Capgemini DevOps Internship Project Report: CI/CD Pipeline

Prepared by: SANGEKARI SHIVANAGARAJU

Overview

In this project, I built a CI/CD pipeline for a Java web application. The main goal was to automate the entire process, from coding to deployment, using key DevOps tools.

The project started with storing the application code on GitHub for version control. A Jenkins server, hosted on an AWS computer, was set up to automatically build the application when new code was pushed. For the build process, Maven was used to package the Java code.

After the build, the application was containerized using Docker. This is a crucial step because it ensures the app runs the same way on any machine. To automate the deployment, I used Ansible to configure the servers and run the Docker container. Finally, the environment was prepared for Kubernetes (Amazon EKS) to show how the application could be managed at scale. This automated setup resulted in a smooth, reliable, and easy-to-manage process.

Technology Used

- **Cloud Platform:** Amazon Web Services (AWS) EC2
- **Version Control:** Git & GitHub
- **CI/CD Automation Server:** Jenkins
- **Build Tool:** Apache Maven
- **Application Server:** Apache Tomcat
- **Configuration Management:** Ansible
- **Containerization:** Docker
- **Container Registry:** Docker Hub
- **Orchestration:** Kubernetes (Amazon EKS)
- **SSH Client:** MobaXterm

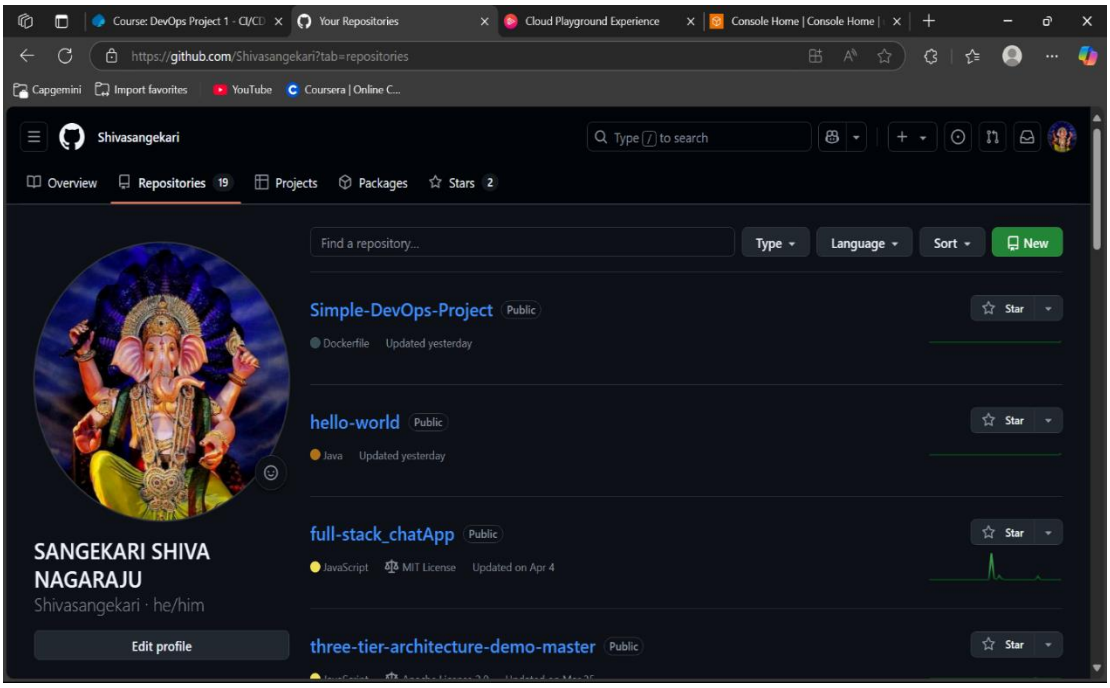
Implementation

The project was implemented by following these steps.

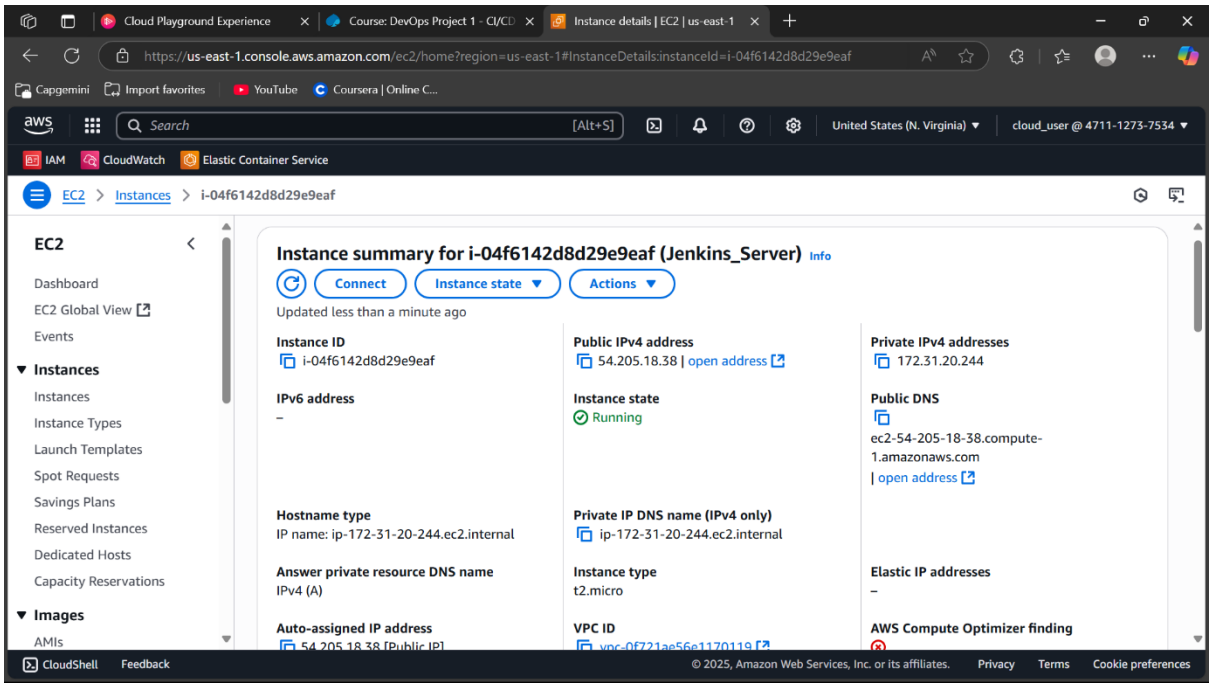
Step 1: Initial Setup of Jenkins and Source Code

The first stage involved getting the basic tools and environment ready.

