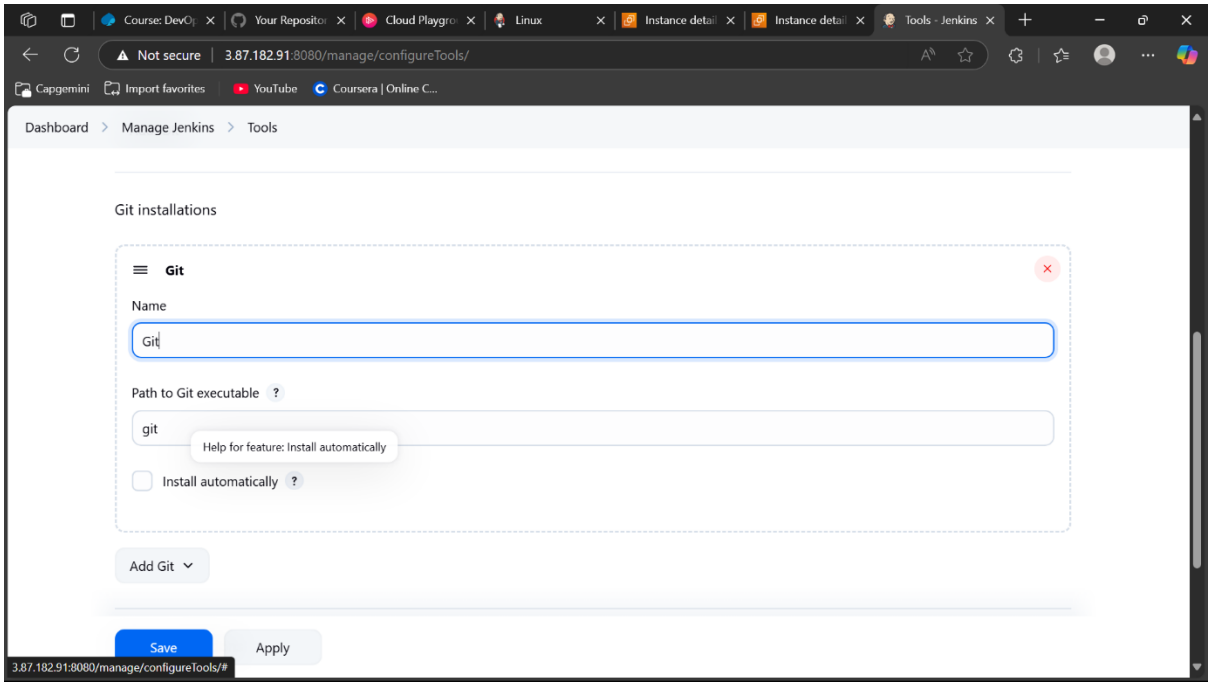




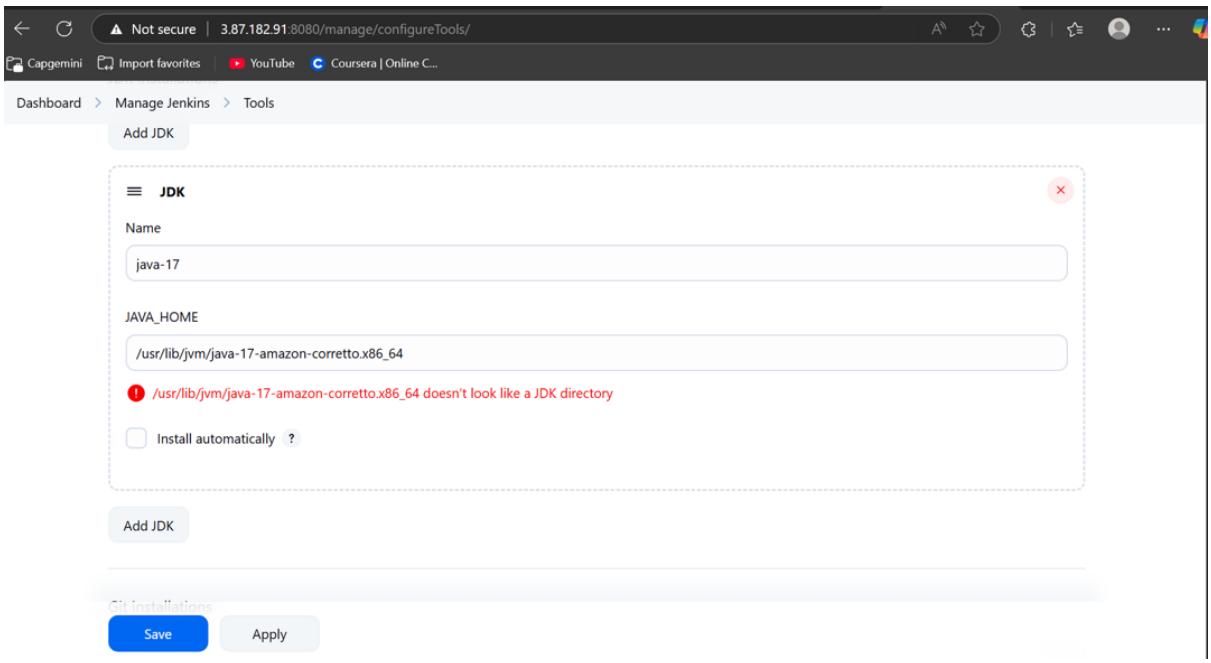
- Later I installed github plugin to pull the code from github repository to jenkins



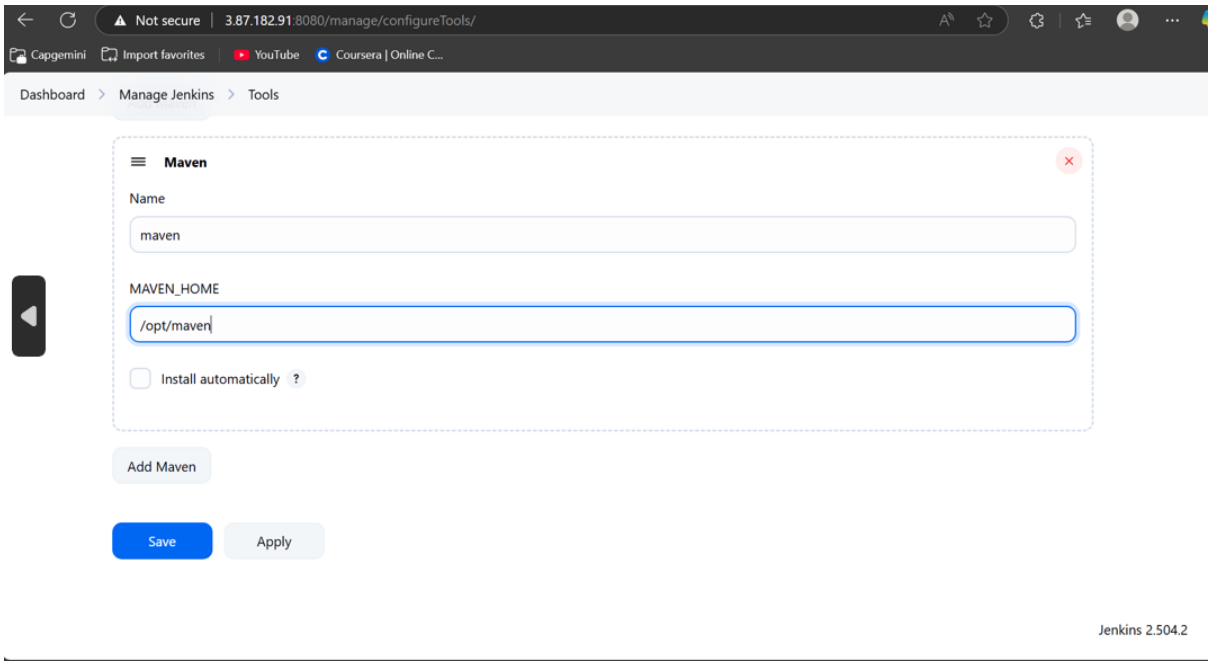
- And configured it accordingly.



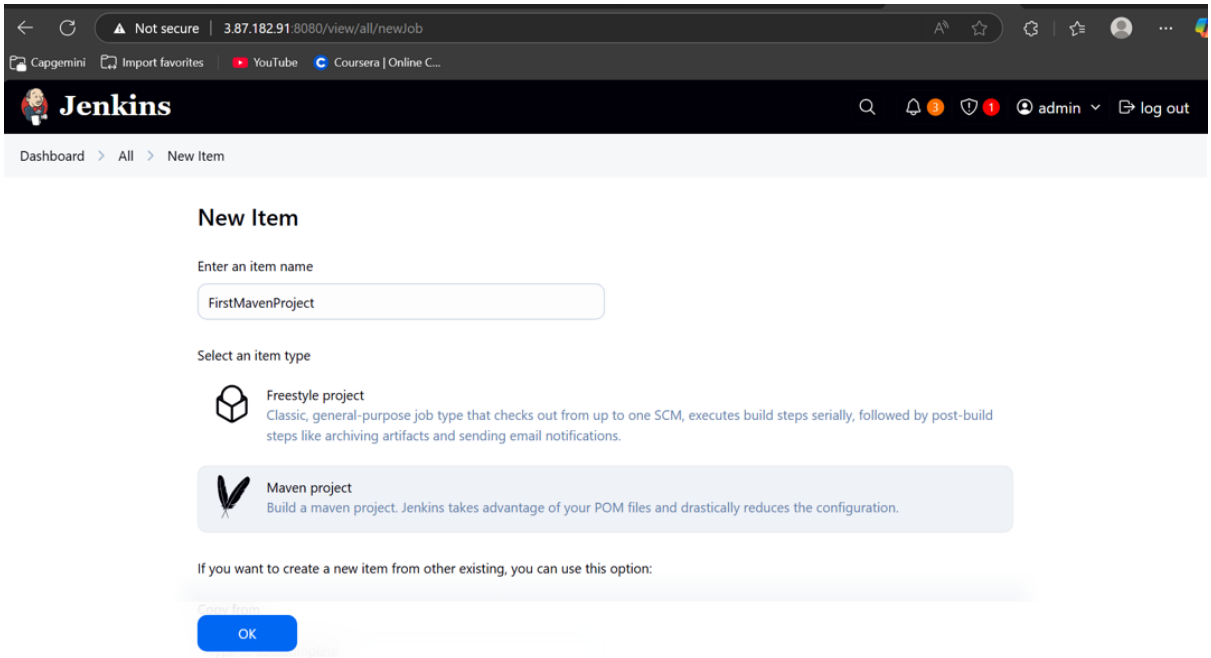
- Configured maven and java in the configure.xml file in the Jenkins server on aws and installed the plugins named maven in jenkins

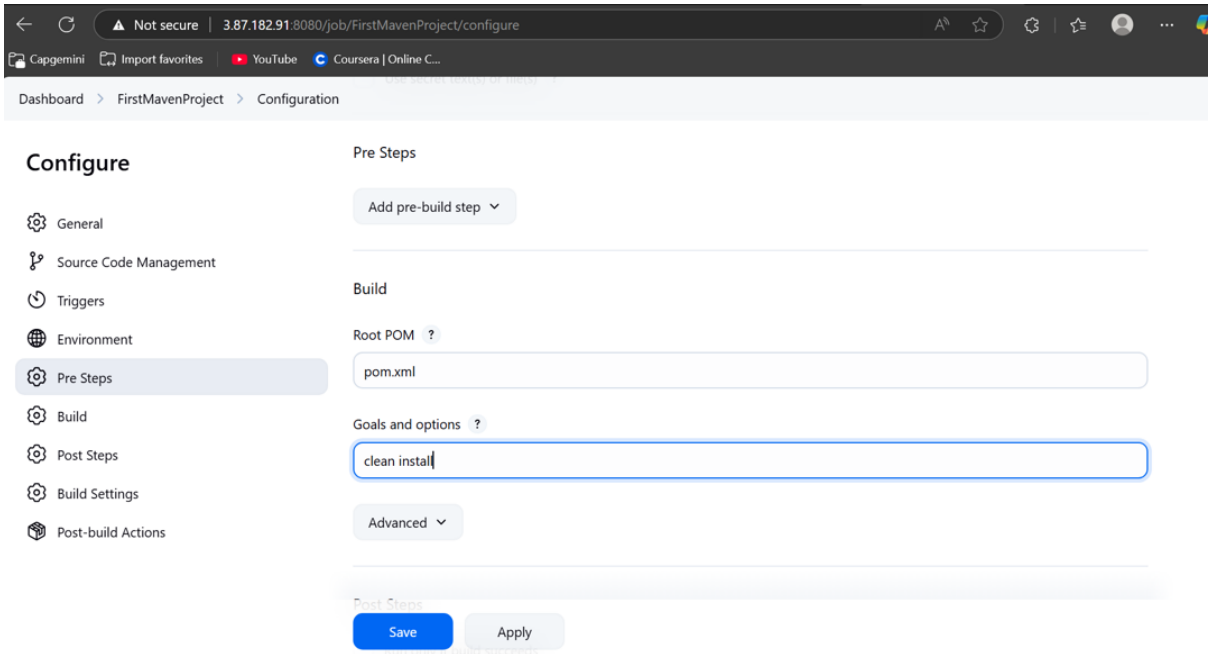


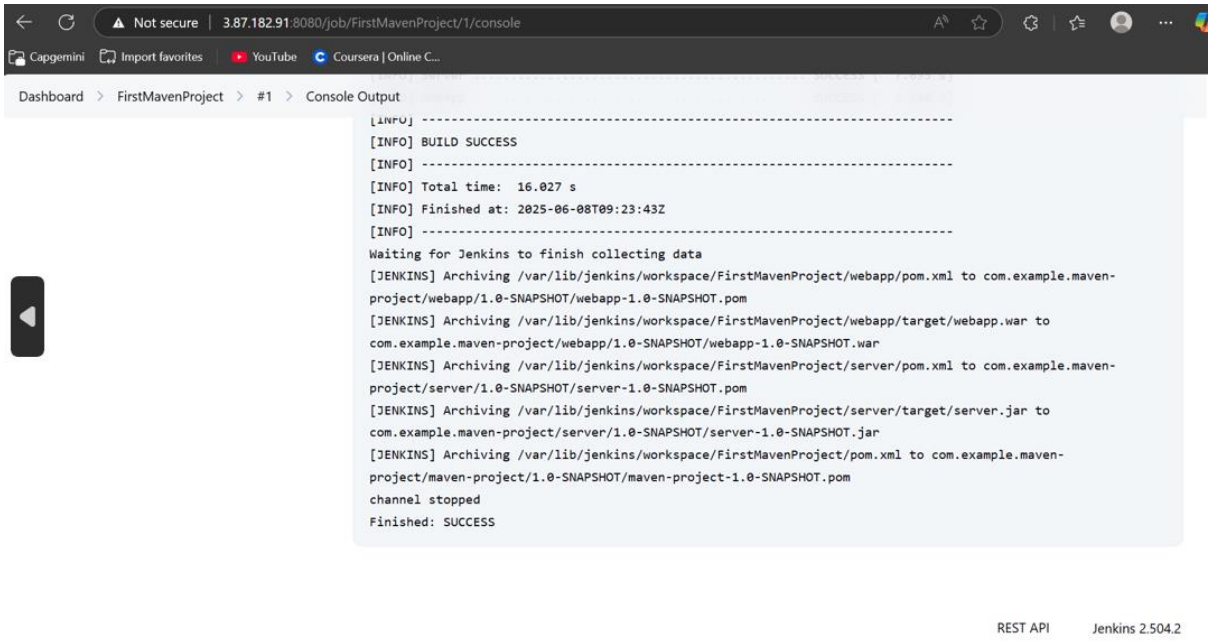
- Configured them accordingly in the jenkins

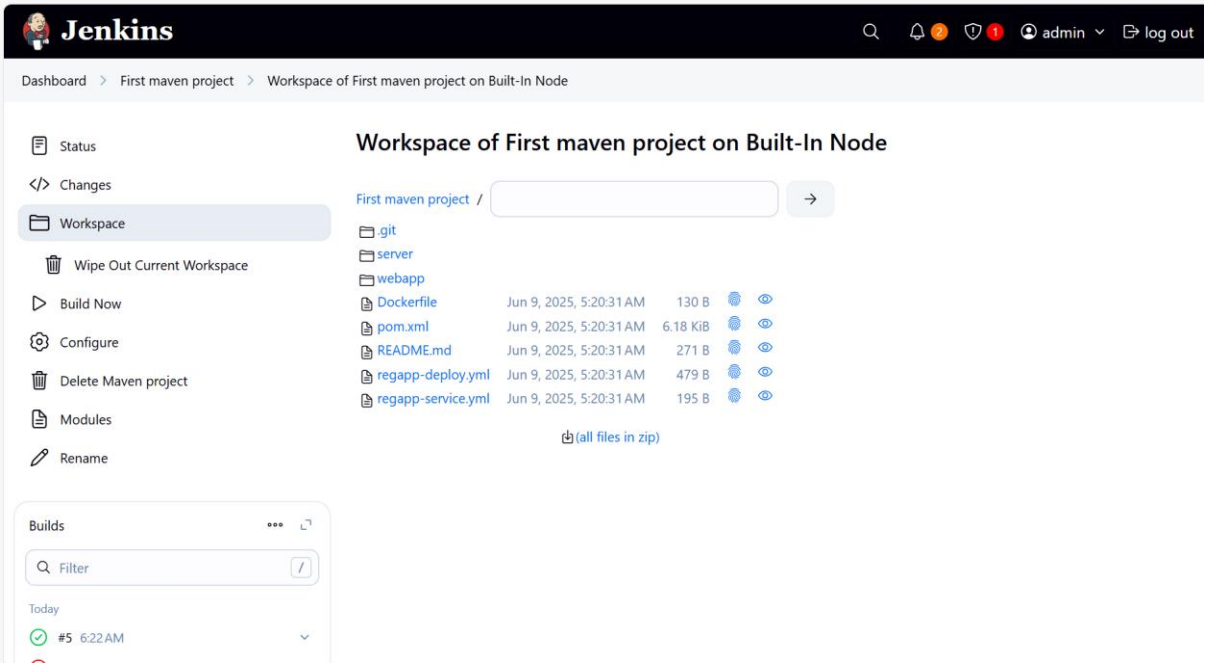


- Later created my first maven project to pull the code from github and build the code using maven