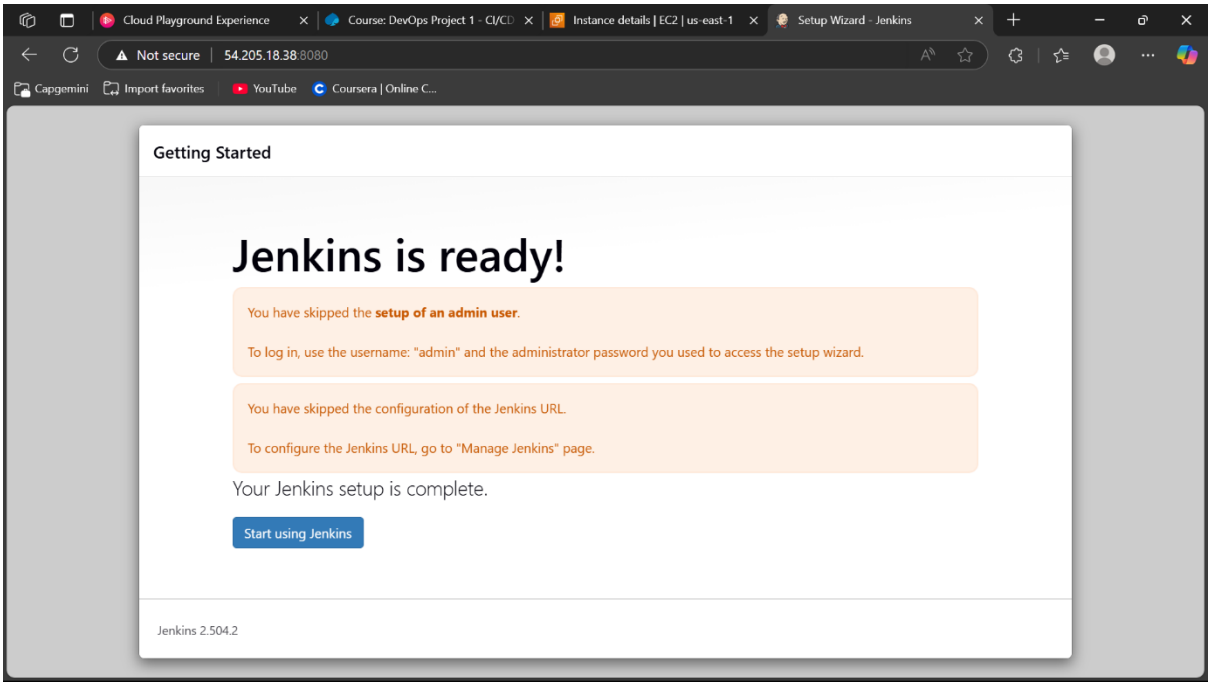
1. **GitHub Repository:** I used my GitHub account to create a repository named hello-world for storing the Java application code.



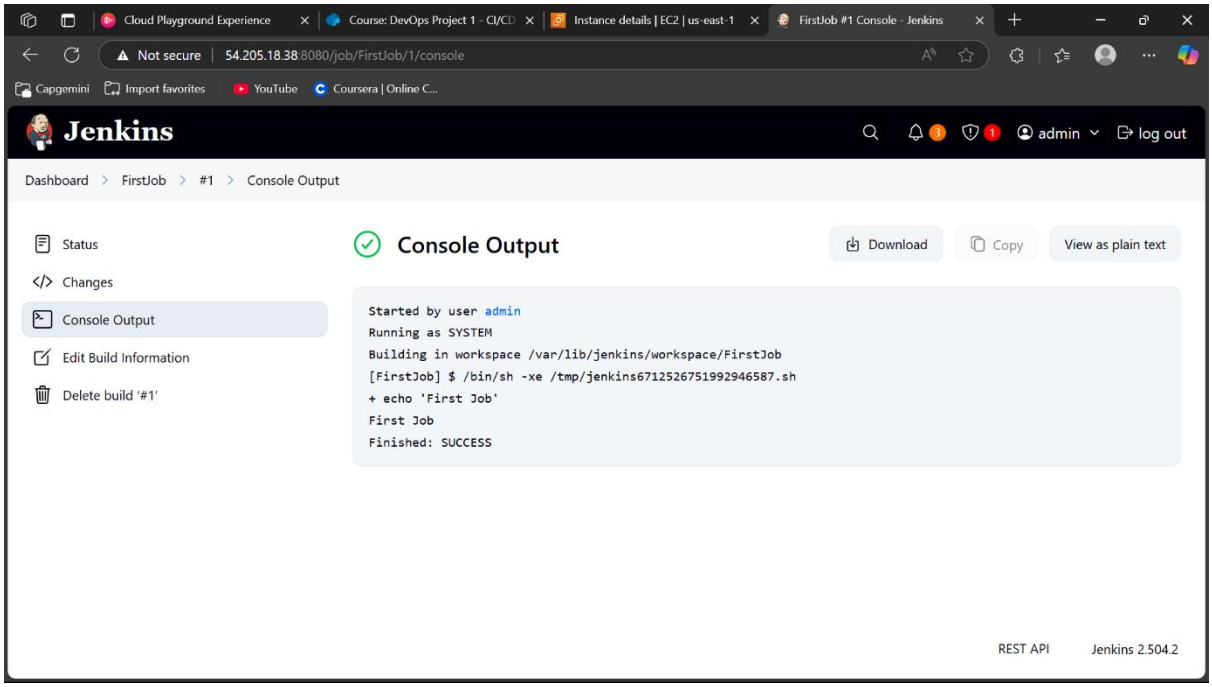
2. **Jenkins Server on AWS:** An AWS EC2 instance (t2.micro) was launched to host the Jenkins server. The security group was configured to allow inbound traffic on port 8080 for the Jenkins web UI and port 22 for SSH access.



3. **Jenkins Installation:** After connecting to the server with MobaXterm, Java and Jenkins were installed. I started the Jenkins service and checked its status to confirm it was running. The final setup steps were then completed in a web browser.



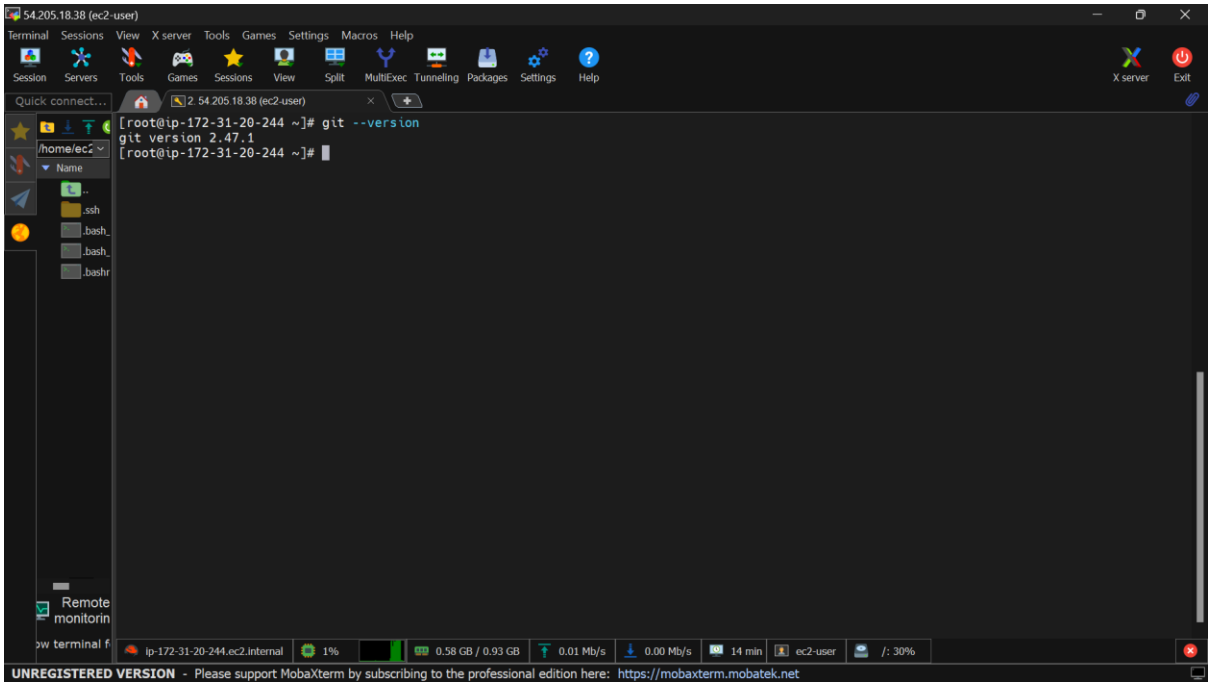
4. **First Jenkins Job:** To test that the Jenkins installation was successful, I created a simple Freestyle job called FirstJob. This job just ran a basic echo command. The successful output confirmed Jenkins was working correctly.

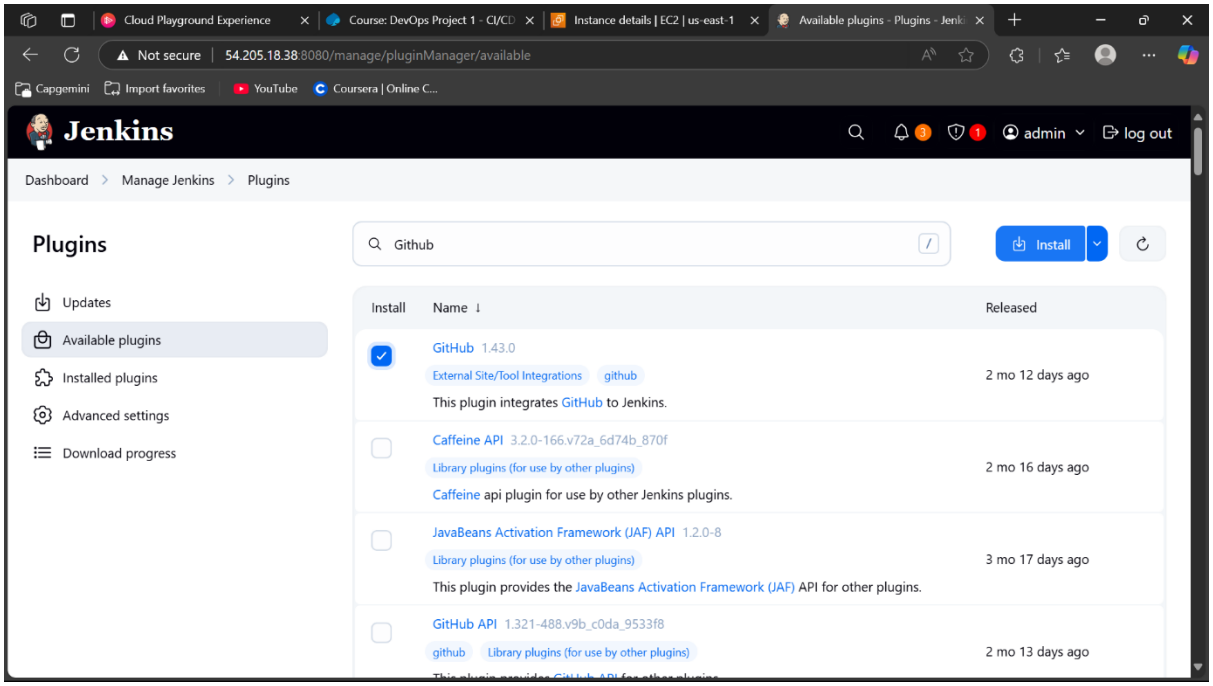


Step 2: Automating the Application Build

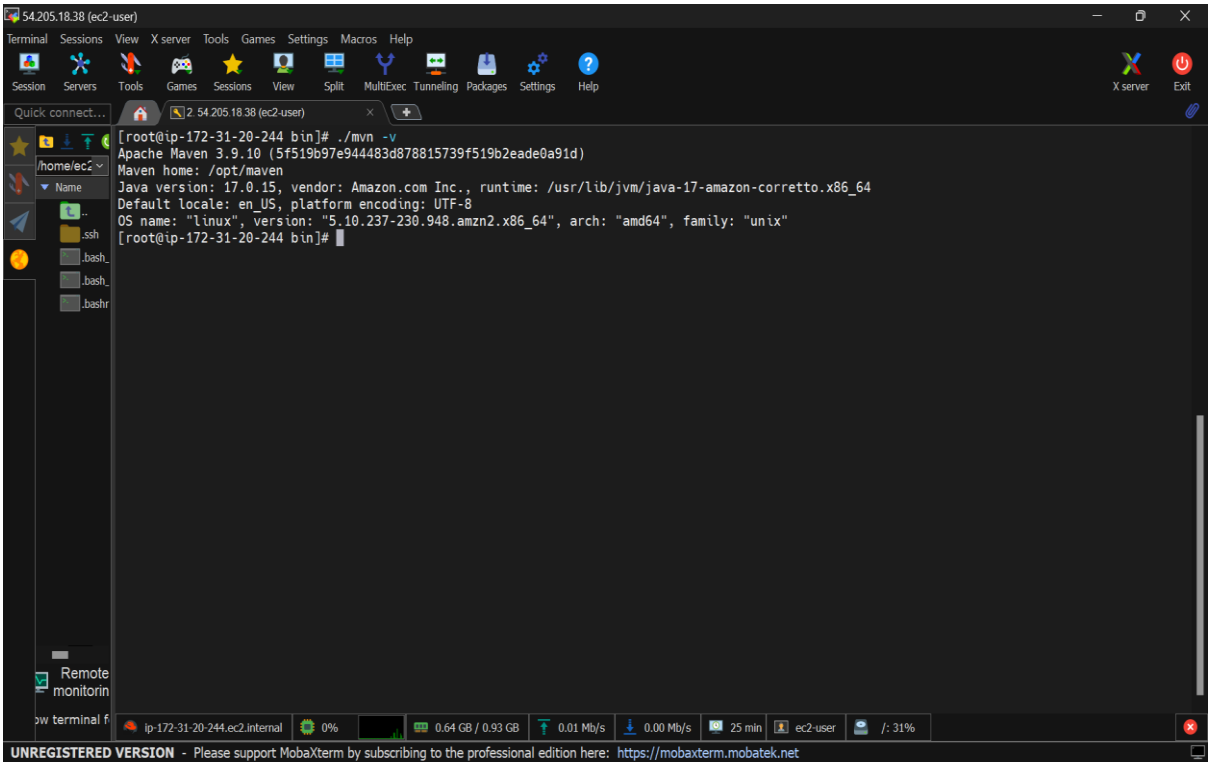
The next step was to configure Jenkins to build the application from the source code automatically.