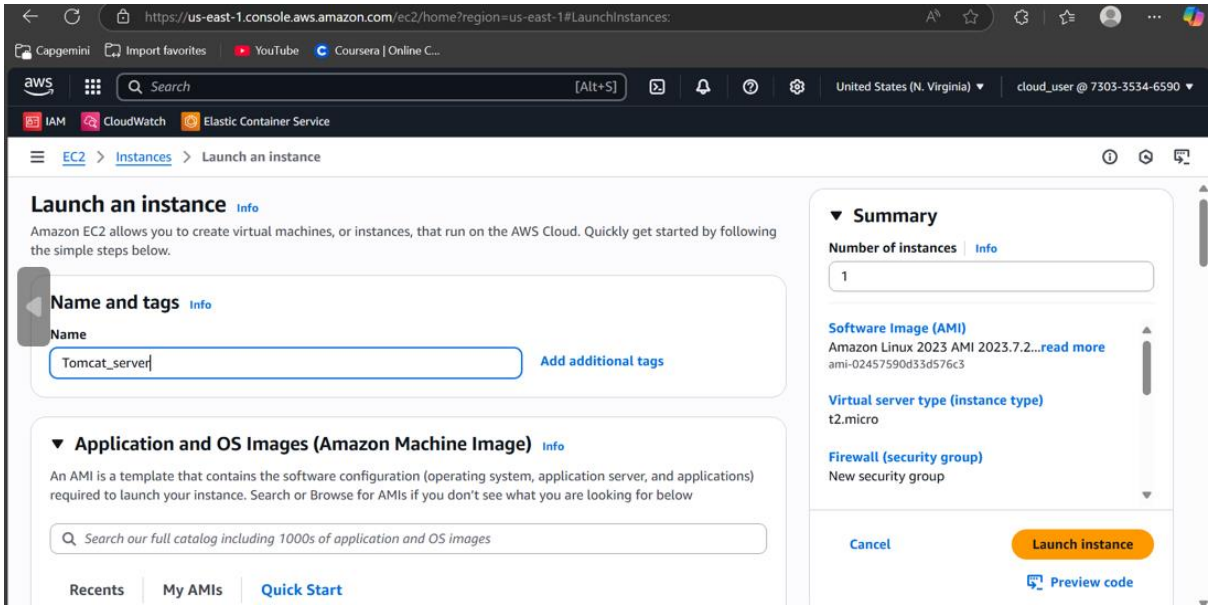


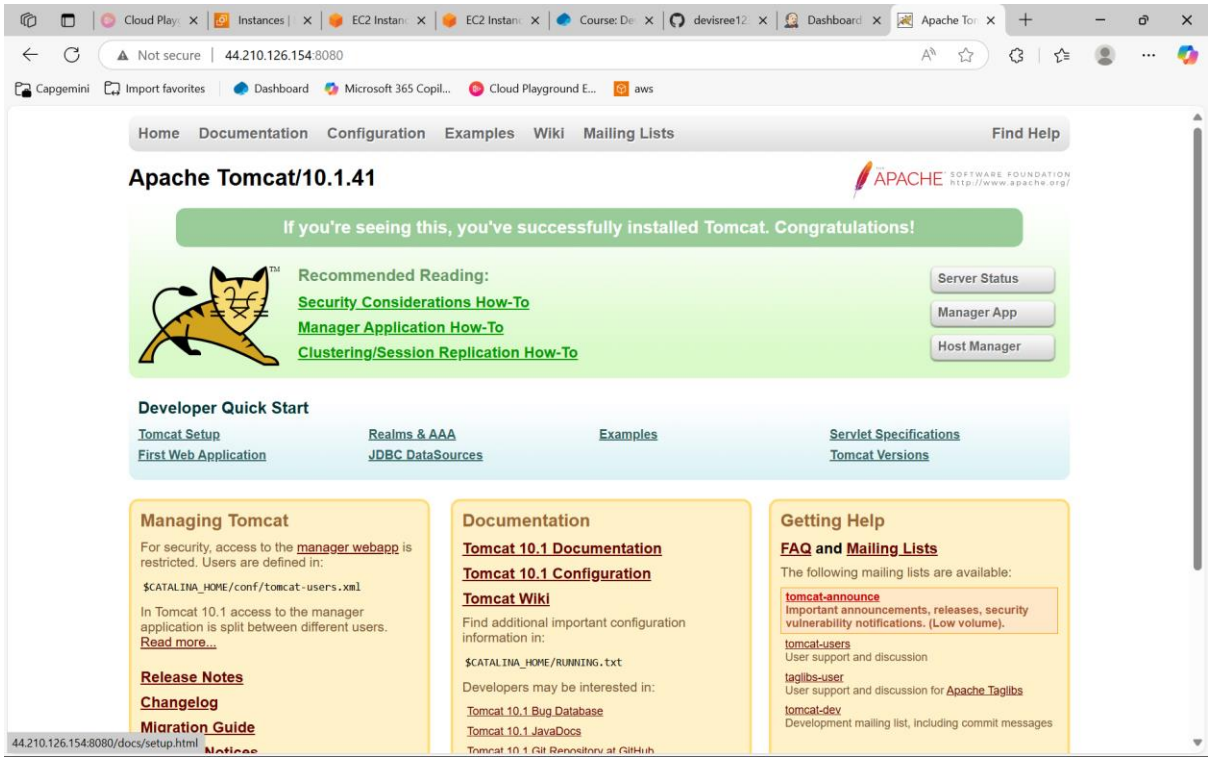


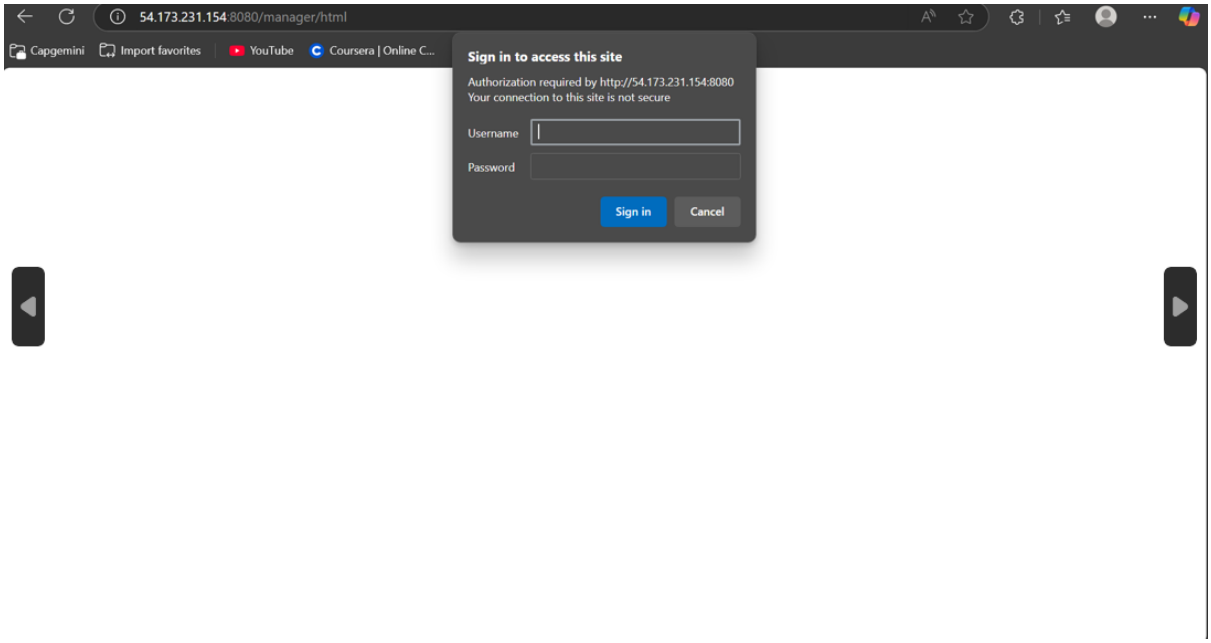
- Created an instance named Tomcat-server and installed java and tomcat to deploy the java application

1. Install apache tomcat:

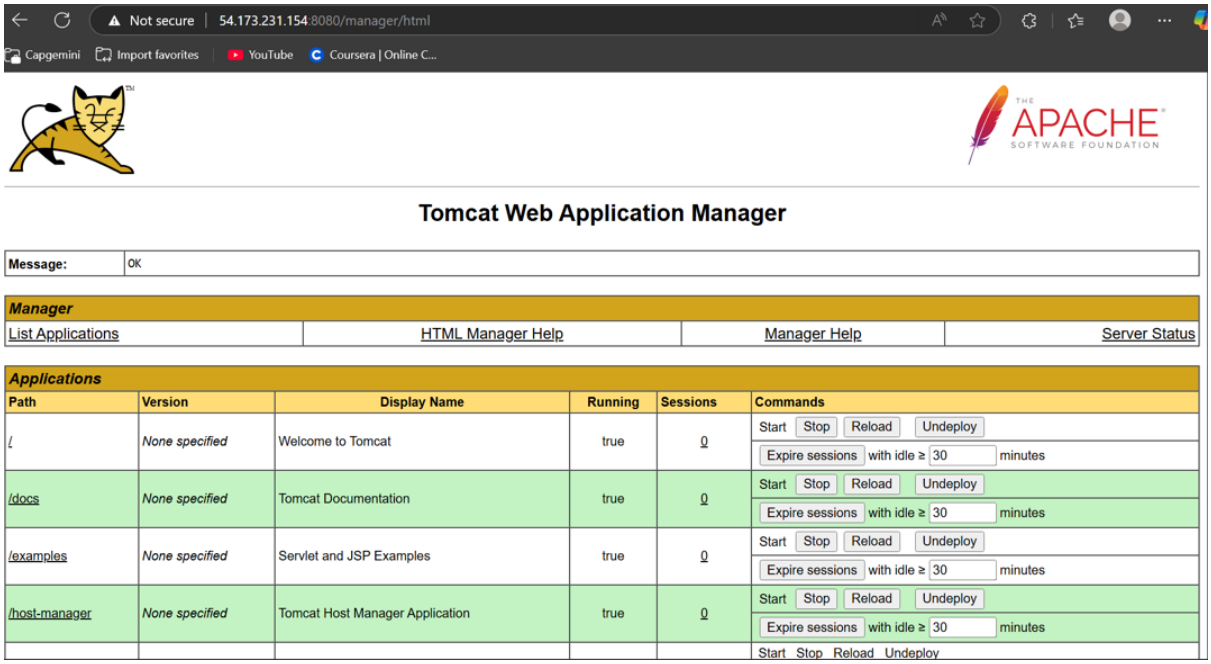
Sudo yum install tomcat



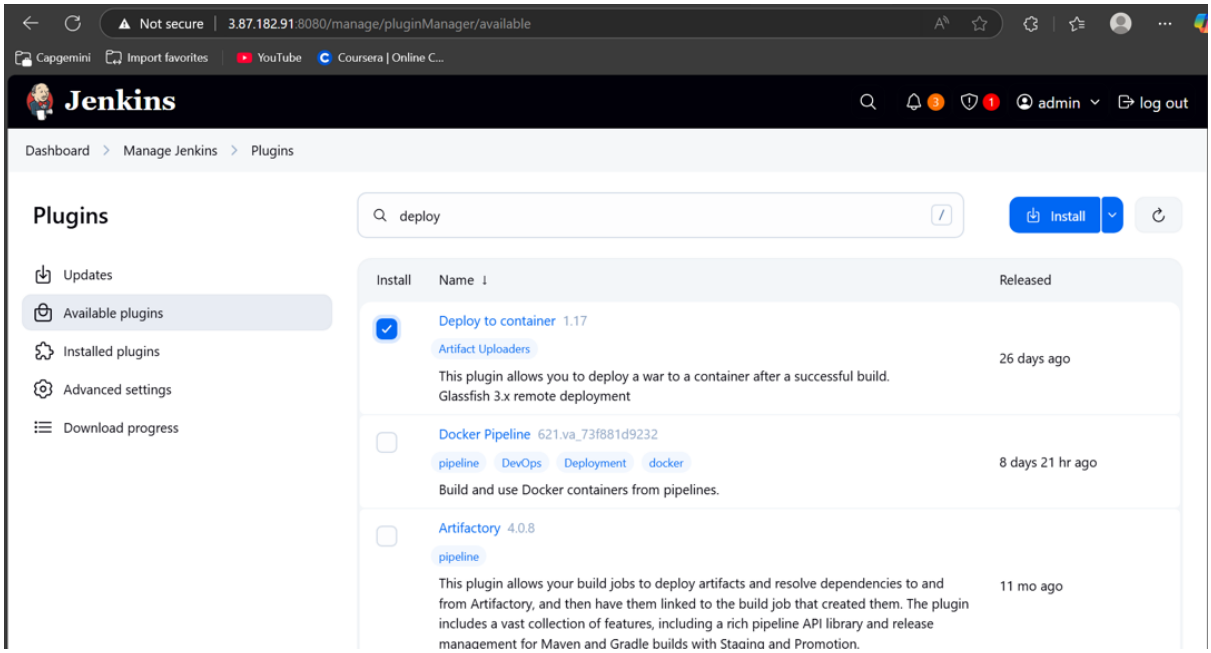


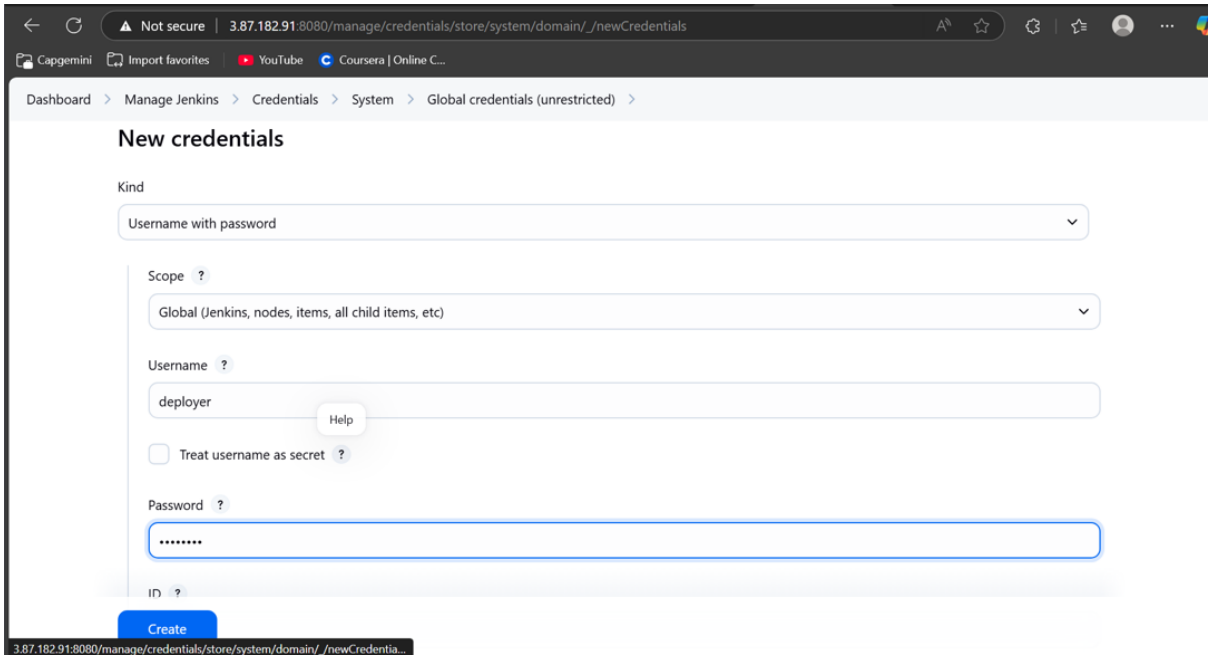


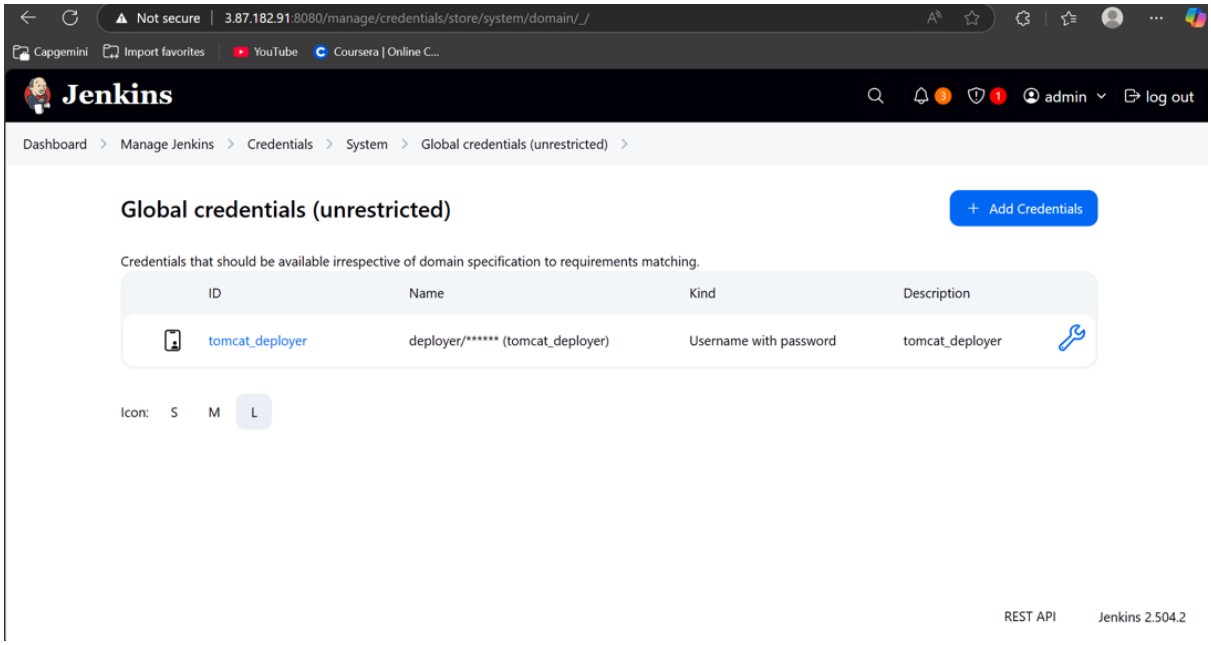
- Added manager script credentials in Jenkins to verify successful connection between Jenkins and tomcat.



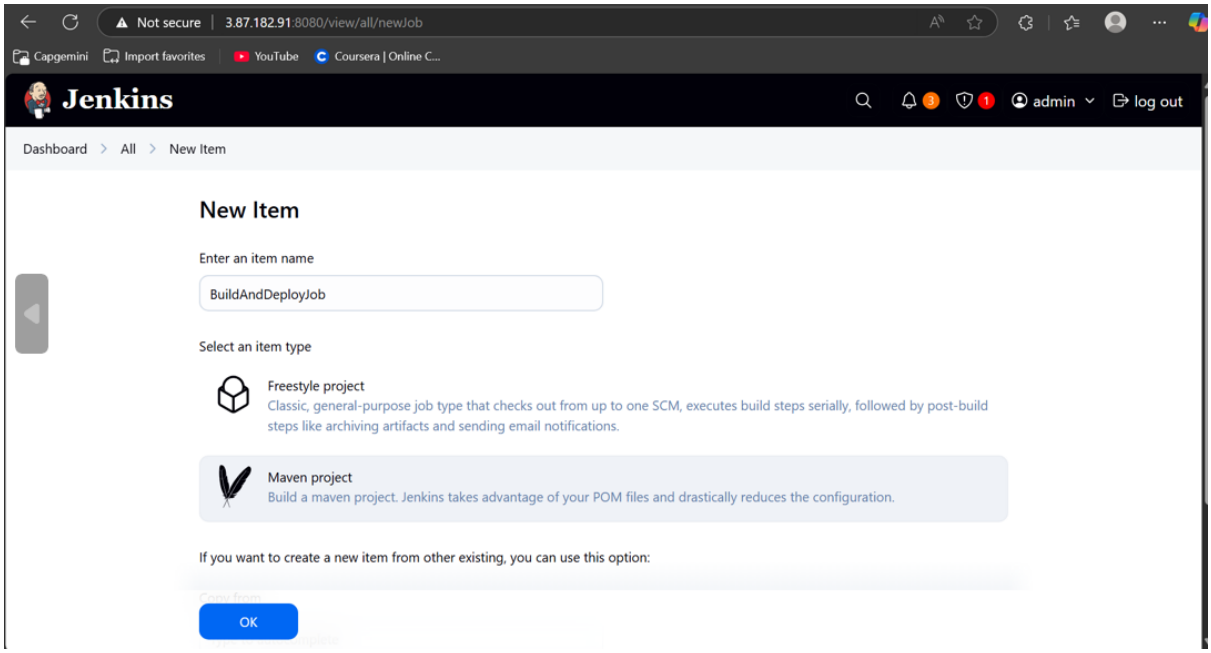
- Install the plugin named Deploy to container to deploy our java application.

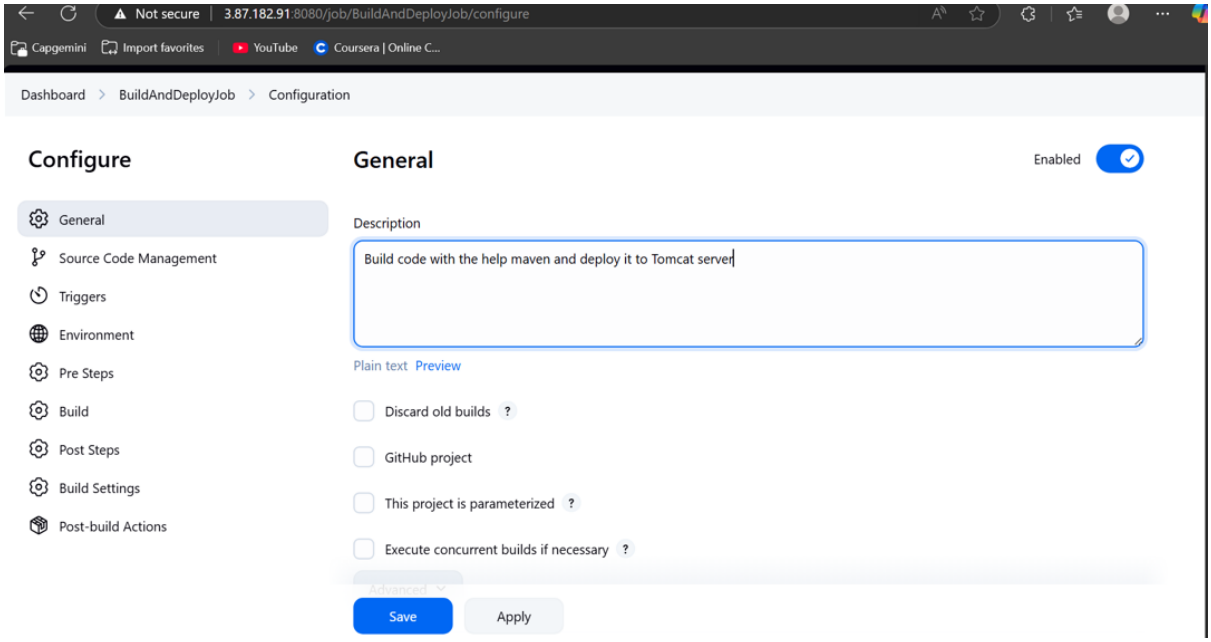


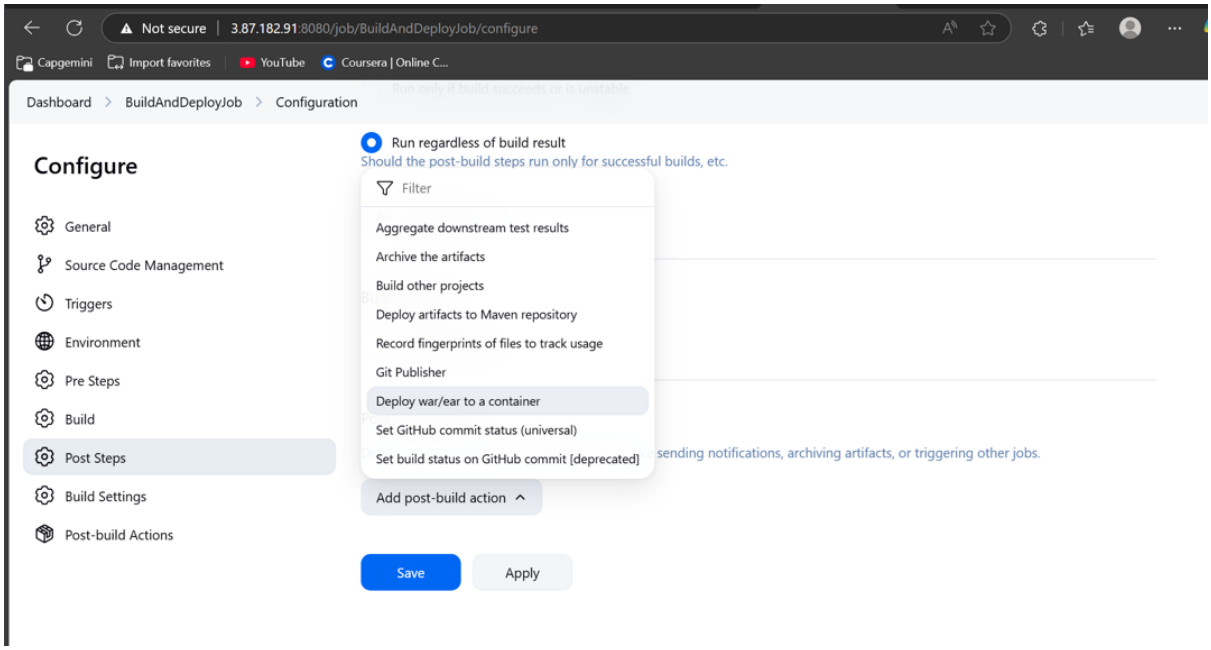


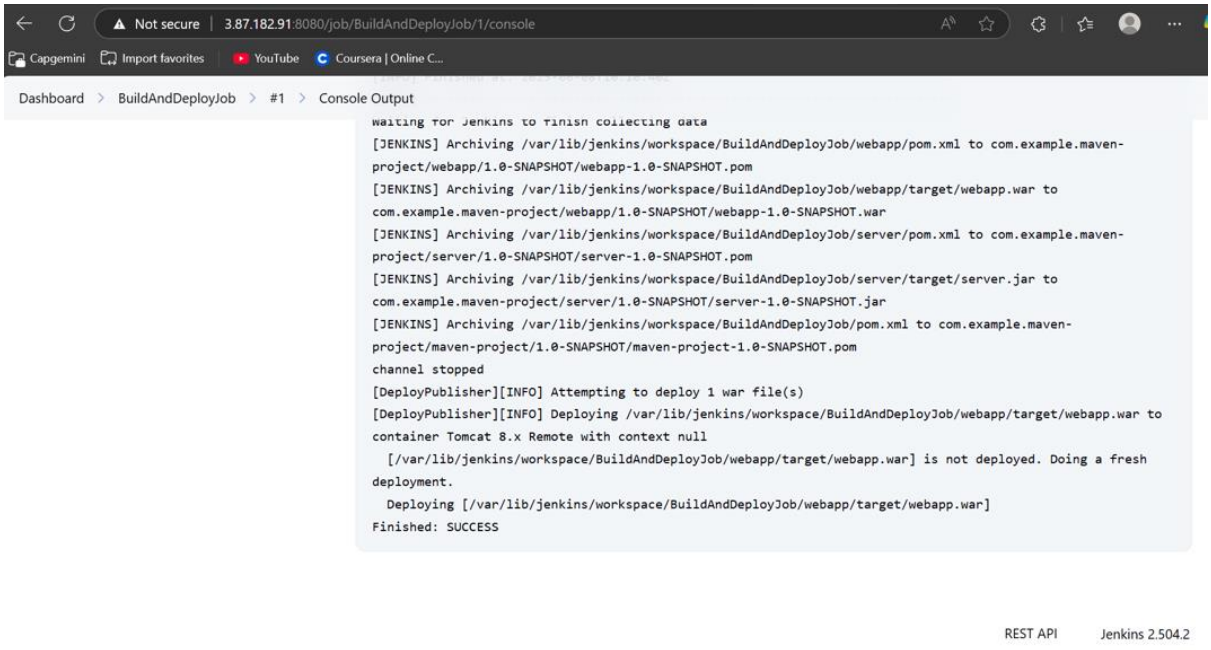


- Created an Jenkins pipeline named Build and deploy job to deploy the java application using Jenkins and tomcat server.

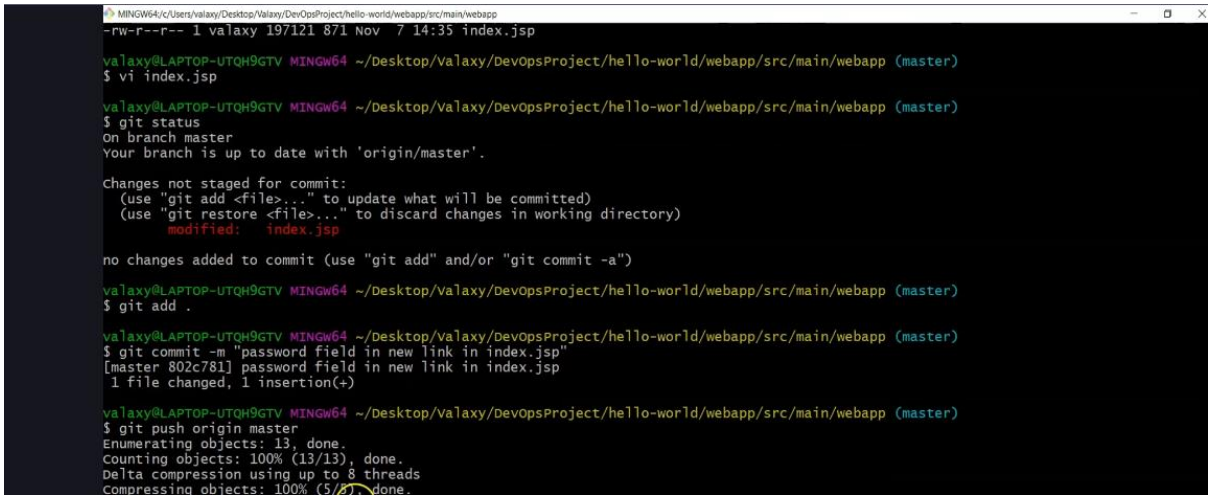



Not secure | 3.87.182.91:8080/job/BuildAndDeployJob/configure
Capgemini
Import favorites
YouTube
Coursera | Online C...
Dashboard > BuildAndDeployJob > Configuration
Configure
General
Source Code Management
Triggers
Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
General
Enabled
Description
Build code with the help maven and deploy it to Tomcat server
Plain text Preview
Discard old builds
GitHub project
This project is parameterized
Execute concurrent builds if necessary
Save
Apply

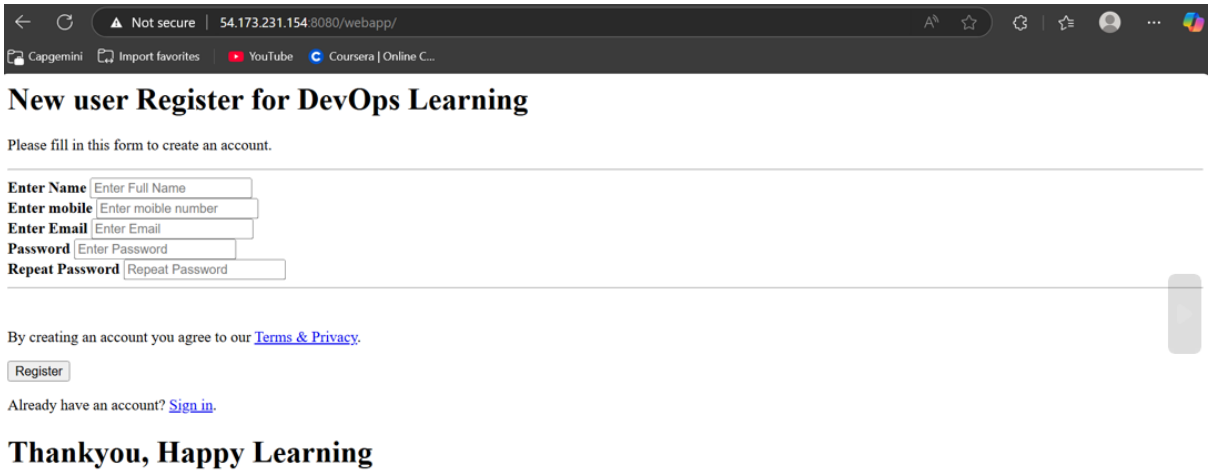

Not secure | 3.87.182.91:8080/job/BuildAndDeployJob/configure
Capgemini
Import favorites
YouTube
Coursera | Online C...
Dashboard > BuildAndDeployJob > Configuration
Configure
General
Source Code Management
Triggers
Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Run regardless of build result
Should the post-build steps run only for successful builds, etc.
Filter
Aggregate downstream test results
Archive the artifacts
Build other projects
Deploy artifacts to Maven repository
Record fingerprints of files to track usage
Git Publisher
Deploy war/ear to a container
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
sending notifications, archiving artifacts, or triggering other jobs.
Add post-build action
Save
Apply



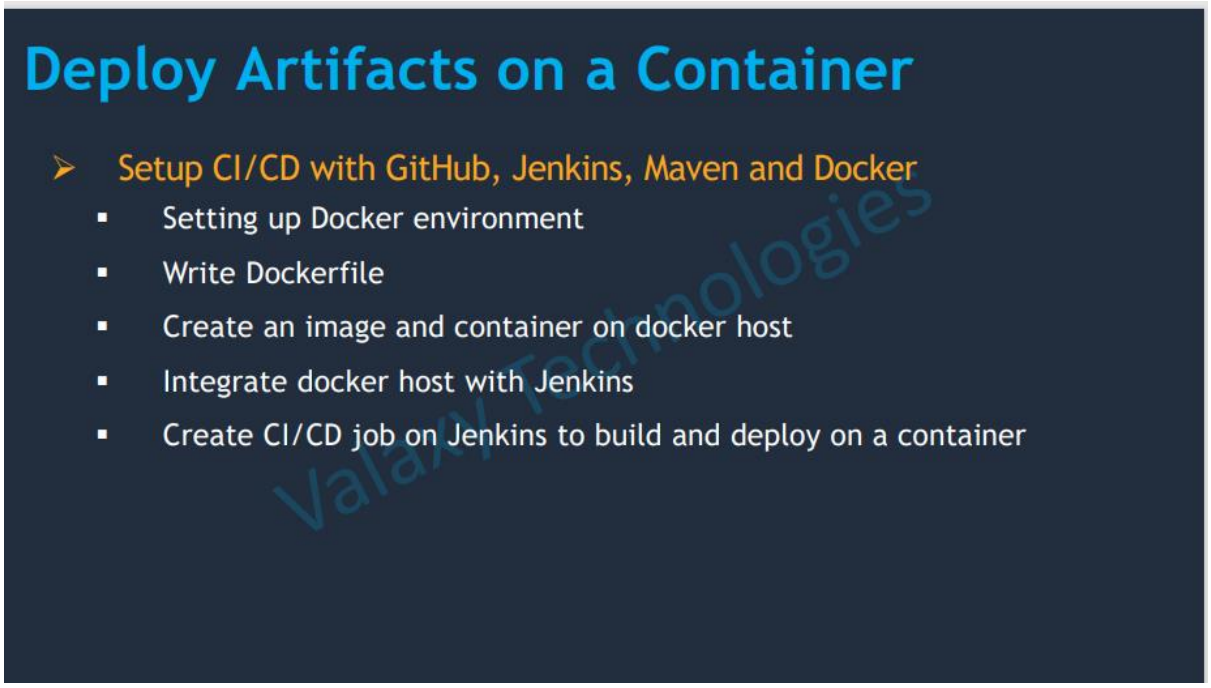
- To check whether the changes are updated or not we use git in this we cloned the java application to our local machine and later committed the changes to github

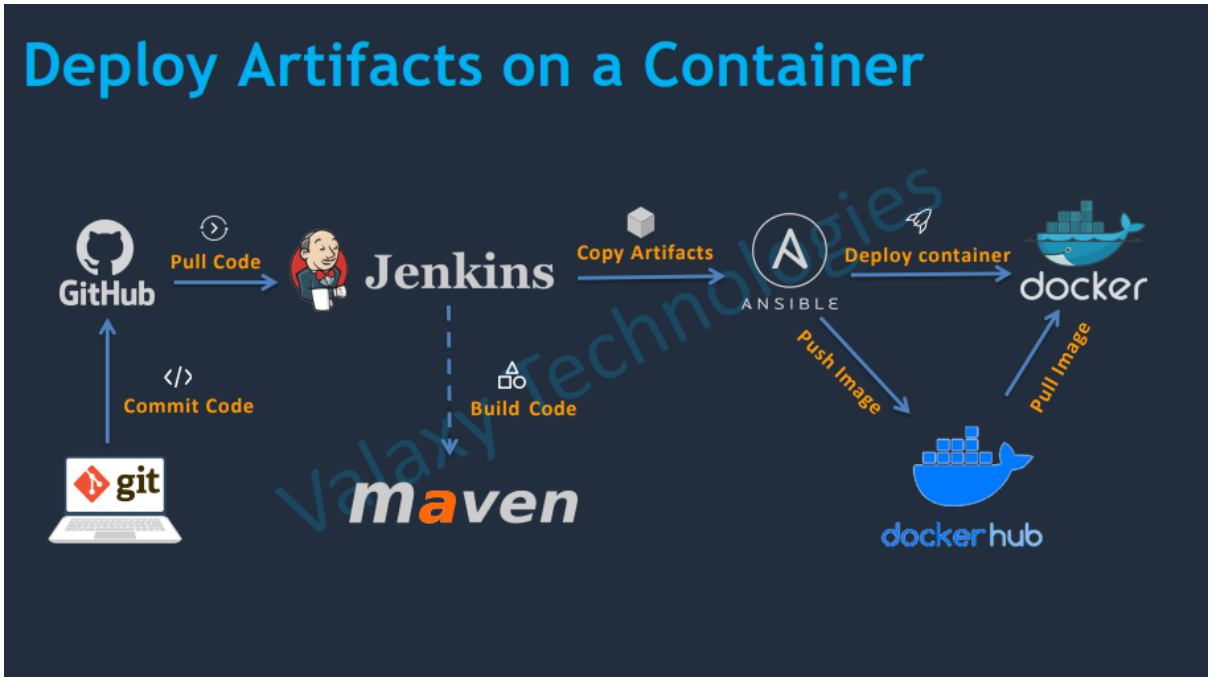


- Now Jenkins pull the code from github repository and build the application



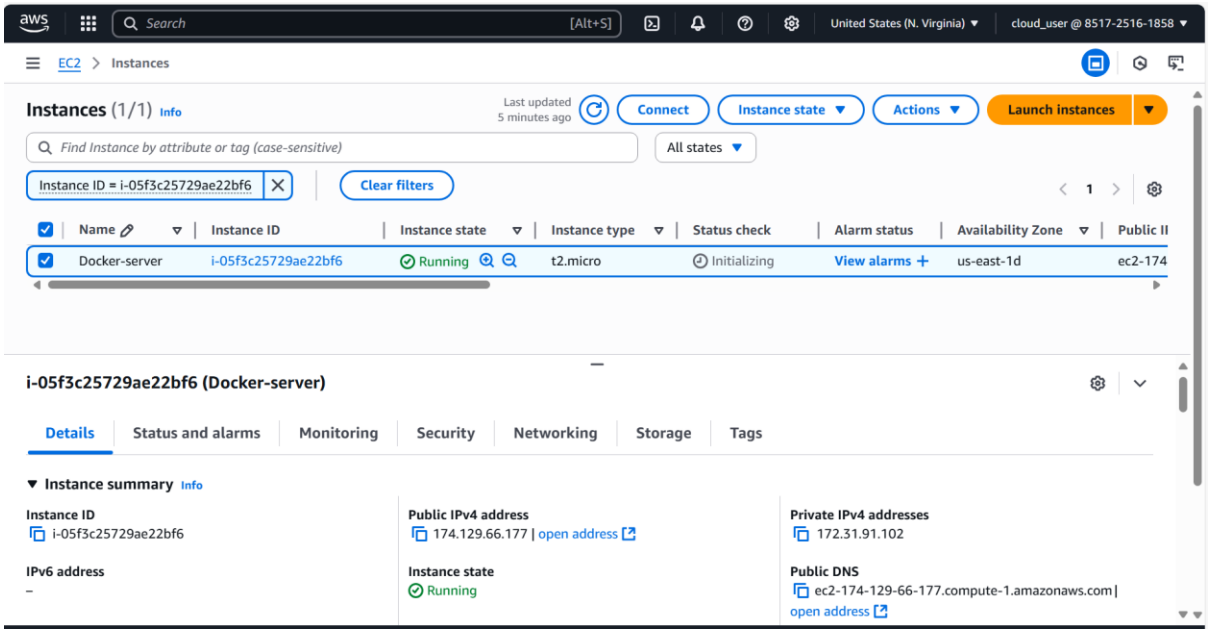
2. **Deploying java application on Docker**





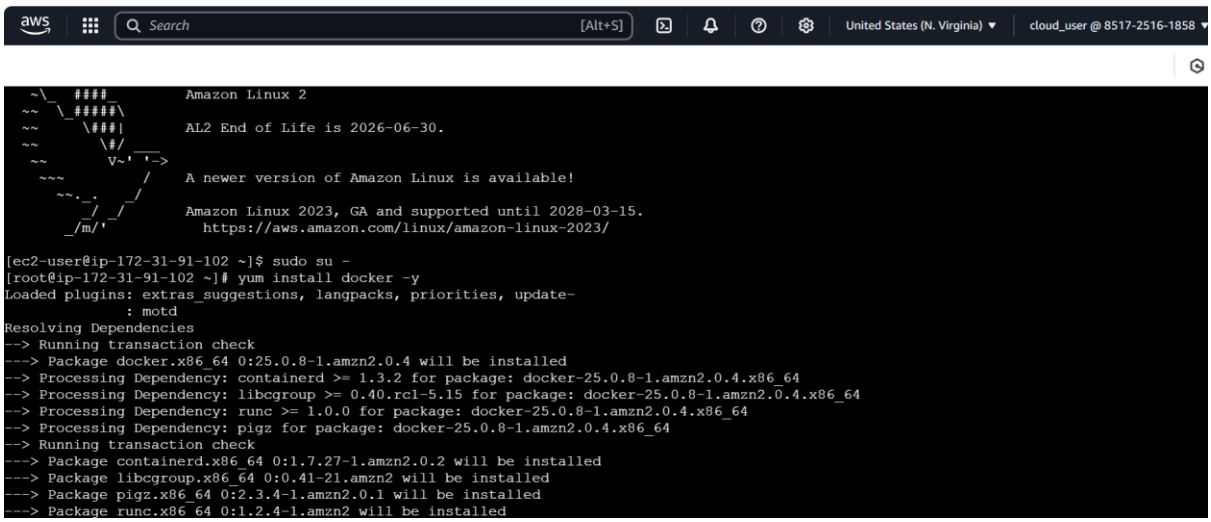
1. Setup Docker Environment

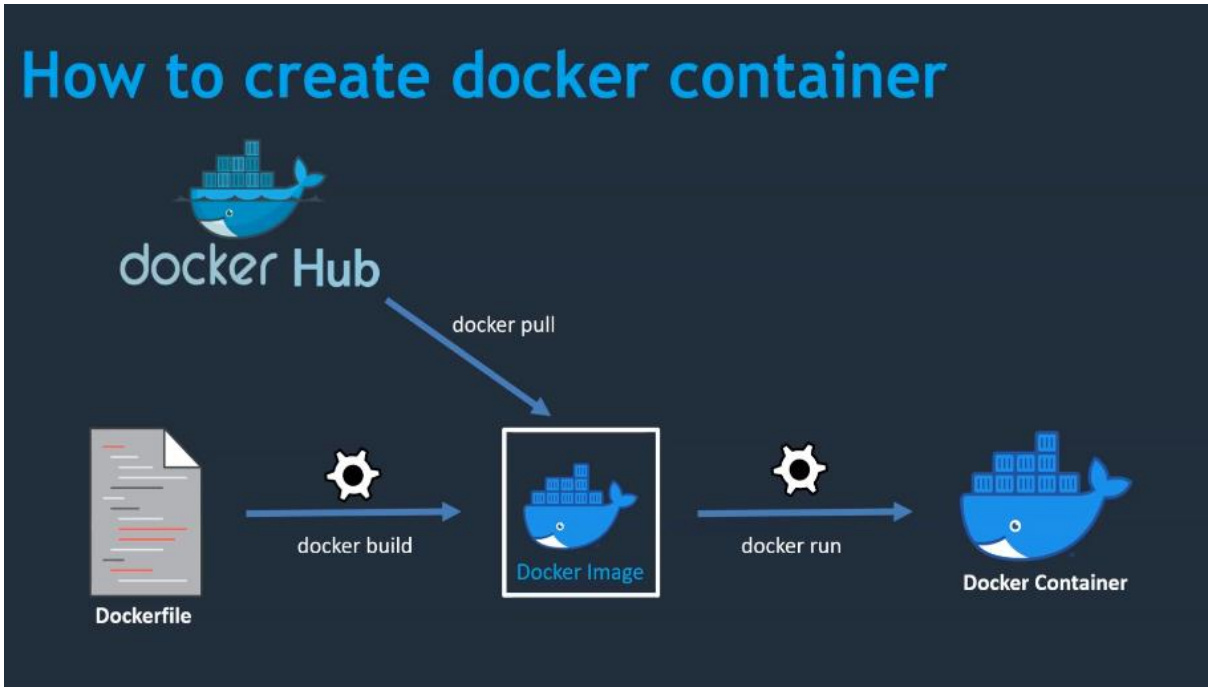
- Install Docker Engine on the build or deployment servers (e.g., Jenkins server or AWS EC2 instances).
- Verify Docker installation with basic commands like docker --version and docker run hello-world.
- Configure Docker daemon and user permissions to allow Jenkins and other tools to interact with Docker.