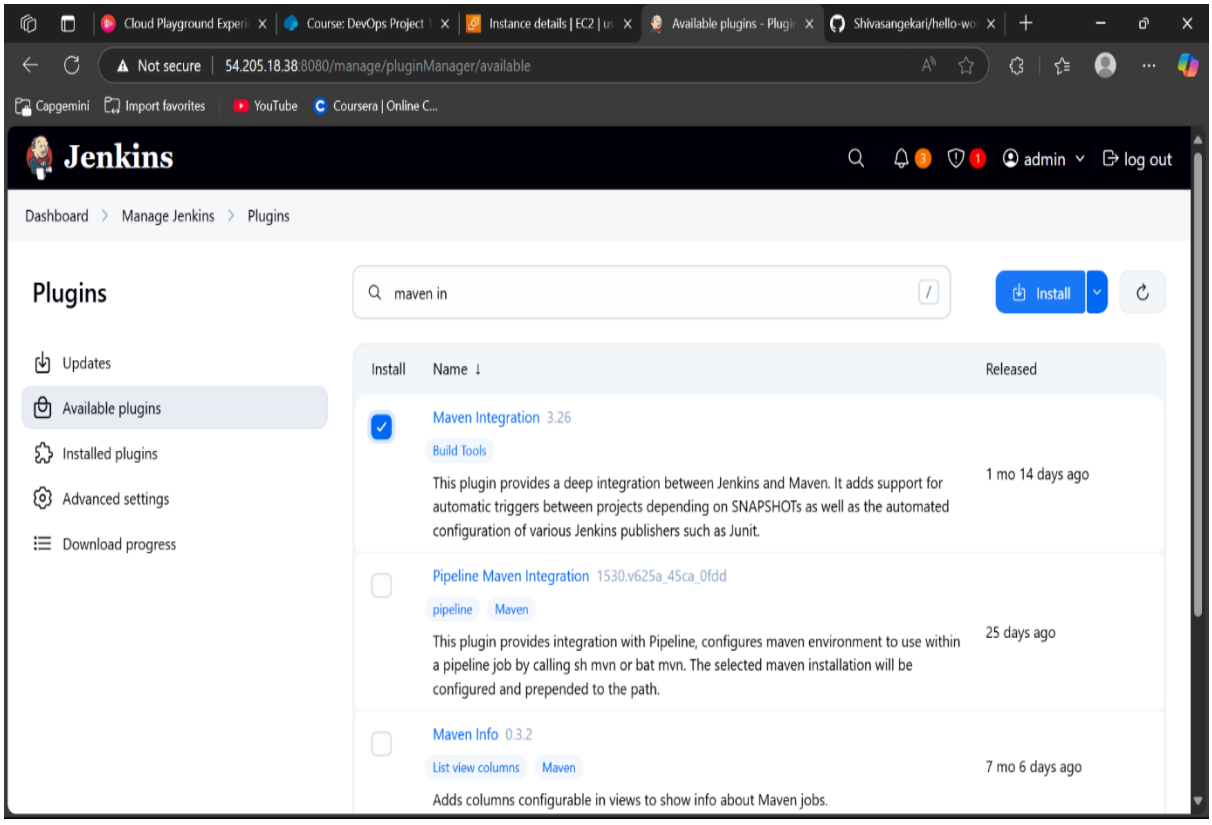
1. **Tool Installation:** Git was installed on the Jenkins server. From the Jenkins dashboard, the **GitHub** and **Maven Integration** plugins were added to extend its functionality.