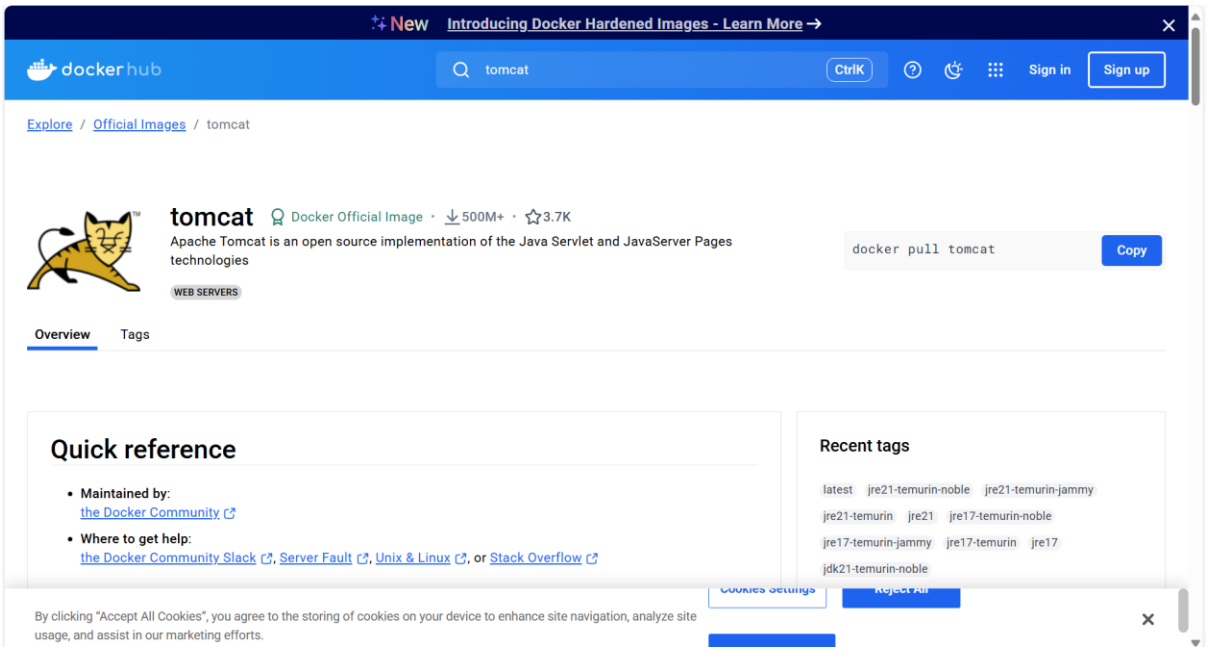


2. Create a Tomcat Container

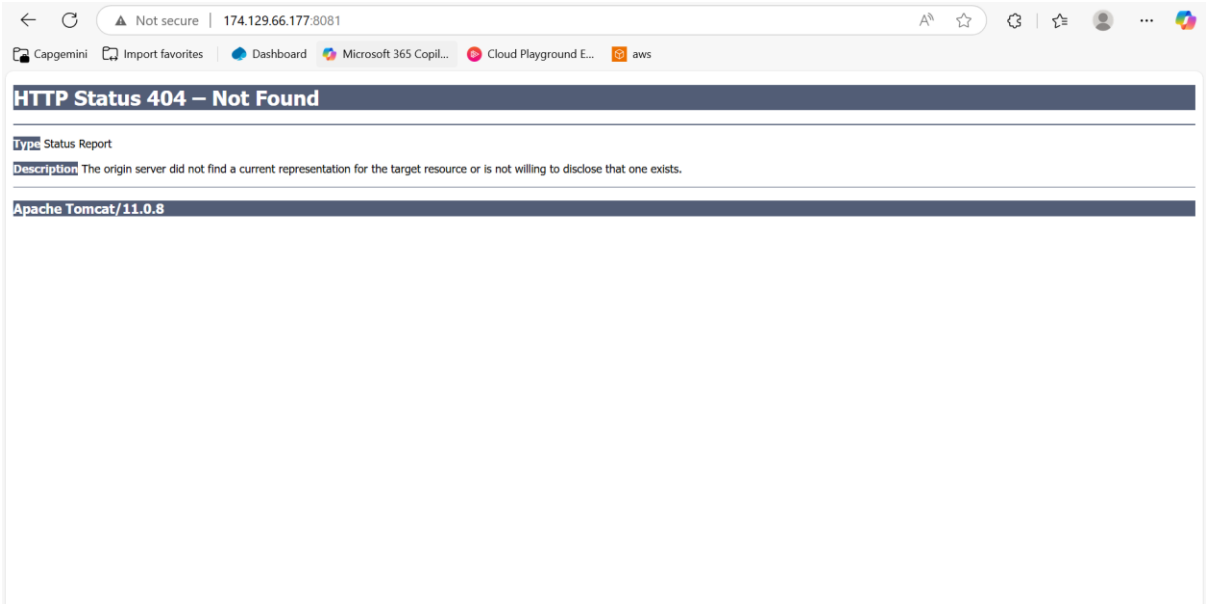
- Use the official Tomcat Docker image to spin up a containerized Tomcat server.
- Run commands such as docker run -d -p 8080:8080 tomcat to launch the container.
- Verify the container is running and accessible via the mapped port.





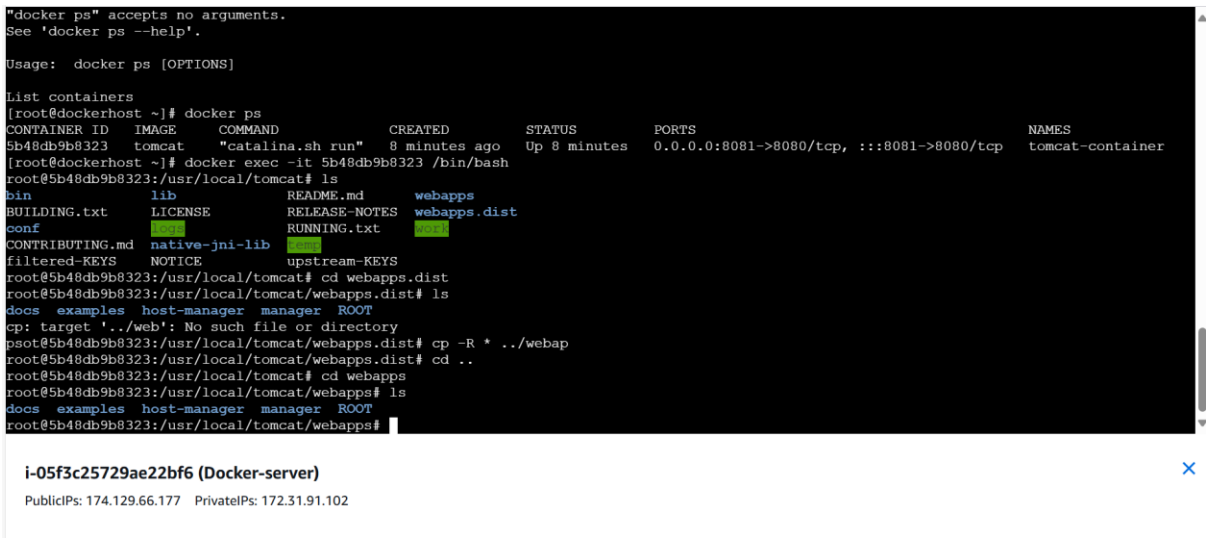


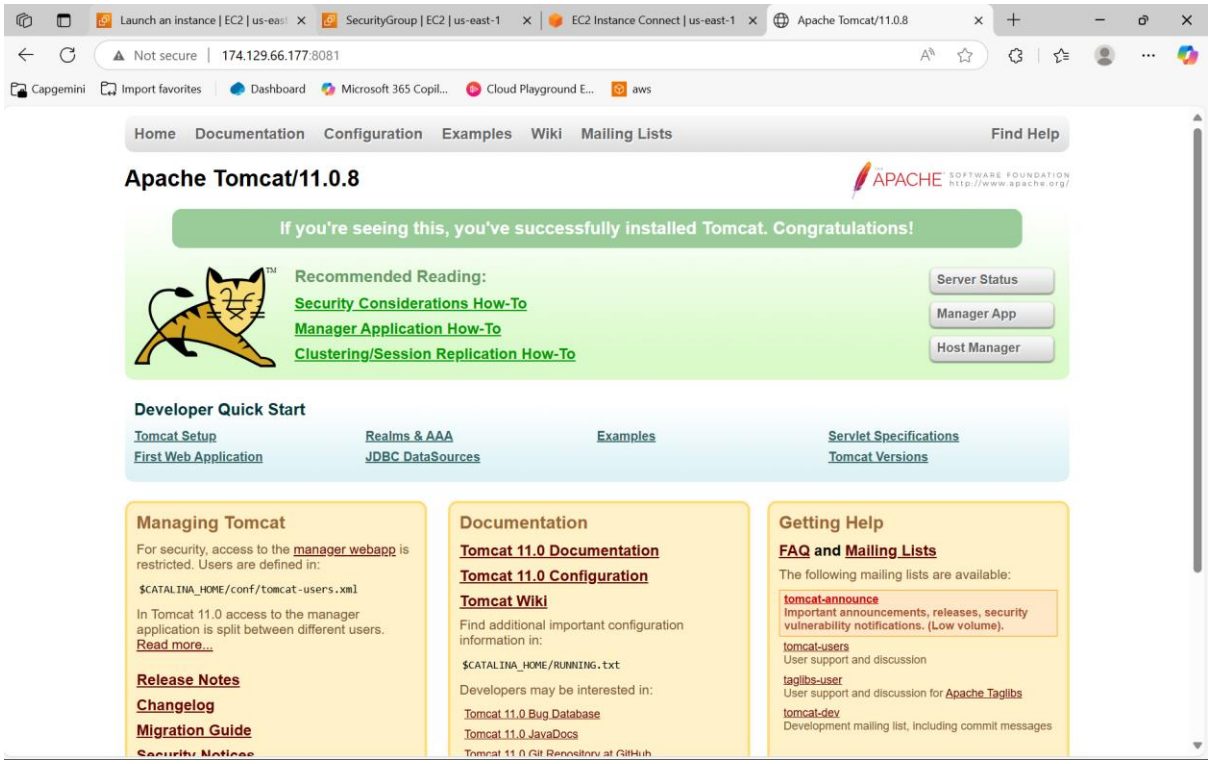

```
Digest: sha256:d2f9bdcd5b35fc7da231df399a9cda0d49ff402053d47f008dcb499ef3bcf950
Status: Downloaded newer image for tomcat:latest
docker.io/library/tomcat:latest
[root@dockerhost ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
tomcat       latest    86433616469a   21 hours ago   468MB
[root@dockerhost ~]# docker run -d --name tomcat-container -p 8081:8080 tomcat
5b48db9b83237e67c3474ff16bbefefed83f75b1bac37892b3973f67daad2266
[root@dockerhost ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND             CREATED          STATUS          PORTS                                       NAMES
5b48db9b8323   tomcat    "catalina.sh run"   12 seconds ago   Up 11 seconds   0.0.0.0:8081->8080/tcp, :::8081->8080/tcp   tomcat-containe
r
[root@dockerhost ~]# docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS         PORTS                                       NAMES
5b48db9b8323   tomcat    "catalina.sh run"   3 minutes ago   Up 3 minutes   0.0.0.0:8081->8080/tcp, :::8081->8080/tcp   tomcat-container
[root@dockerhost ~]#
```

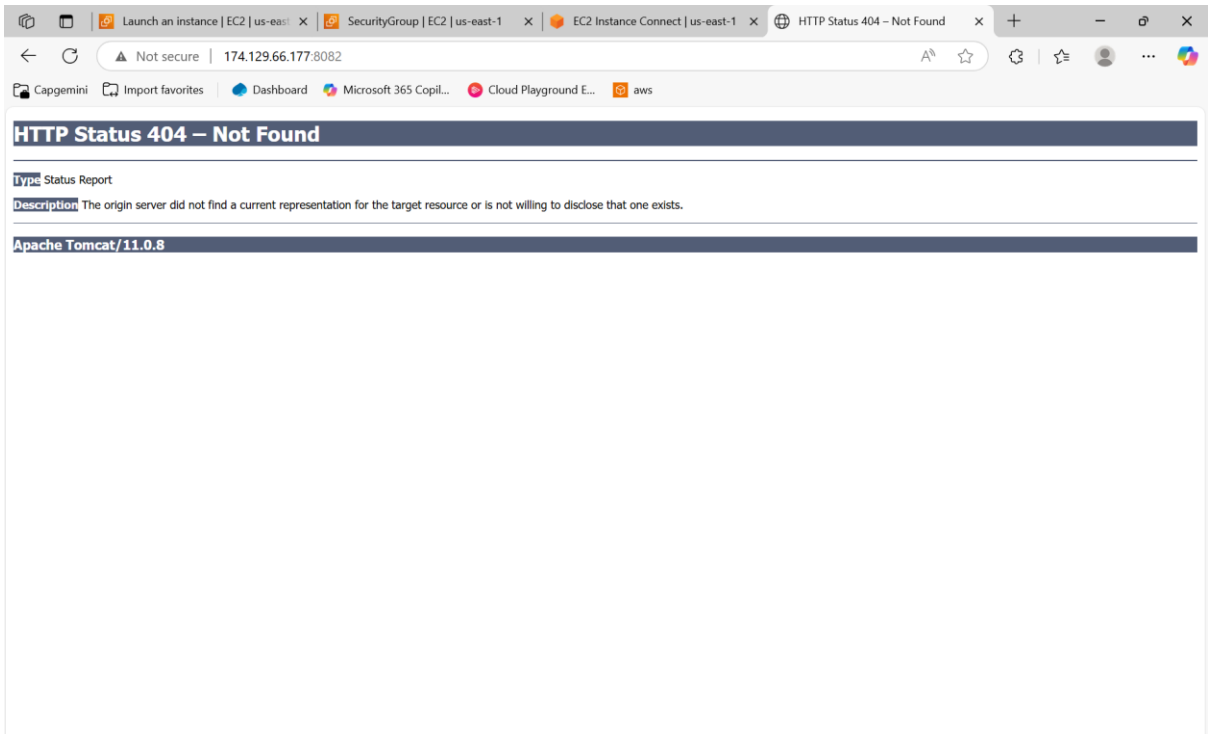


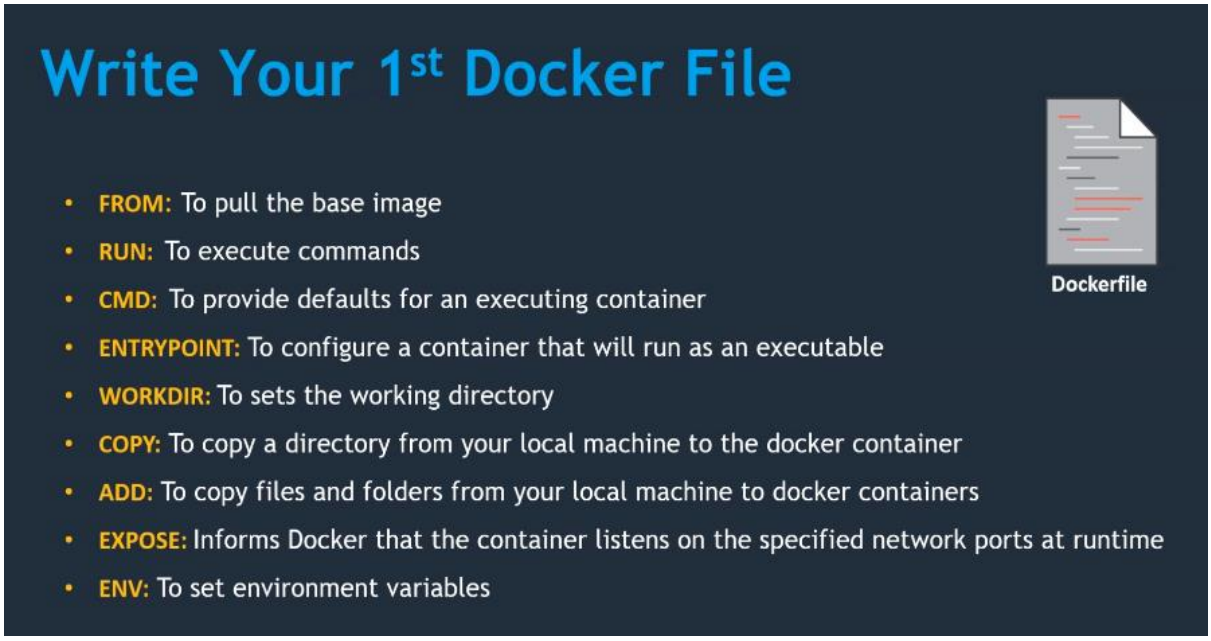
3. Fixing Tomcat Container Issues

- Troubleshoot common problems such as port conflicts, volume mounts, or missing dependencies.
- Adjust Dockerfile or container settings to fix deployment errors or startup failures.
- Check container logs with docker logs <container_id> to identify issues.



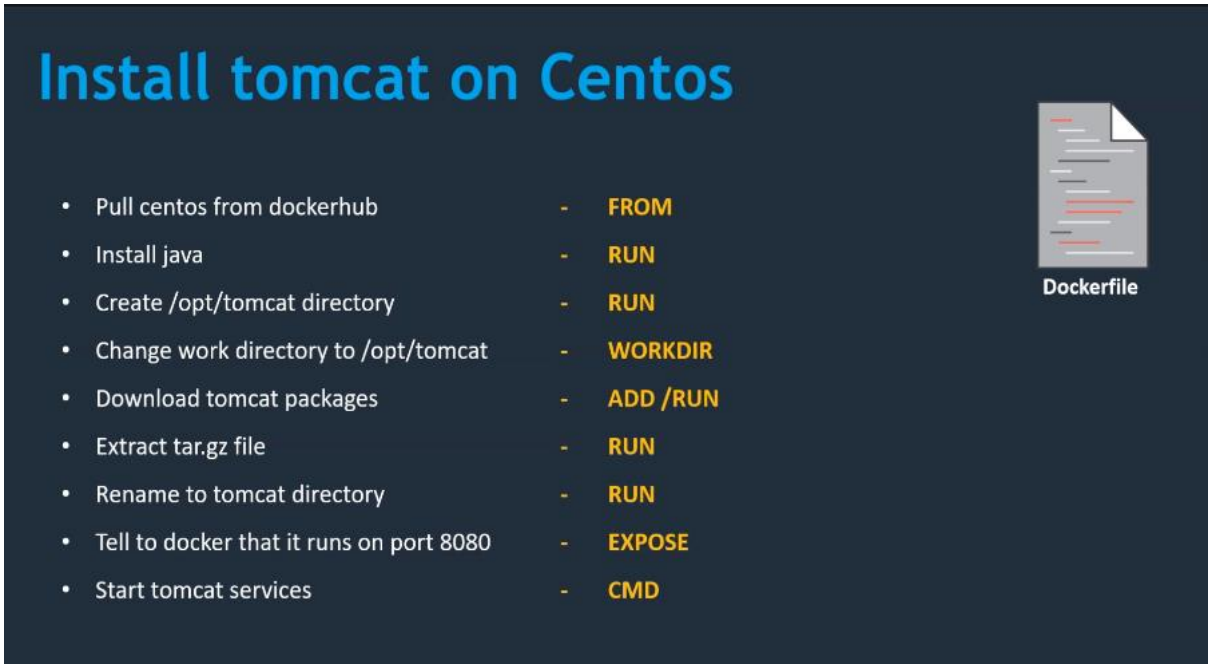

Launch an instance | EC2 | us-east
SecurityGroup | EC2 | us-east-1
EC2 Instance Connect | us-east-1
Apache Tomcat/11.0.8
Not secure | 174.129.66.177:8081
Capgemini
Import favorites
Dashboard
Microsoft 365 Copil...
Cloud Playground E...
aws
Home
Documentation
Configuration
Examples
Wiki
Mailing Lists
Find Help
Apache Tomcat/11.0.8
APACHE SOFTWARE FOUNDATION http://www.apache.org/
If you're seeing this, you've successfully installed Tomcat. Congratulations!
Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To
Server Status
Manager App
Host Manager
Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions
Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
$CATALINA_HOME/conf/tomcat-users.xml
In Tomcat 11.0 access to the manager application is split between different users.
Read more...
Release Notes
Changelog
Migration Guide
Documentation
Tomcat 11.0 Documentation
Tomcat 11.0 Configuration
Tomcat Wiki
Find additional important configuration information in:
$CATALINA_HOME/RUNNING.txt
Developers may be interested in:
Tomcat 11.0 Bug Database
Tomcat 11.0 JavaDocs
Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user
User support and discussion for Apache Taglibs
tomcat-dev
Development mailing list, including commit messages


Launch an instance | EC2 | us-east
SecurityGroup | EC2 | us-east-1
EC2 Instance Connect | us-east-1
HTTP Status 404 – Not Found
Not secure | 174.129.66.177:8082
Capgemini
Import favorites
Dashboard
Microsoft 365 Copil...
Cloud Playground E...
aws
HTTP Status 404 – Not Found
Type Status Report
Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.
Apache Tomcat/11.0.8



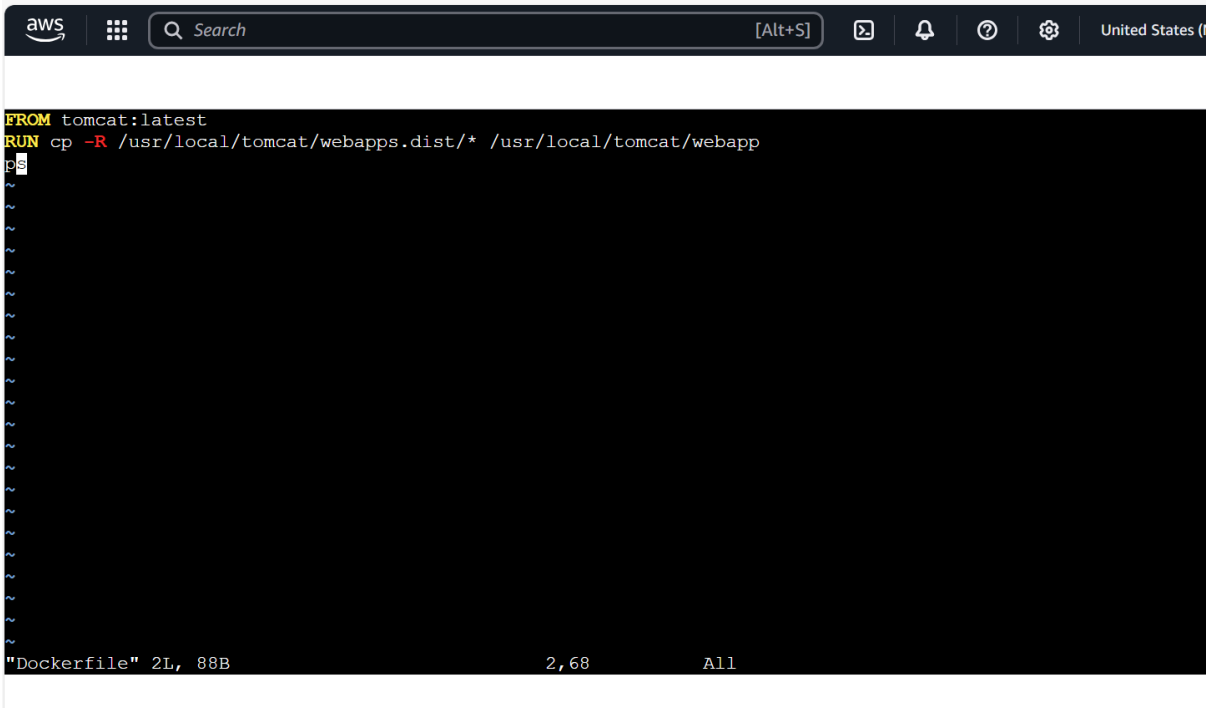
4. Create a First Dockerfile

- Write a basic Dockerfile that defines the environment and instructions to build a container image.
- Include steps like setting the base image (e.g., FROM tomcat), copying application artifacts, and exposing necessary ports.
- Build the Docker image locally using docker build -t mytomcat:latest .

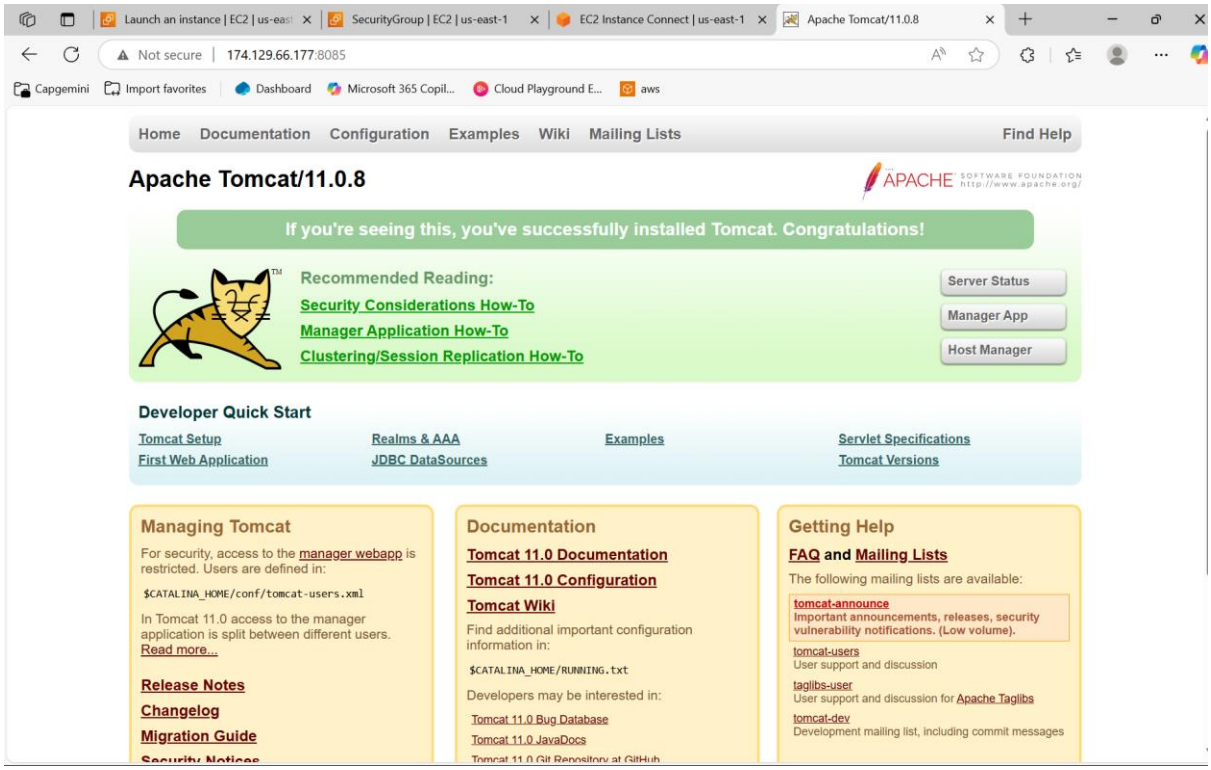


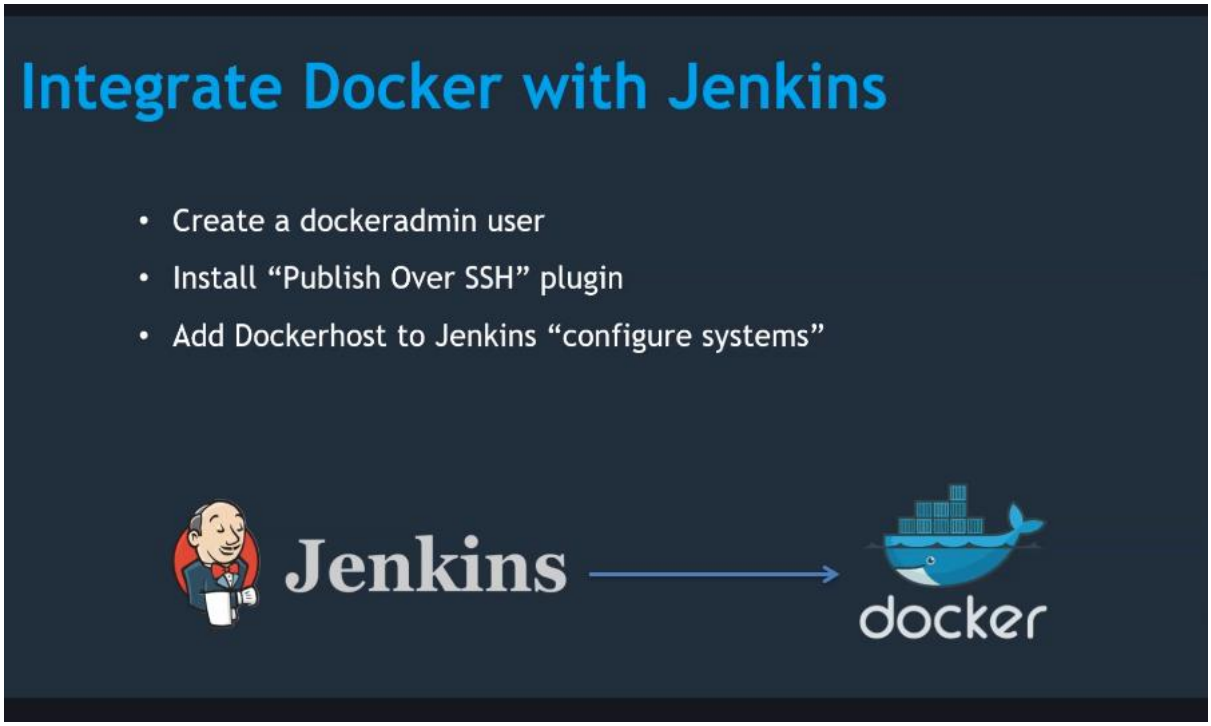
5. Create a Customized Dockerfile for Tomcat

- Enhance the Dockerfile to include custom configurations, environment variables, or scripts needed for the application.
- Automate deployment steps like copying WAR files into the Tomcat webapps directory within the image.
- Optimize the Docker image size and startup time for faster deployments.



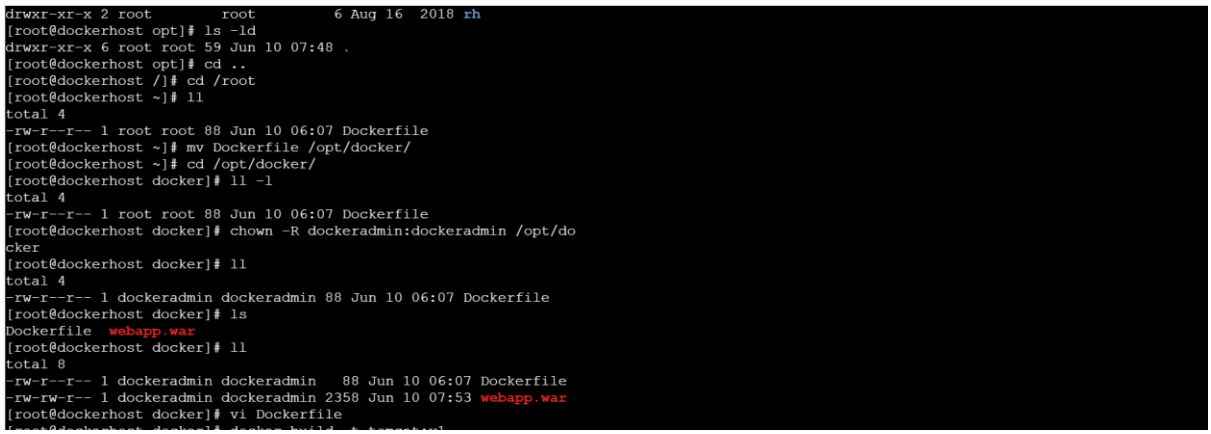
```
[root@dockerhost ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
demotomcat   latest    db95efe31251   9 seconds ago    474MB
tomcat       latest    86433616469a   22 hours ago     468MB
8085:8080 demotomcat docker run -d --name demotomcat-container -p
3a0f86423cfabf593cdd20bbfa20ee1cd1db1d609afde0ead2bc00f44c7d94c9
[root@dockerhost ~]# docker ps -a
CONTAINER ID   IMAGE        COMMAND             CREATED          STATUS                      PORTS
  NAMES
3a0f86423cfa   demotomcat   "catalina.sh run"   5 seconds ago    Up 4 seconds                0.0.0.0:8085->8080/tcp, :::8085->8080/tcp
  demotomcat-container
f286e17a97de   tomcat       "catalina.sh run"   27 minutes ago   Up 26 minutes               0.0.0.0:8082->8080/tcp, :::8082->8080/tcp
  tomcat2
5b48db9b8323   tomcat       "catalina.sh run"   42 minutes ago   Exited (143) 28 minutes ago
  tomcat-container   vi Dockerfile
[root@dockerhost ~]#
```





6. Integrate Docker with Jenkins

- Configure Jenkins with Docker plugins to enable Docker commands within Jenkins jobs.
- Set Jenkins agents with Docker installed or use Docker-in-Docker setups.
- Define Jenkins pipeline steps that build, tag, and push Docker images.

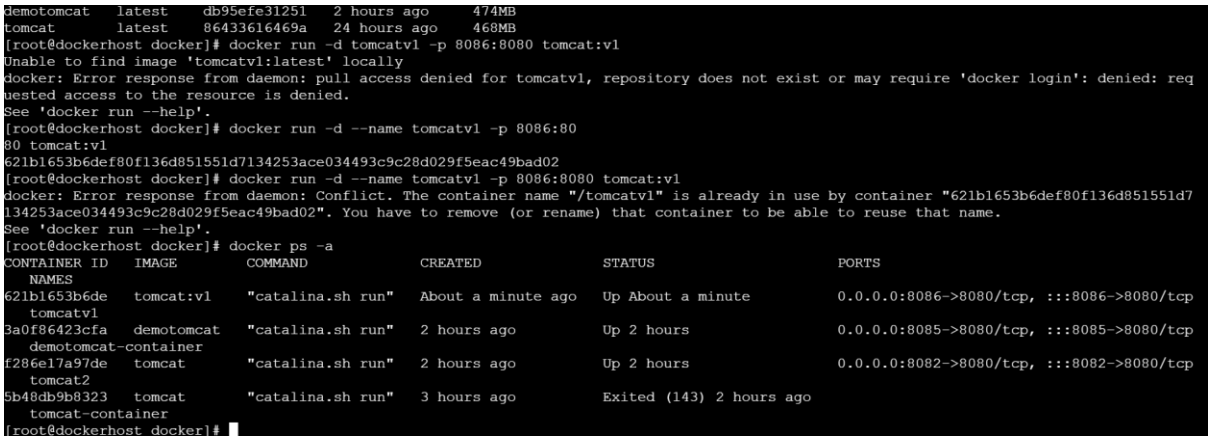


7. Jenkins Job to Build and Copy Artifacts onto Docker Host

- Automate the process of building the Java application with Maven.
- Copy the build artifact (WAR/JAR) to the Docker build context or directly to the Docker host.
- Trigger Docker image build and container deployment from Jenkins jobs.

8. Update Tomcat Dockerfile to Automate Deployment Process

- Modify the Dockerfile to fully automate application deployment by embedding the build artifact during image creation.
- Ensure that each new Docker image contains the latest version of the application ready for deployment.
- Automate versioning and tagging of Docker images in the pipeline.

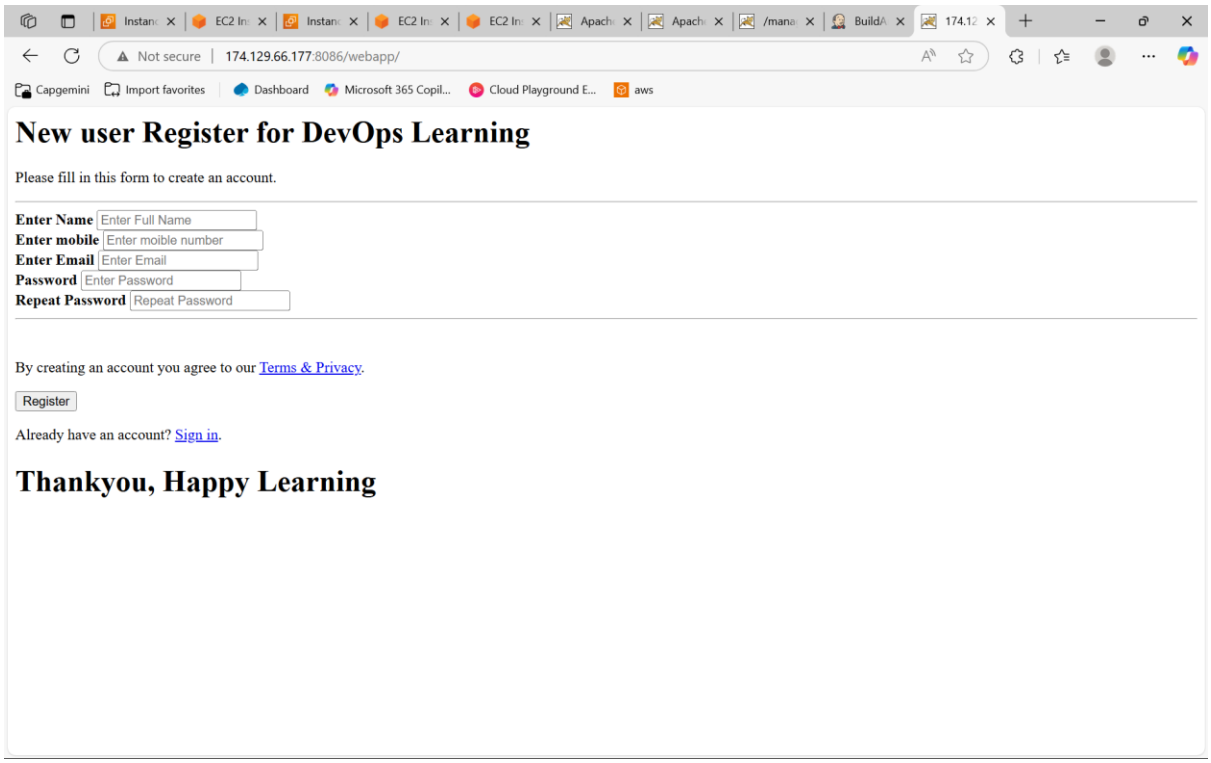


9. Automate Build and Deployment on Docker Container

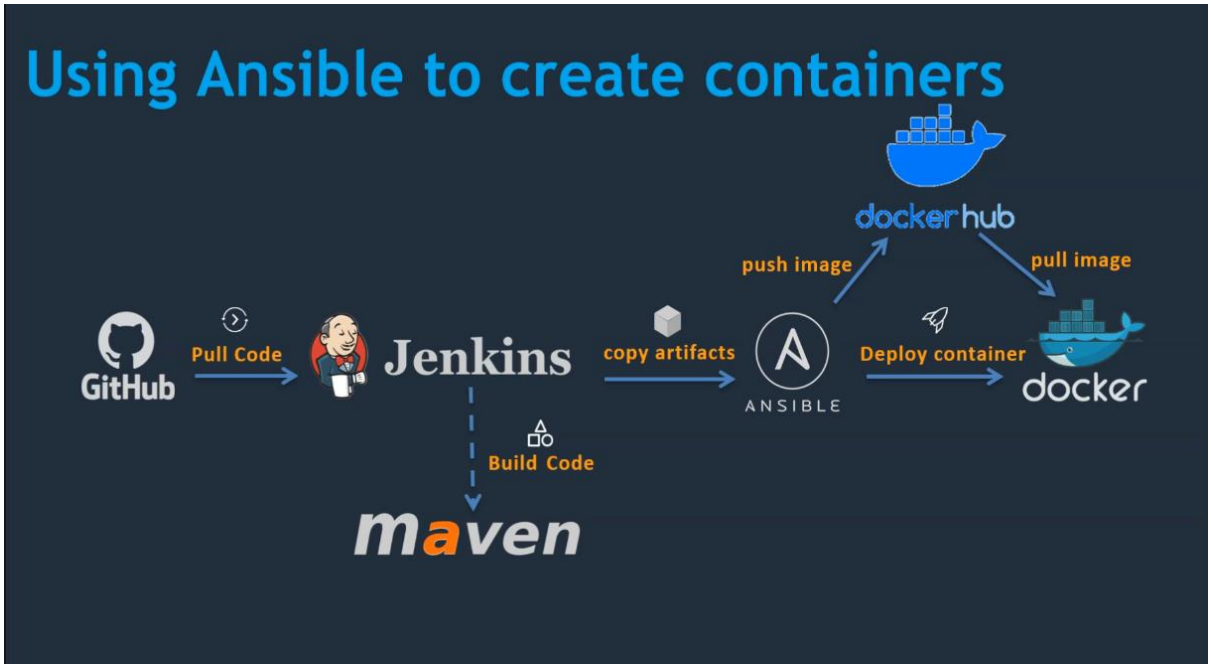
- Create Jenkins pipeline stages for continuous building, testing, and deploying Docker containers.
- Use Docker Compose or Kubernetes if needed to manage multi-container deployments.
- Automate container lifecycle management including start, stop, and rollback procedures.