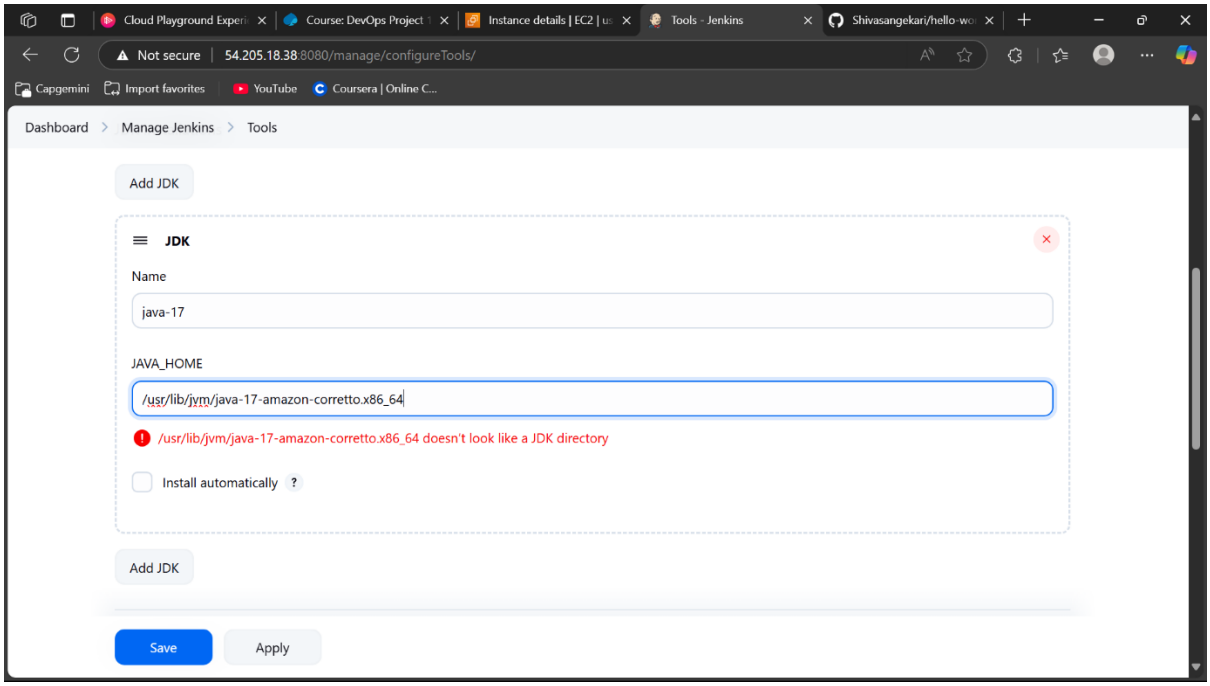




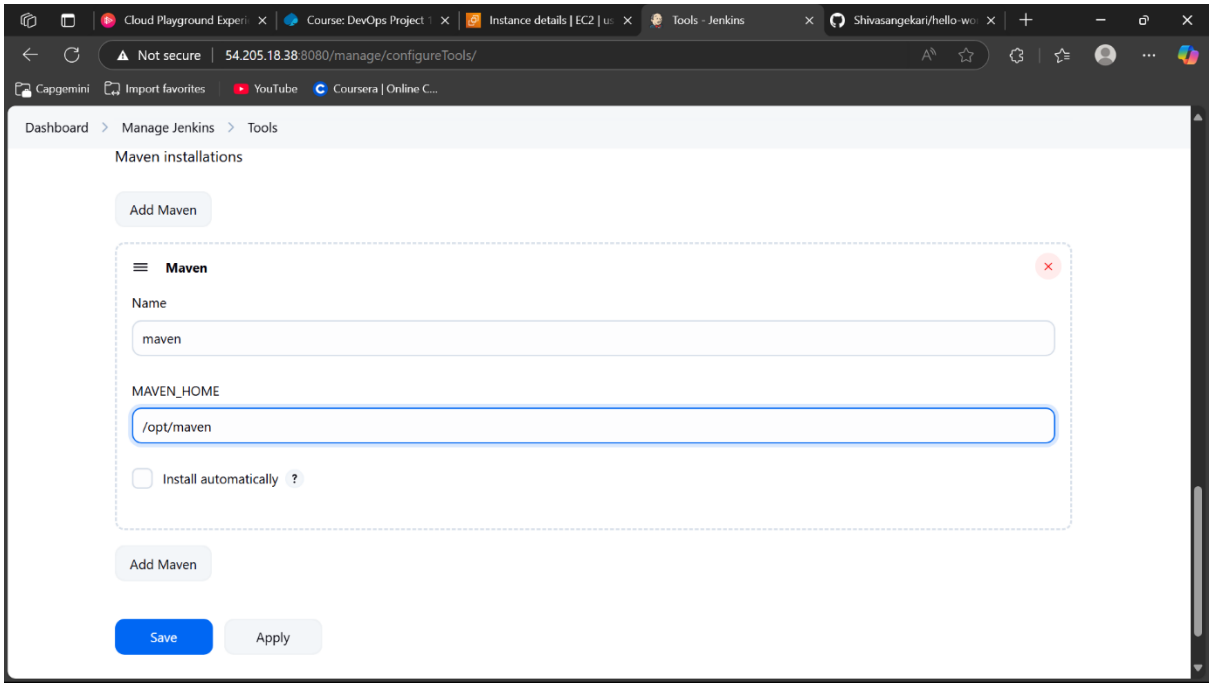
2. **Java and Maven Configuration:** In the Jenkins 'Global Tool Configuration' settings, I added the paths for the Java 17 and Maven installations. This allows Jenkins to find and use these tools during the build process.

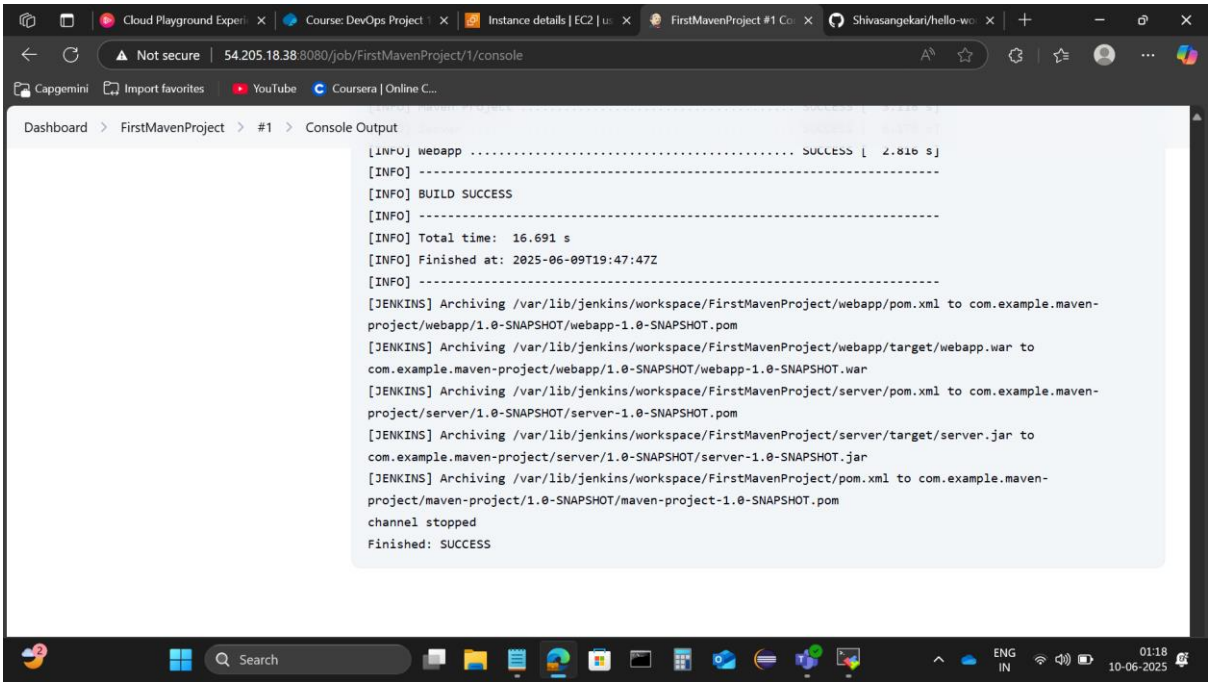

54.205.18.38 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
2. 54.205.18.38 (ec2-user)
[root@ip-172-31-20-244 bin]# ./mvn -v
Apache Maven 3.9.10 (5f519b97e944483d878815739f519b2eade0a91d)
Maven home: /opt/maven
Java version: 17.0.15, vendor: Amazon.com Inc., runtime: /usr/lib/jvm/java-17-amazon-corretto.x86_64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.10.237-230.948.amzn2.x86_64", arch: "amd64", family: "unix"
[root@ip-172-31-20-244 bin]#
/home/ec2
Name
.ssh
.bash_
.bash_
.bashr
Remote monitorin
ow terminal f
ip-172-31-20-244.ec2.internal
0%
0.64 GB / 0.93 GB
0.01 Mb/s
0.00 Mb/s
25 min
ec2-user
/: 31%
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