10. Jenkins Job to Automate CI/CD to Deploy Application on Docker Container

- Combine all steps into a fully automated Jenkins pipeline job that listens for code changes.
- Upon each commit, Jenkins triggers build, test, Docker image creation, and deployment to the Docker environment.
- Achieve continuous integration and continuous deployment with minimal manual intervention.

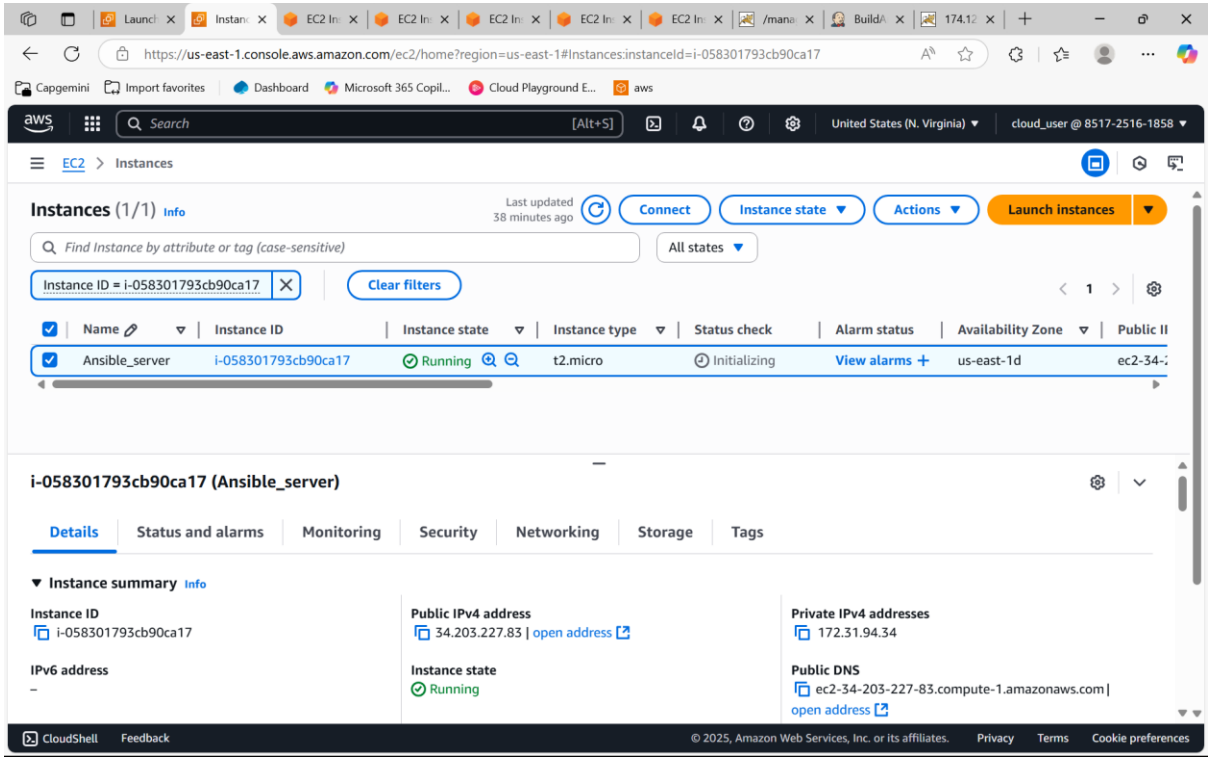


3.Integrating Ansible in CI/CD Pipeline



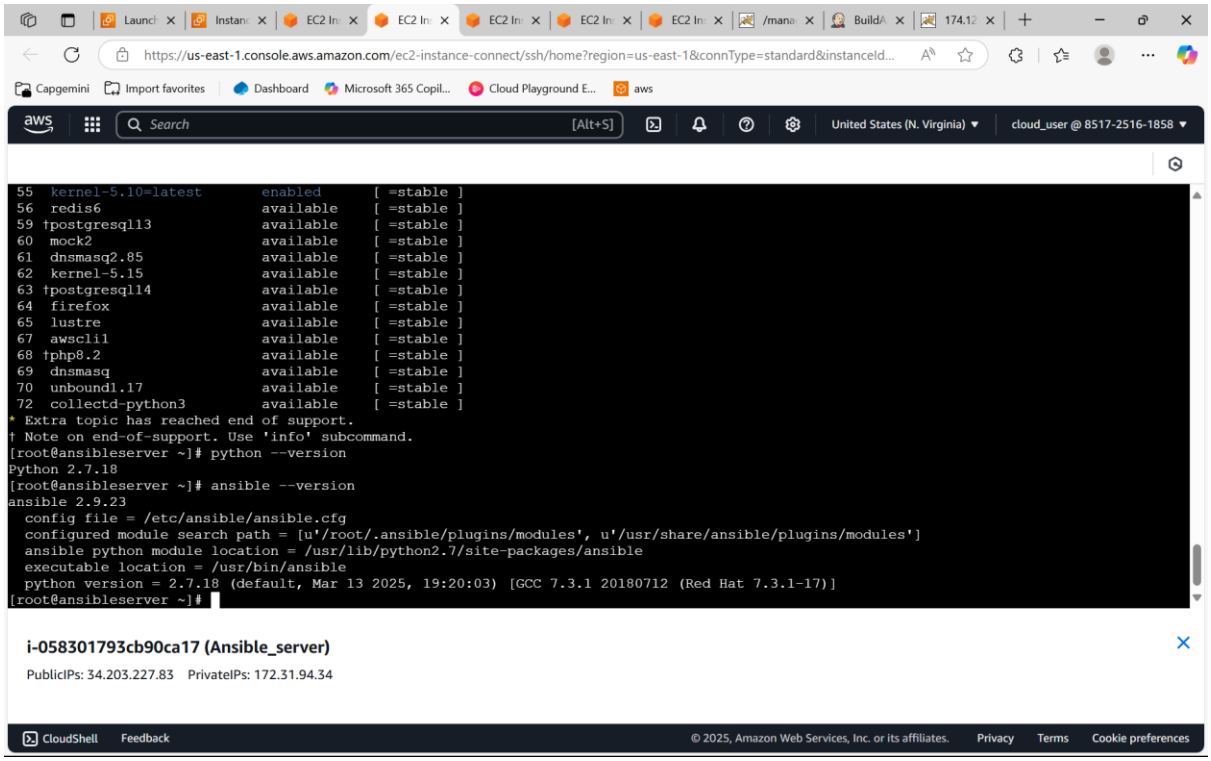
1. Section Introduction – Why Do We Need Ansible?

- Ansible is a powerful automation tool that simplifies configuration management, application deployment, and orchestration.
- It helps manage infrastructure as code with easy-to-write YAML playbooks, reducing manual intervention and errors.
- Integrating Ansible into CI/CD pipelines allows automated, repeatable, and consistent deployment processes across environments.



2. Ansible Installation

- Install Ansible on the control node (e.g., Jenkins server or a dedicated automation server).
- Verify the installation with commands like ansible --version and configure necessary dependencies such as Python and SSH.
- Set up inventory files defining the target hosts and configure SSH access for passwordless login.

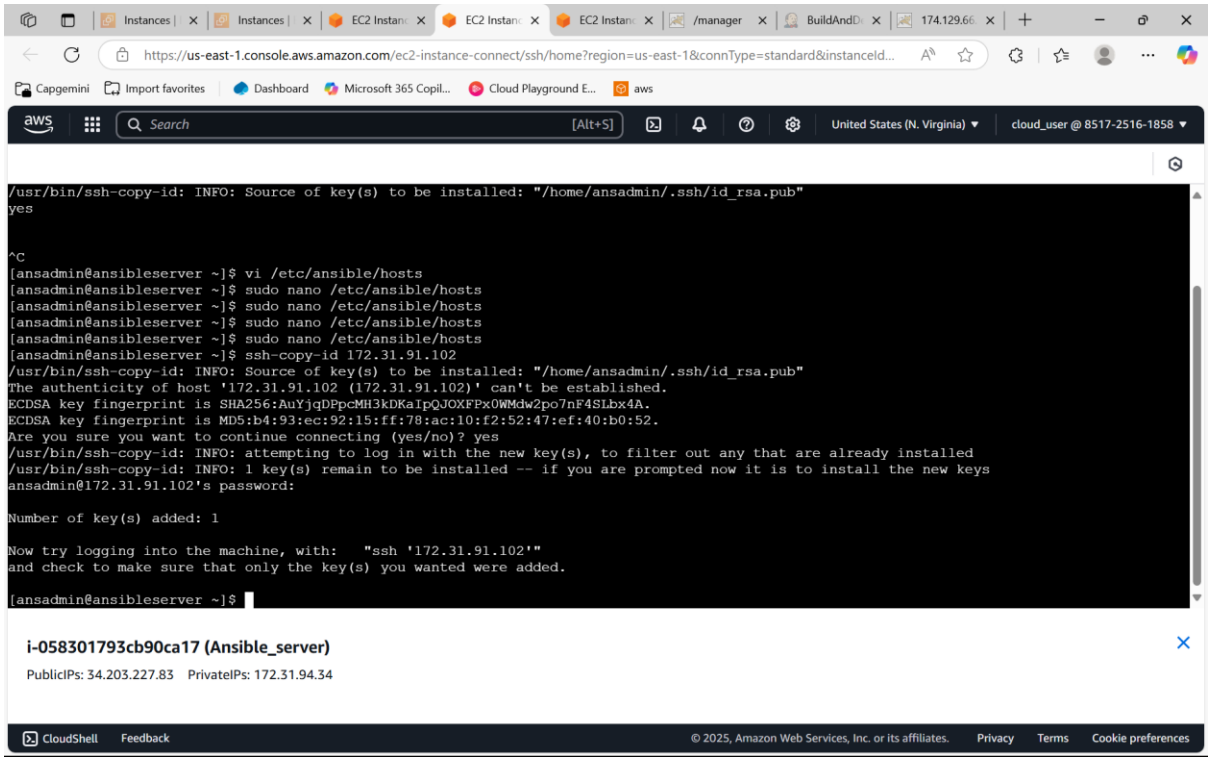


3. Integrate Docker with Ansible

- Use Ansible Docker modules to manage Docker images, containers, and networks programmatically.
- Create playbooks that automate Docker tasks such as building images, starting containers, and cleaning up old containers.
- Enable Ansible to interact with Docker daemons on remote or local hosts.

4. Integrate Ansible with Jenkins

- Configure Jenkins to run Ansible playbooks as build or post-build steps using plugins like "Ansible plugin".
- Pass variables and parameters from Jenkins to Ansible playbooks for flexible deployments.
- Use Jenkins to trigger Ansible-driven deployments as part of the CI/CD pipeline automatically.

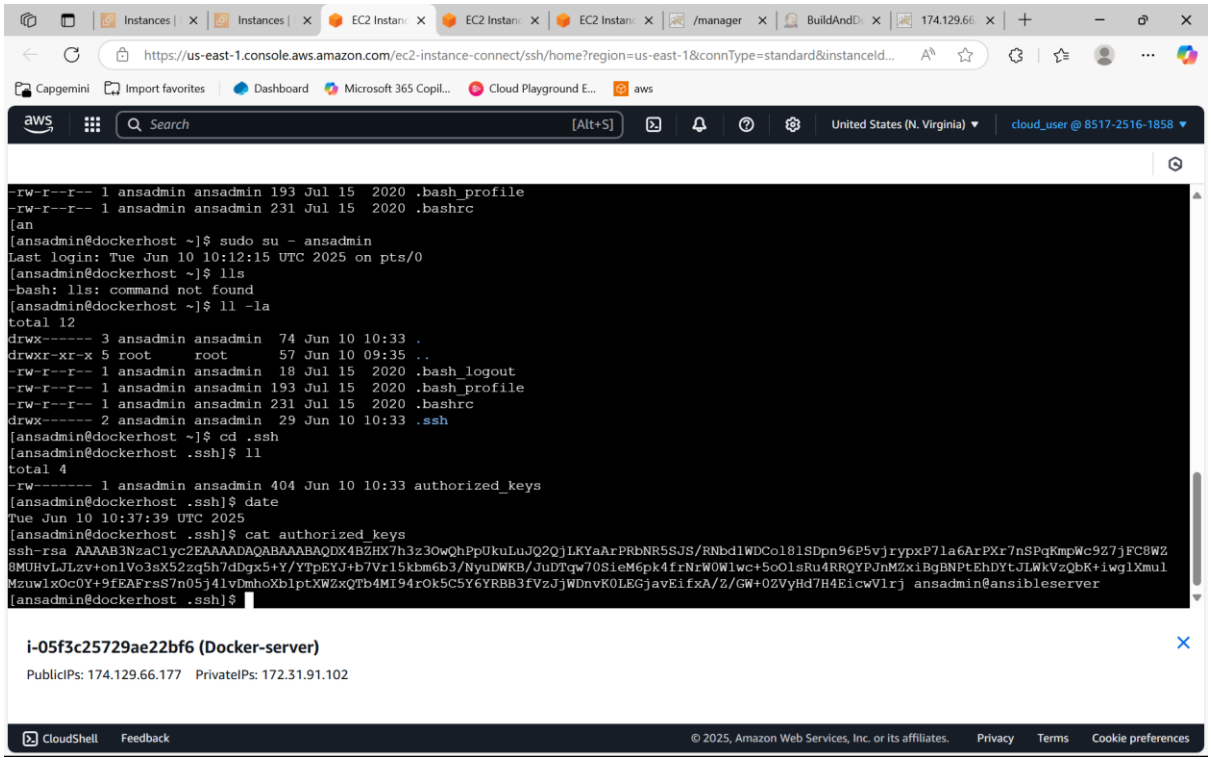


5. Build an Image and Create Container on Ansible

- Write Ansible playbooks that build Docker images from Dockerfiles on specified hosts.
- Automate the creation and startup of containers based on the newly built images.
- Manage container lifecycle through Ansible tasks, including start, stop, and removal.

6. Ansible Playbook to Create Image and Container

- Develop modular playbooks with clear tasks to build Docker images and deploy containers.
- Include steps to pull code, copy artifacts, and configure container environment variables.
- Use handlers and conditional tasks for efficient automation and error handling.



7. Copy Image onto Docker Hub

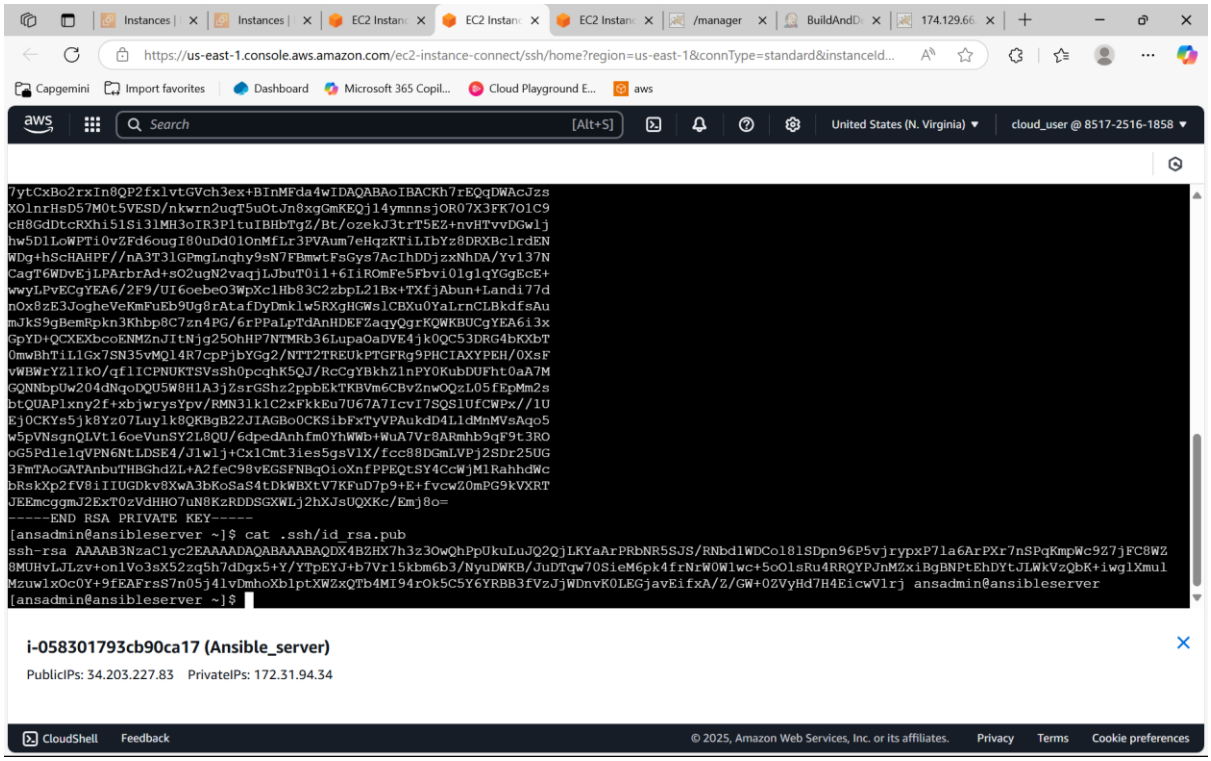
- Automate pushing Docker images to Docker Hub or other container registries using Ansible tasks.
- Manage authentication securely within playbooks or Jenkins credentials.
- Facilitate image distribution and version control across different environments.

8. Jenkins Job to Build an Image onto Ansible

- Configure Jenkins pipelines to invoke Ansible playbooks that build Docker images.
- Use Jenkins environment variables to control playbook behavior and image tags.
- Integrate image build step with source code changes to maintain continuous integration.

9. How to Create Container on Docker Host Using Ansible Playbook – DevOps Project

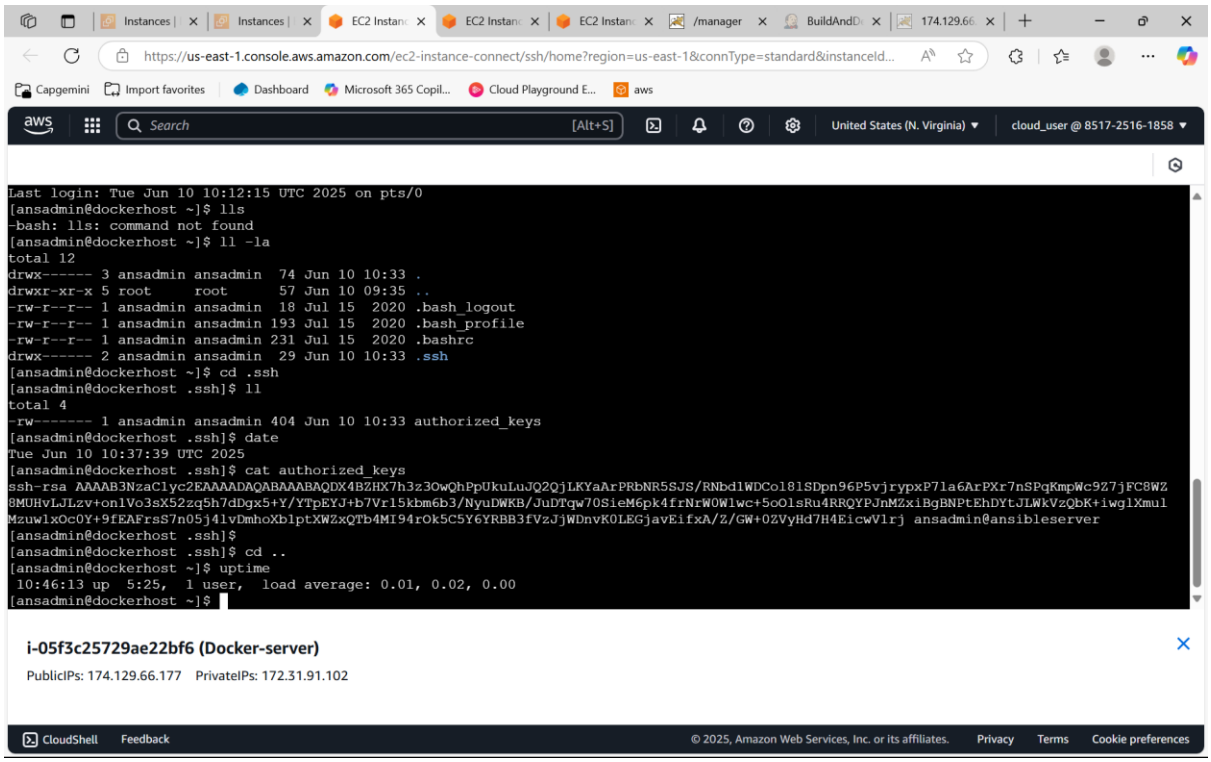
- Use Ansible playbooks to launch containers on remote Docker hosts, ensuring environment consistency.
- Configure network, volumes, and ports in playbooks to match application requirements.
- Validate container status and logs through automated post-deployment checks.



10. Continuous Deployment of Docker Container Using Ansible Playbook

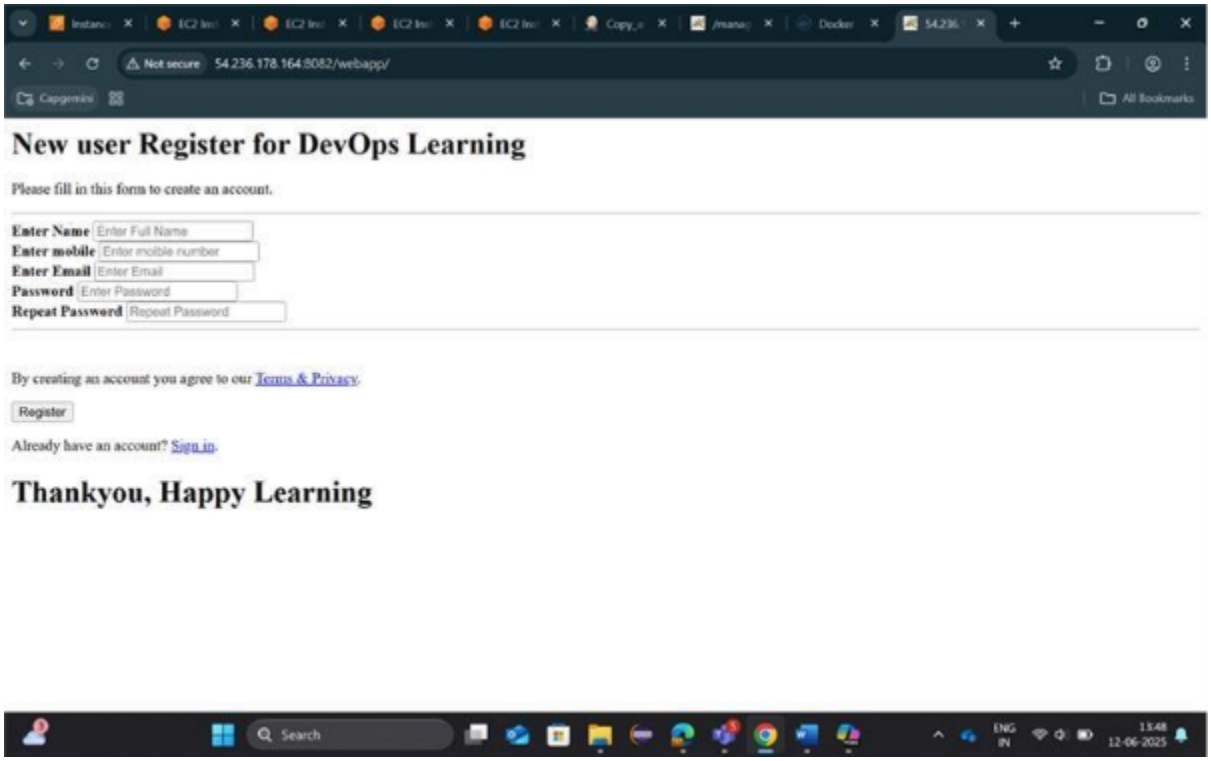
- Set up Ansible playbooks to handle rolling updates, zero-downtime deployments, or blue-green deployment strategies.
- Automate rollback processes in case of failures using Ansible's idempotency features.
- Incorporate monitoring and alerting hooks within the playbooks.
- Ansible executions.

```
[ansadmin@ansibleserver ~]$ cd ..
[ansadmin@ansibleserver home]$ ansible all -m ping
[WARNING]: Platform linux on host 172.31.91.102 is using the discovered Python
interpreter at /usr/bin/python, but future installation of another Python
interpreter could change this. See https://docs.ansible.com/ansible/2.9/referen
ce_appendices/interpreter_discovery.html for more information.
172.31.91.102 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
[ansadmin@ansibleserver home]$ ansible
```



11. Jenkins CI/CD to Deploy on Container Using Ansible

- Combine Jenkins and Ansible for a fully automated pipeline where code commits trigger builds and deployments.
- Use Jenkins to orchestrate the entire process from building code, creating Docker images, and deploying containers via Ansible.
- Ensure pipeline reliability and auditability with logs from both Jenkins and Ansible executions.



4.Kubernetes on AWS

1. Why Use Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps ensure applications are highly available, scalable, and use cloud resources efficiently.

Running Kubernetes on AWS combines the flexibility of Kubernetes with the power of AWS infrastructure. This setup makes it easier to manage large, reliable applications in the cloud.

2. Ways to Install Kubernetes

You can install Kubernetes in different ways, such as:

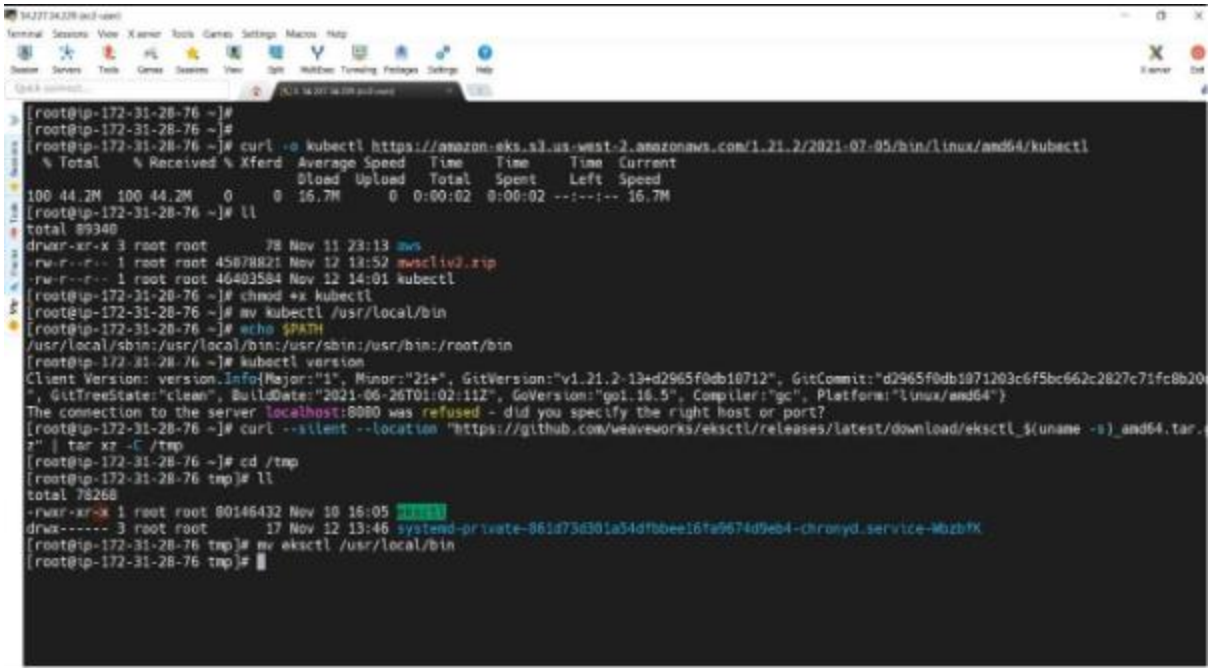
- **kubeadm** – Manual setup for more control
- **minikube** – Best for local testing
- **kops** – Used for setting up Kubernetes on AWS

- **Amazon EKS (Elastic Kubernetes Service)** – A **managed service**, recommended for production environments on AWS because it automates much of the setup and maintenance

3. Installing Kubernetes with Amazon EKS

To set up an EKS cluster:

- Use the AWS Management Console or CLI to create a cluster
- Set up networking (VPC, subnets) and IAM roles
- Launch EC2 worker nodes to run workloads
- Use **kubectl** to manage the cluster by updating the local kubeconfig file



4. Set Up a Bootstrap Server

To use eksctl (a CLI tool for EKS), you need a **bootstrap server**—this could be your local machine or an EC2 instance.

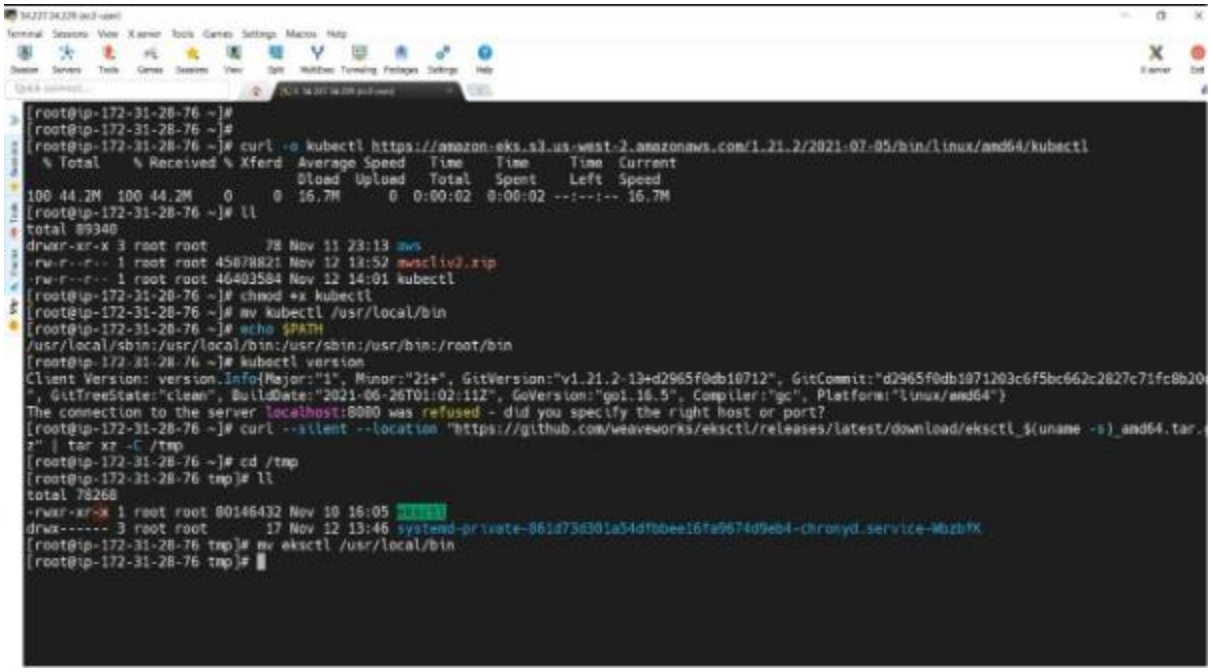
Install the following on it:

- eksctl
- AWS CLI
- kubectl

5. Set Up Kubernetes Using eksctl

Using eksctl simplifies creating a cluster. It automatically handles:

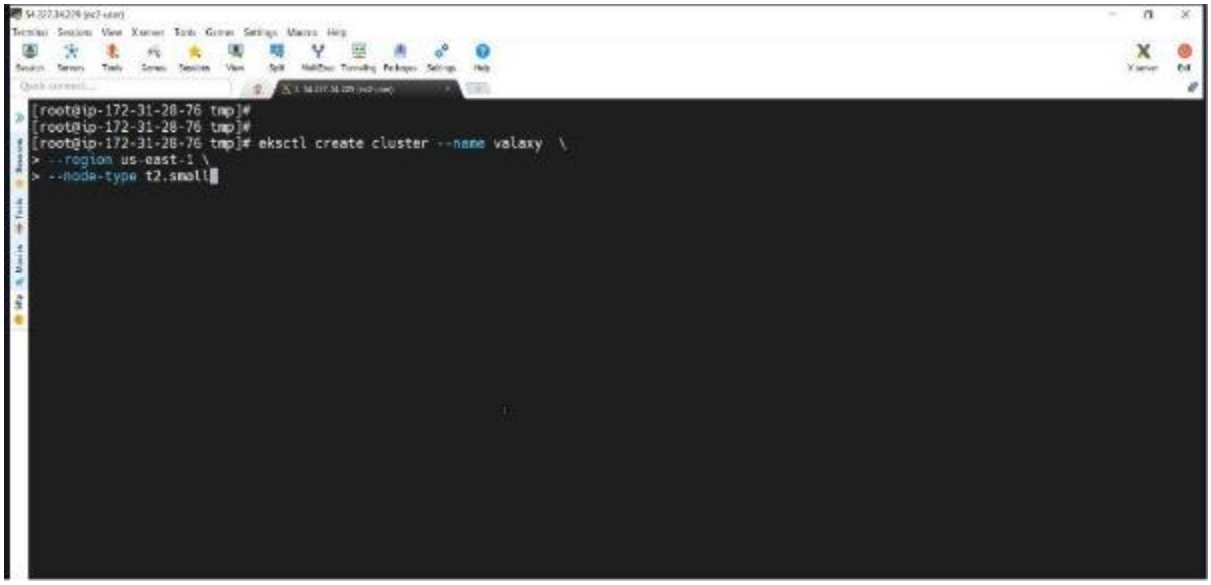
- Creating node groups
- Setting up networking
- Connecting all components



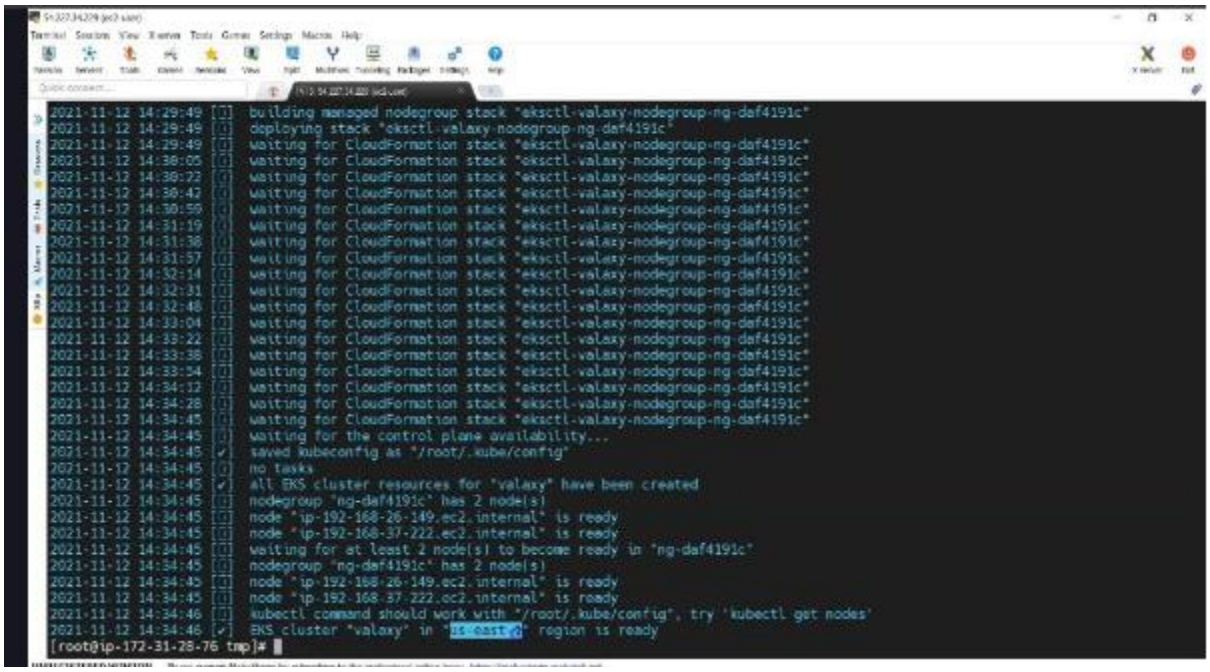
6. Run Basic Kubernetes Commands

Interact with your cluster using kubectl:

- kubectl get nodes – Check running nodes
- kubectl get pods – View application pods
- kubectl describe <resource> – Get details about any resource



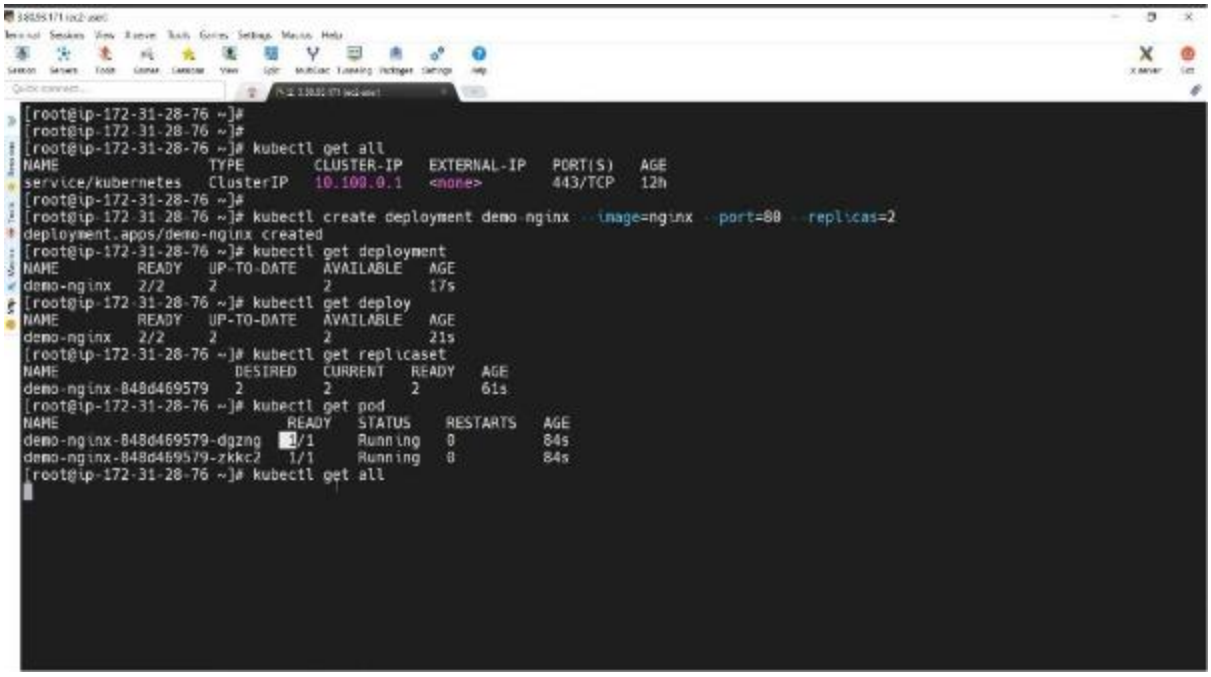
- **7. Create Your First Manifest File**
- Write a **YAML** file that defines your application (Pod, Deployment, etc.).
  For example, create a Pod with a container image and resource limits.



8. Create a Service Manifest

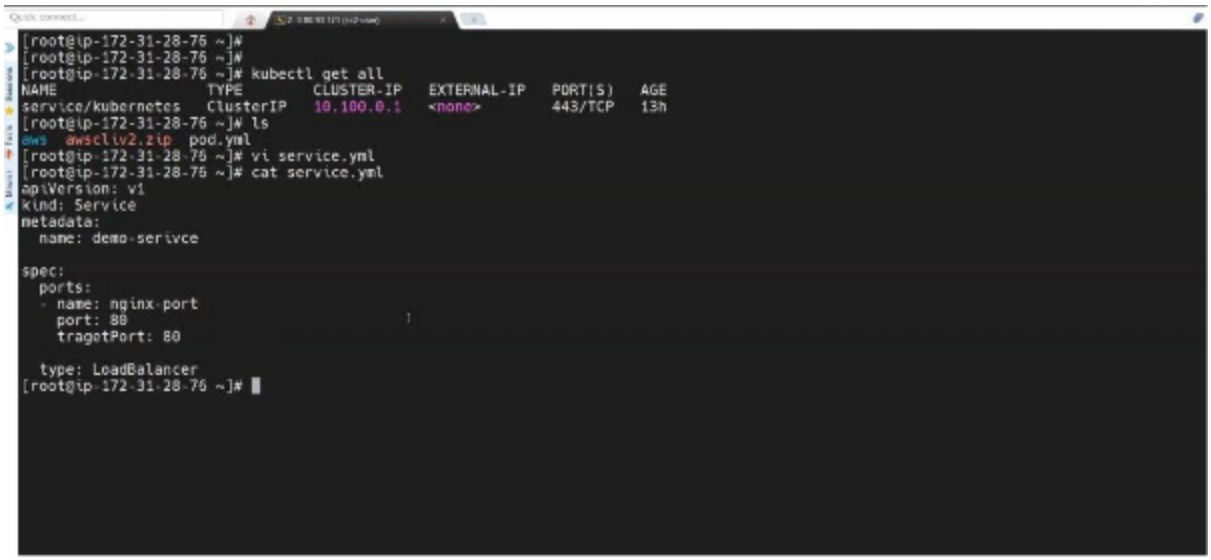
To make your application accessible, create a **Service**. Choose a type:

- ClusterIP – Internal access
- NodePort – Exposes app on a node port
- LoadBalancer – Public access through AWS load balancer



- **9. Labels and Selectors**

-
- Use **labels** (key-value pairs) to tag resources.
  Selectors connect Pods to Services or Deployments based on these labels, allowing better grouping and dynamic management.