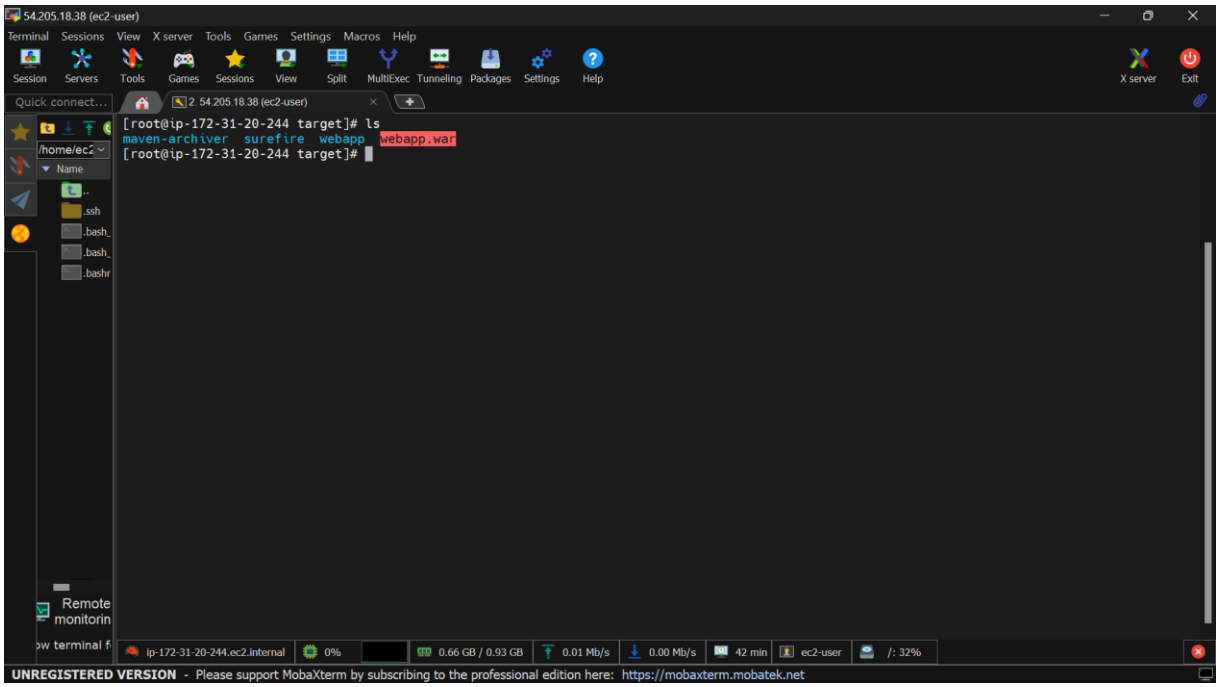

Cloud Playground Experi
Course: DevOps Project
Instance details | EC2 | us
Available plugins - Plugi
Shivasangekari/hello-wo
Not secure
54.205.18.38:8080/manage/pluginManager/available
Capgemini
Import favorites
YouTube
Coursera | Online C...
Jenkins
admin
log out
Dashboard
Manage Jenkins
Plugins
Plugins
maven in
Install
Updates
Available plugins
Installed plugins
Advanced settings
Download progress
Install
Name
Released
Maven Integration 3.26
Build Tools
This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTs as well as the automated configuration of various Jenkins publishers such as Junit.
1 mo 14 days ago
Pipeline Maven Integration 1530.v625a_45ca_0fdd
pipeline
Maven
This plugin provides integration with Pipeline, configures maven environment to use within a pipeline job by calling sh mvn or bat mvn. The selected maven installation will be configured and prepended to the path.
25 days ago
Maven Info 0.3.2
List view columns
Maven
Adds columns configurable in views to show info about Maven jobs.
7 mo 6 days ago



3. **Building the Project with Maven:** A new 'Maven project' was created in Jenkins. It was configured to pull code from the GitHub repository and execute the clean install Maven command. The job ran successfully, producing a webapp.war file as the build artifact.