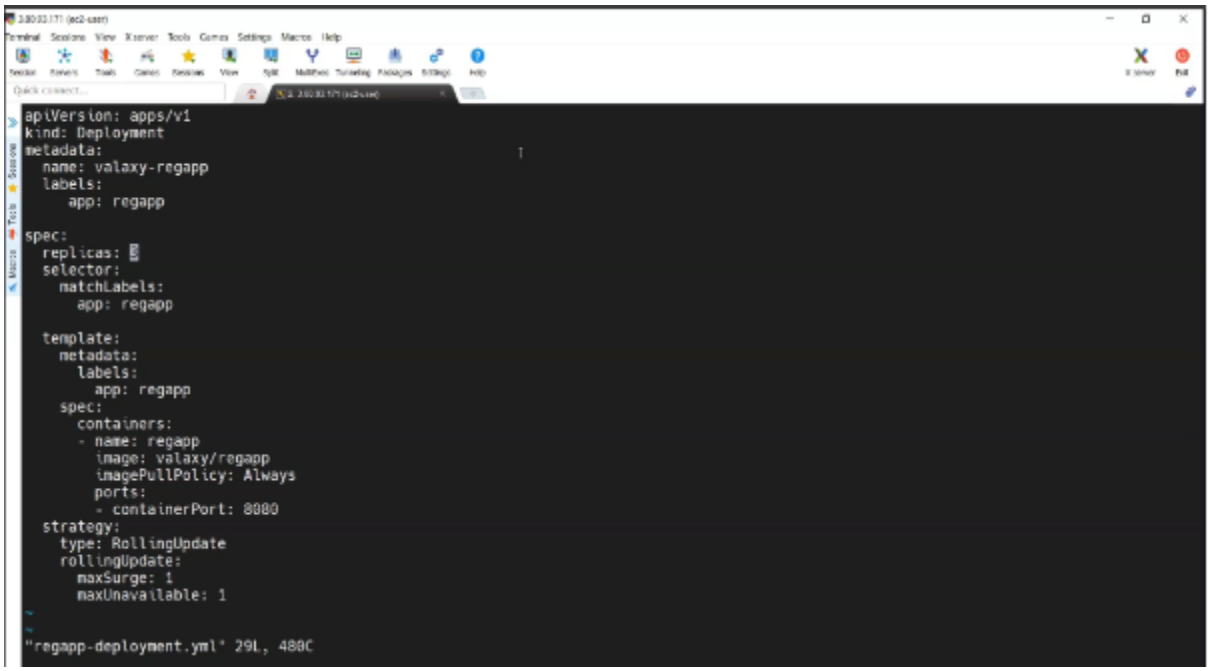
Integrating Kubernetes with CI/CD

1. Write a Deployment File

Create a deployment YAML file defining:

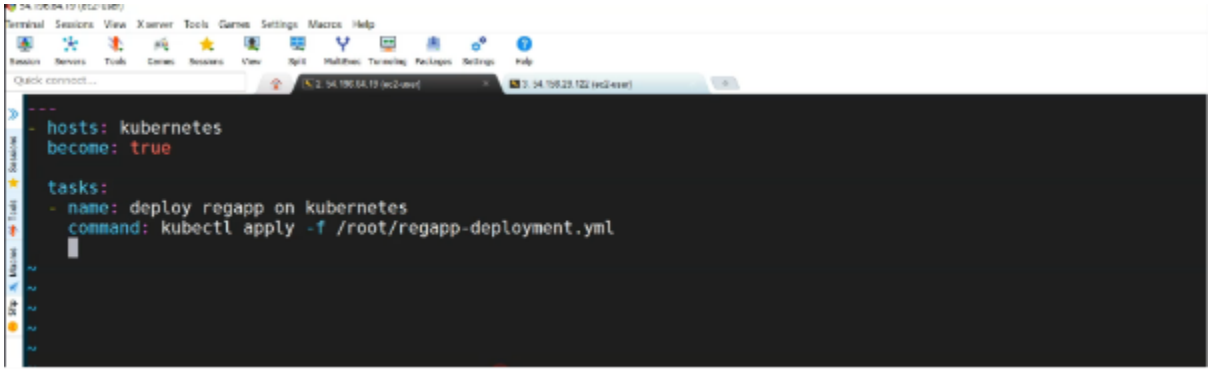
- Container image
- Number of replicas
- Update strategy
- Resource limits

This ensures automatic scaling and reliable app updates.



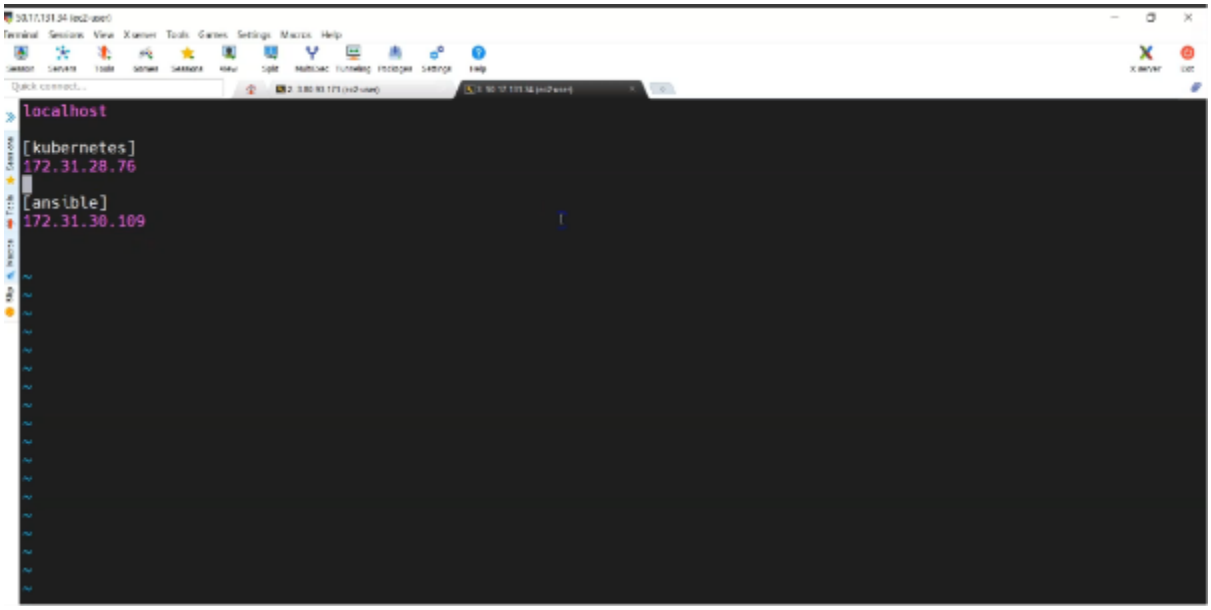
2. Apply Deployment and Service

- Apply deployment and service manifests using kubectl apply -f commands.
- The deployment creates pods running the application, while the service exposes pods to internal or external traffic.
- Validate pod status and service accessibility through Kubernetes commands.



3. Integrate Kubernetes Bootstrap Server with Ansible

- Use Ansible to automate management of Kubernetes bootstrap server where kubectl and cluster tools are configured.
- Playbooks can handle setup tasks such as configuration, authentication, and cluster maintenance.
- Enables automated deployment workflows integrated with infrastructure provisioning.

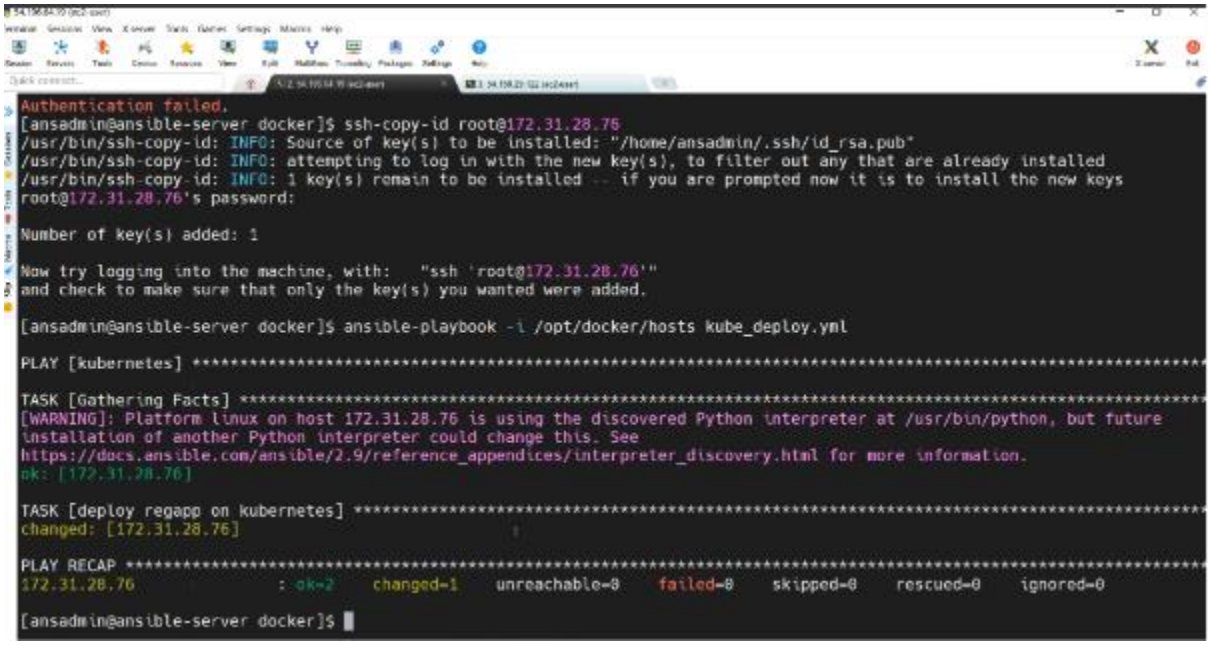


4. Use Ansible Playbooks to Apply Manifests

Create Ansible playbooks that:

- Run kubectl apply for Deployment and Service YAMLs
- Manage app updates and rollbacks

This promotes **Infrastructure as Code** practices.

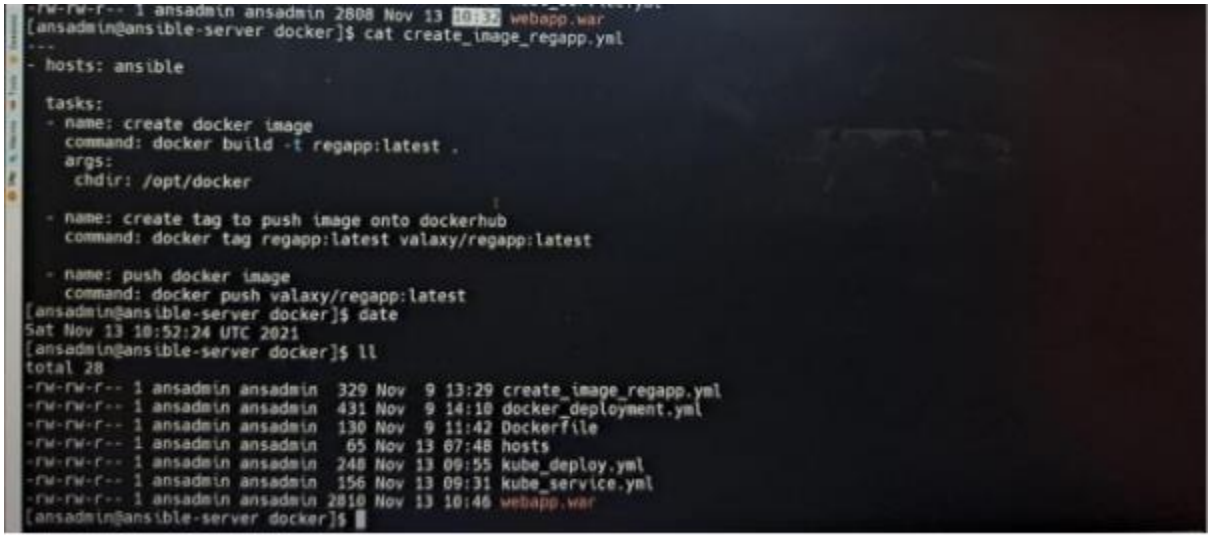


5. Automate Deployment with Jenkins

Use Jenkins to:

- Trigger Ansible playbooks or kubectl commands
- Automate app build, test, and deployment
- Run these pipelines on every code commit or pull request

Monitor success or failure right from Jenkins.



6. Build Docker Image for Kubernetes

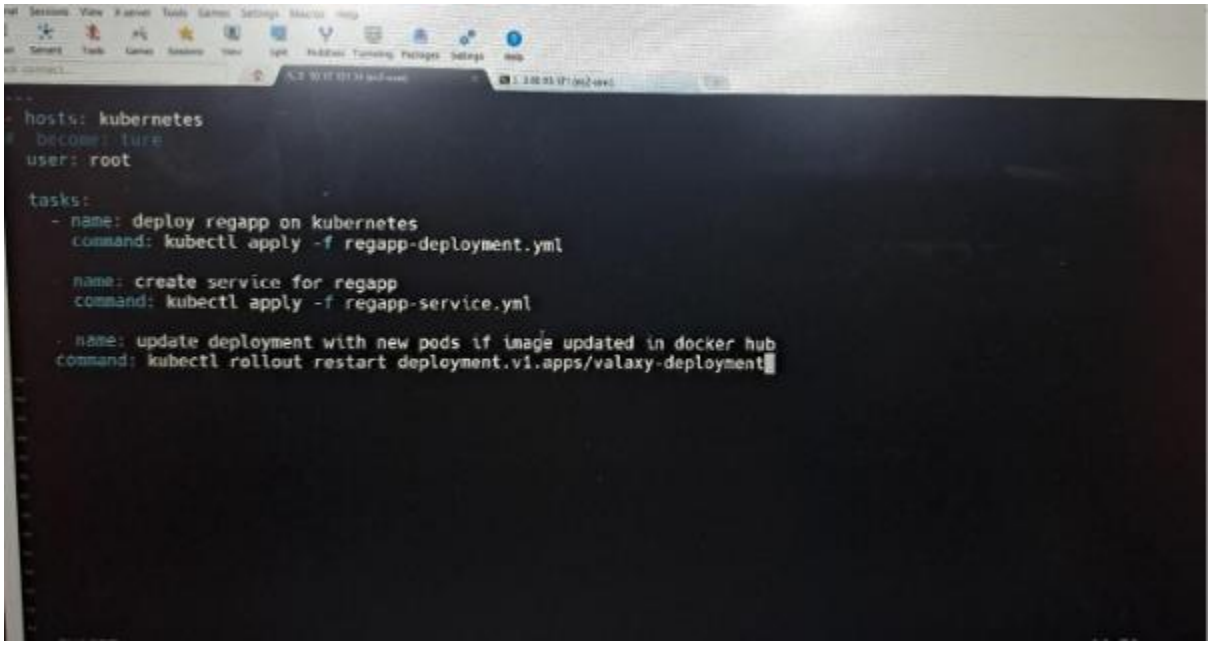
Set up a Jenkins job to:

- Build Docker images from the latest code
- Push them to a container registry (Docker Hub or AWS ECR)

This ensures your Kubernetes deployment always uses the latest version.

7. Enable Rolling Update to Create Pod from Latest Docker Image

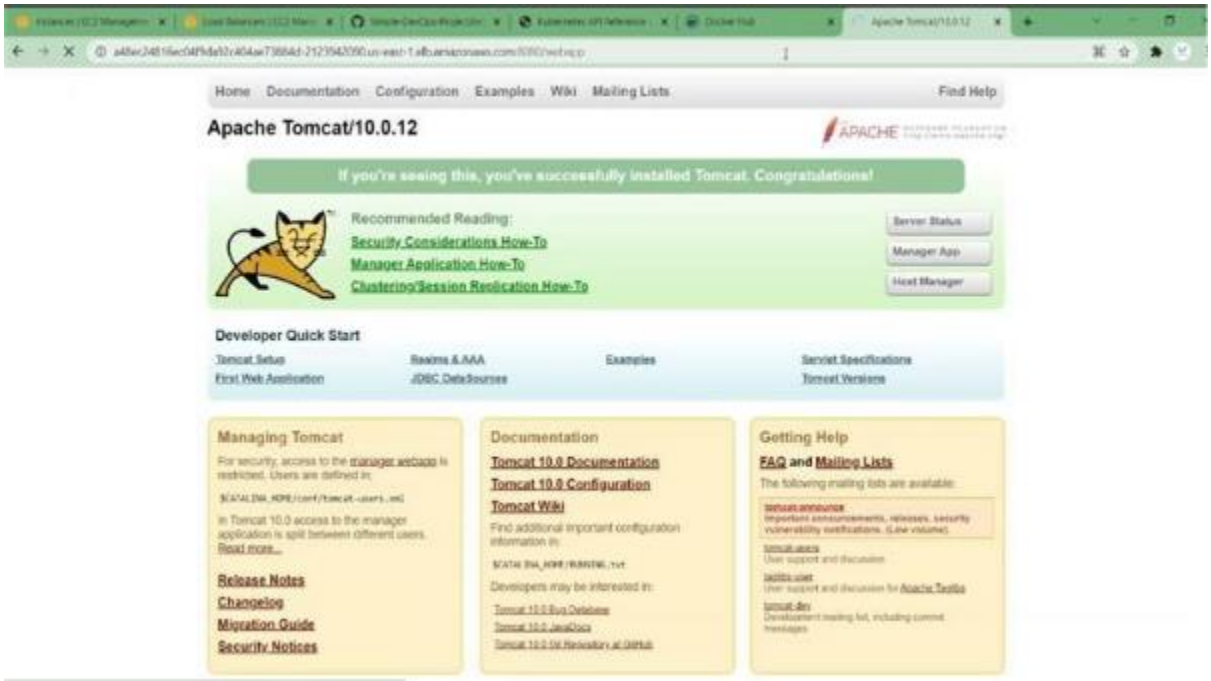
- Configure deployment strategy in Kubernetes manifest to support rolling updates.
- Allows zero downtime deployments by gradually replacing old pods with new ones running updated images.
- Monitors rollout progress and automatically rolls back on failures.



8. End-to-End CI/CD Pipeline

Combine all CI/CD steps:

- Build and test code
- Create Docker image
- Deploy to Kubernetes
- Roll out updates automatically



9. Clean Up Resources

Use Jenkins or Ansible to:

- Remove unused Pods, Services, and Deployments
- Tear down test clusters when no longer needed

```
[root@ip-172-31-25-112 /]# cd tmp/
[root@ip-172-31-25-112 tmp]# ll
total 16
-rw------- 1 root root   0 Jun 13 07:29 62291b61.eksctl.lock
-rw-r--r-- 1 root root 203 Jun 13 07:34 pod.yml
-rw-r--r-- 1 root root 481 Jun 13 07:36 regapp-deploy.yml
-rw-r--r-- 1 root root 197 Jun 13 07:36 regapp-service.yml
-rw-r--r-- 1 root root 183 Jun 13 07:34 service.yml
drwx------ 3 root root  17 Jun 13 07:08 systemd-private-72ab7dd853ea47c9be9b90aefc2d7ea1-chronyd.service-pgkTY6
[root@ip-172-31-25-112 tmp]# ls
62291b61.eksctl.lock  regapp-deploy.yml   service.yml
pod.yml               regapp-service.yml  systemd-private-72ab7dd853ea47c9be9b90aefc2d7ea1-chronyd.service-pgkTY6
[root@ip-172-31-25-112 tmp]# kubectl delete -f regapp-service.yml
service "valaxy-service" deleted
[root@ip-172-31-25-112 tmp]# kubectl delete -f regapp-deploy.yml
deployment.apps "valaxy-regapp" deleted
[root@ip-172-31-25-112 tmp]#
```

Conclusion

This case study successfully demonstrates the implementation of a fully automated DevOps CI/CD pipeline using widely adopted tools such as Git, Jenkins, Maven, Docker, Ansible, and Kubernetes on AWS infrastructure. By integrating these technologies, the pipeline enabled continuous integration and continuous deployment, significantly improving the speed, reliability, and efficiency of software delivery with minimal manual intervention. The process began with Jenkins automating build and test tasks triggered by code commits to Git, followed by Maven handling Java project builds, and Docker containerizing the applications for consistent deployment. Ansible further streamlined configuration management, while Kubernetes—deployed via Amazon EKS—provided scalable orchestration and seamless rollout of application updates. This automation-first approach allowed early detection of bugs, reduced deployment errors, and supported rolling updates, thereby enhancing software quality and accelerating release cycles. Leveraging AWS cloud services and infrastructure-as-code practices ensured scalability, flexibility, and consistency across environments. Overall, this project highlights the tangible benefits of adopting DevOps methodologies and CI/CD pipelines in modern software development. Future improvements could include integrating advanced monitoring tools, enabling multi-cloud deployment, and adopting more sophisticated rollback and deployment strategies to further optimize the delivery process.