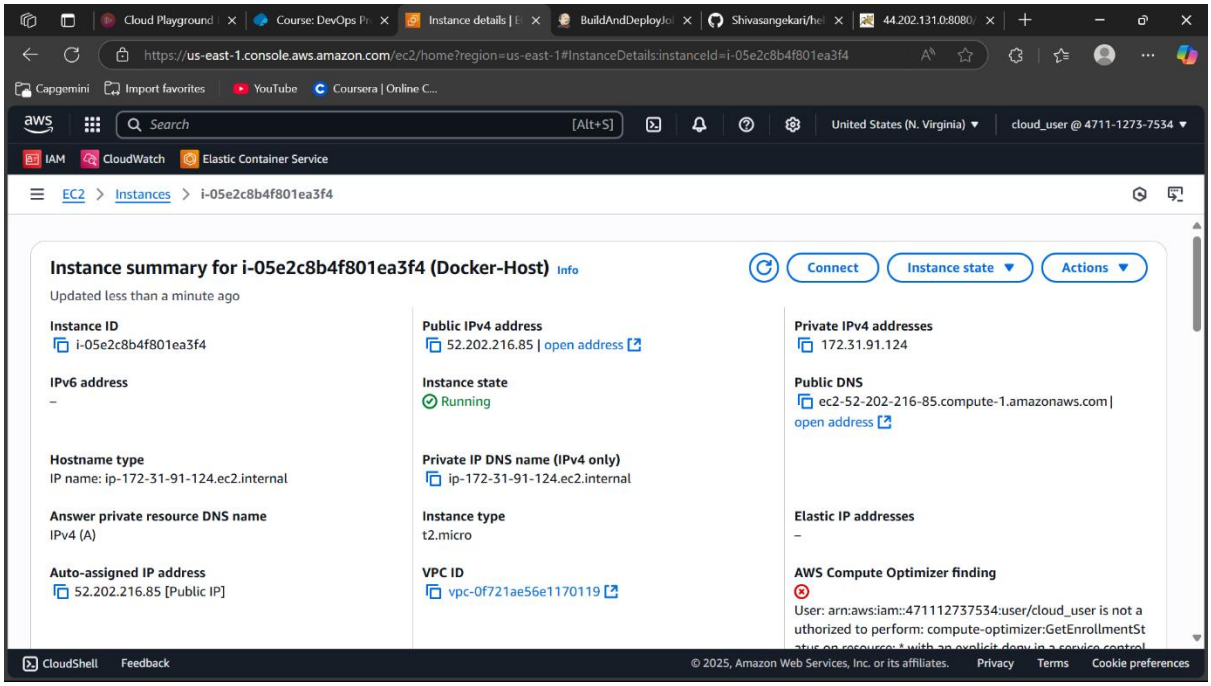


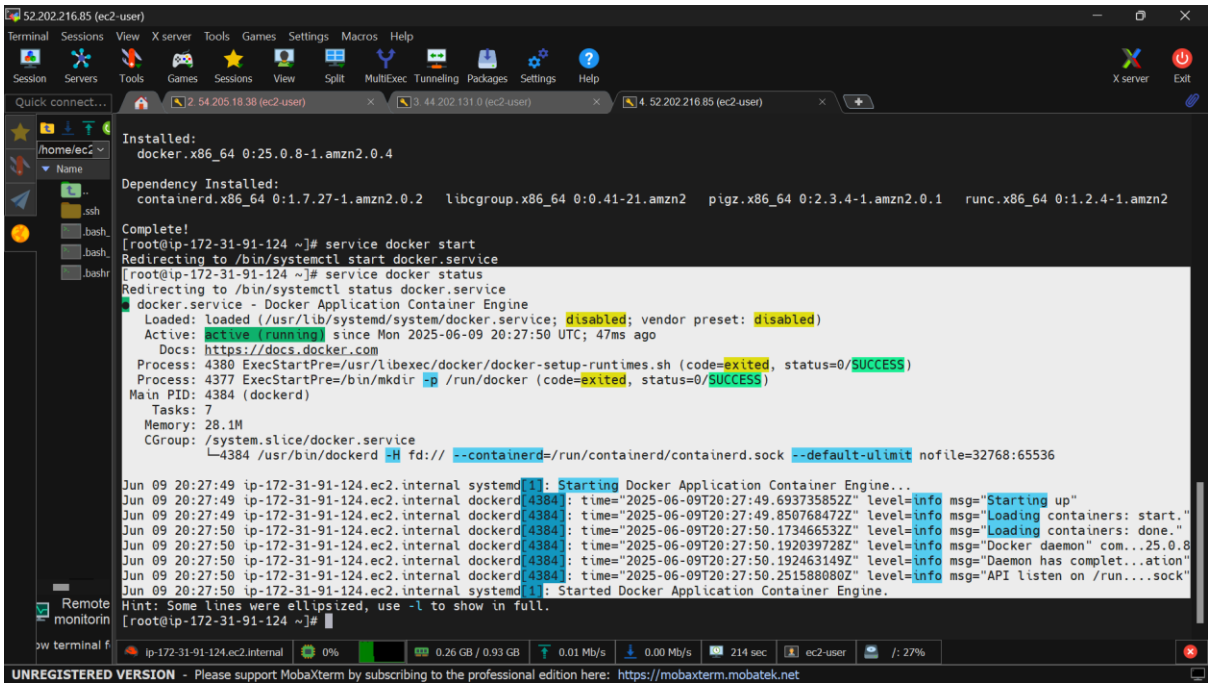


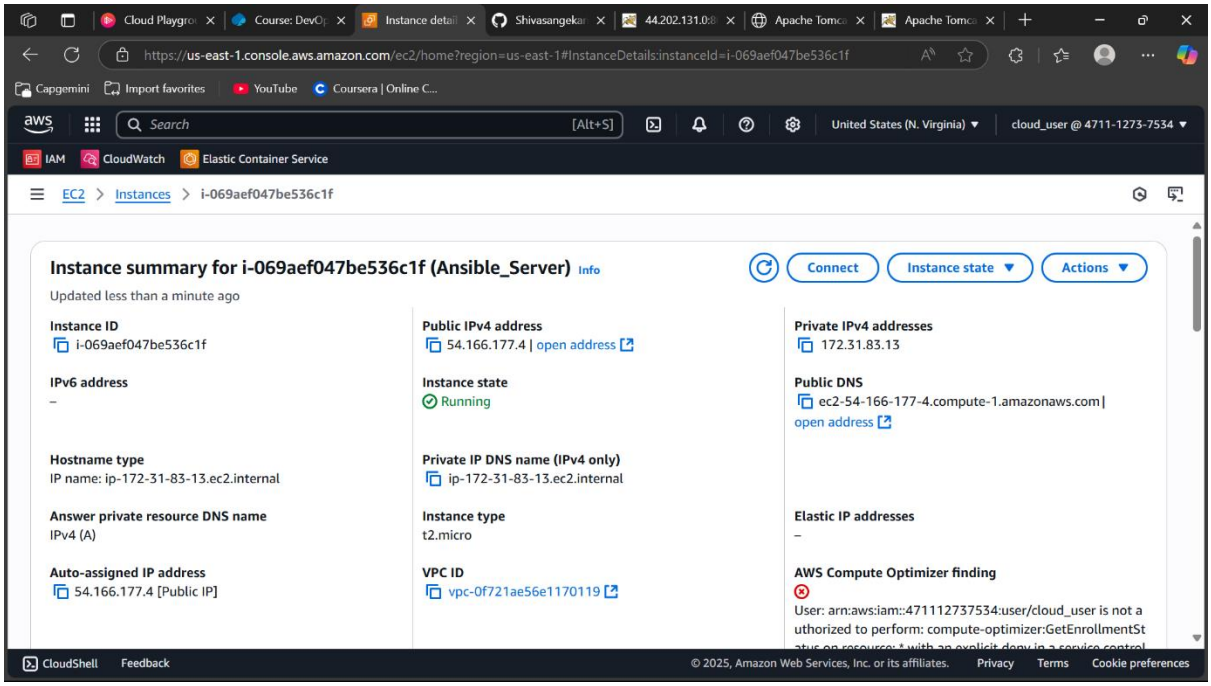
Step 3: Deploying the Application with Docker and Ansible

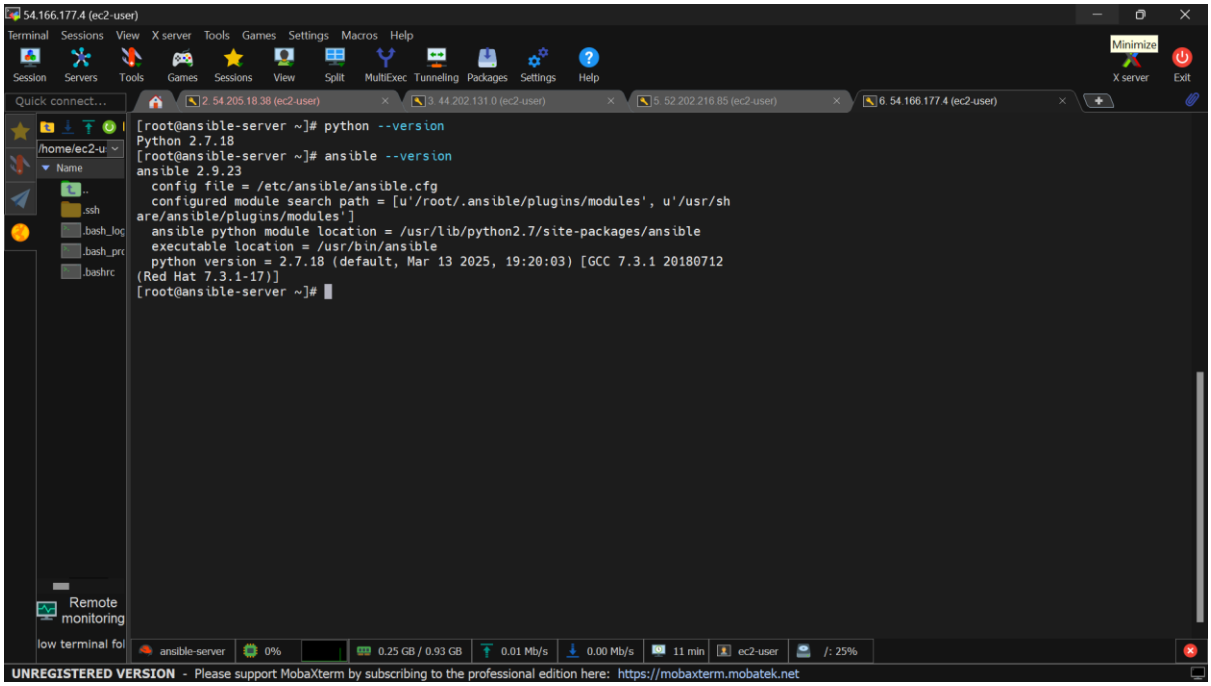
This stage focused on containerizing the built application and automating its deployment.

1. **Docker and Ansible Server Setup:** Two new EC2 instances were launched: one to serve as a **Docker Host** and the other as an **Ansible Controller**. Docker was installed on the Docker Host, while Ansible was installed on the Ansible Controller.

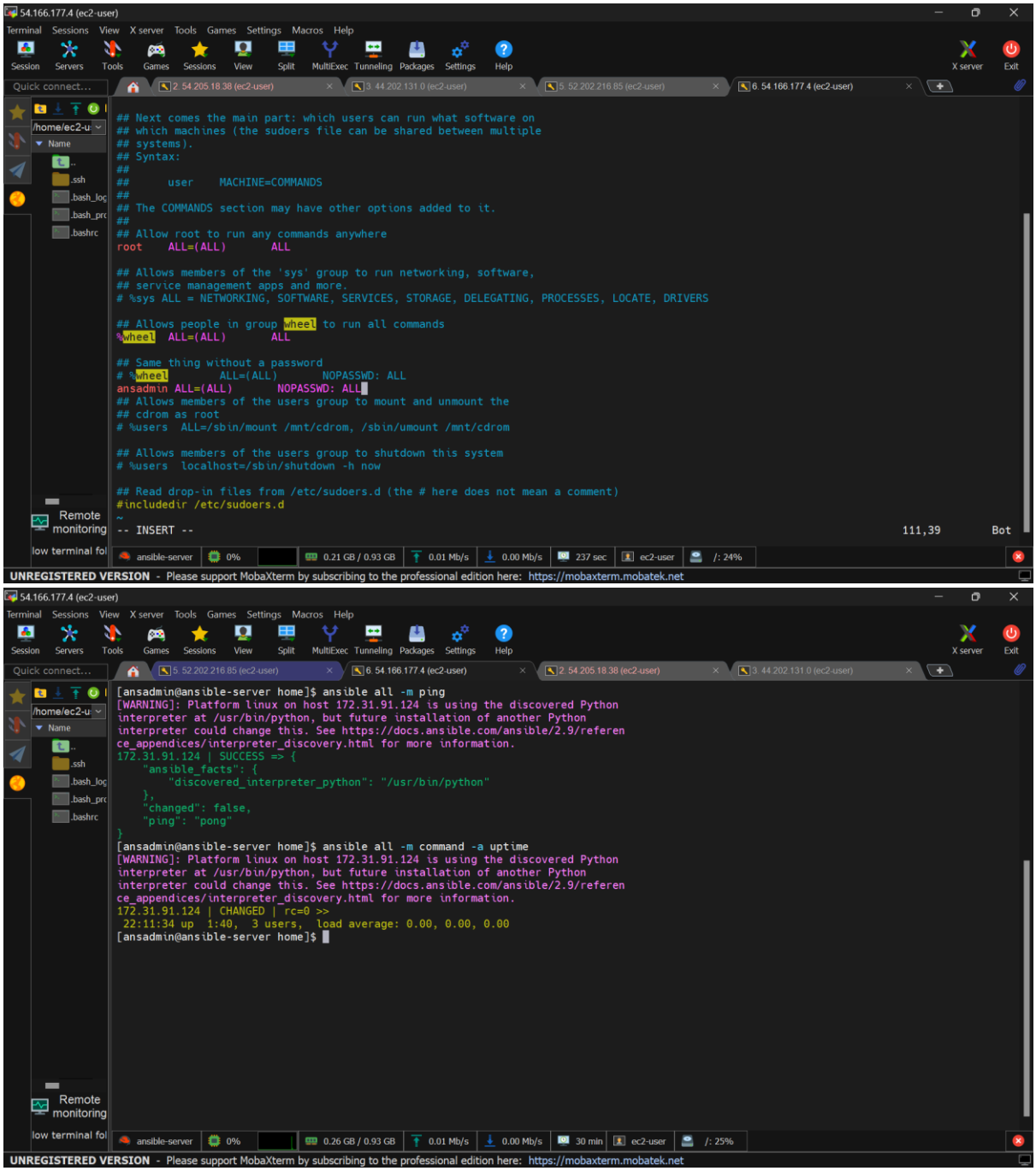

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#InstanceDetails:instanceId=i-05e2c8b4f801ea3f4
EC2 > Instances > i-05e2c8b4f801ea3f4
Instance summary for i-05e2c8b4f801ea3f4 (Docker-Host) Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-05e2c8b4f801ea3f4
Public IPv4 address
52.202.216.85 | open address
Private IPv4 addresses
172.31.91.124
IPv6 address
–
Instance state
Running
Public DNS
ec2-52-202-216-85.compute-1.amazonaws.com | open address
Hostname type
IP name: ip-172-31-91-124.ec2.internal
Private IP DNS name (IPv4 only)
ip-172-31-91-124.ec2.internal
Answer private resource DNS name
IPv4 (A)
Instance type
t2.micro
Elastic IP addresses
–
Auto-assigned IP address
52.202.216.85 [Public IP]
VPC ID
vpc-0f721ae56e1170119
AWS Compute Optimizer finding
User: arn:aws:iam::471112737534:user/cloud_user is not authorized to perform: compute-optimizer:GetEnrollmentStatus on resource: * with an explicit deny in a service control
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


52.202.216.85 (ec2-user)
Installed:
docker.x86_64 0:25.0.8-1.amzn2.0.4
Dependency Installed:
containerd.x86_64 0:1.7.27-1.amzn2.0.2 libcgroup.x86_64 0:0.41-21.amzn2 pigz.x86_64 0:2.3.4-1.amzn2.0.1 runc.x86_64 0:1.2.4-1.amzn2
Complete!
[root@ip-172-31-91-124 ~]# service docker start
Redirecting to /bin/systemctl start docker.service
[root@ip-172-31-91-124 ~]# service docker status
Redirecting to /bin/systemctl status docker.service
● docker.service - Docker Application Container Engine
Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
Active: active (running) since Mon 2025-06-09 20:27:50 UTC; 47ms ago
Docs: https://docs.docker.com
Process: 4380 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
Process: 4377 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
Main PID: 4384 (dockerd)
Tasks: 7
Memory: 28.1M
CGroup: /system.slice/docker.service
└4384 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
Jun 09 20:27:49 ip-172-31-91-124.ec2.internal systemd[1]: Starting Docker Application Container Engine...
Jun 09 20:27:49 ip-172-31-91-124.ec2.internal dockerd[4384]: time="2025-06-09T20:27:49.693735852Z" level=info msg="Starting up"
Jun 09 20:27:49 ip-172-31-91-124.ec2.internal dockerd[4384]: time="2025-06-09T20:27:49.850768472Z" level=info msg="Loading containers: start."
Jun 09 20:27:50 ip-172-31-91-124.ec2.internal dockerd[4384]: time="2025-06-09T20:27:50.173466532Z" level=info msg="Loading containers: done."
Jun 09 20:27:50 ip-172-31-91-124.ec2.internal dockerd[4384]: time="2025-06-09T20:27:50.192039728Z" level=info msg="Docker daemon" com...25.0.8
Jun 09 20:27:50 ip-172-31-91-124.ec2.internal dockerd[4384]: time="2025-06-09T20:27:50.192463149Z" level=info msg="Daemon has complet...ation"
Jun 09 20:27:50 ip-172-31-91-124.ec2.internal dockerd[4384]: time="2025-06-09T20:27:50.251588080Z" level=info msg="API listen on /run....sock"
Jun 09 20:27:50 ip-172-31-91-124.ec2.internal systemd[1]: Started Docker Application Container Engine.
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-91-124 ~]#
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

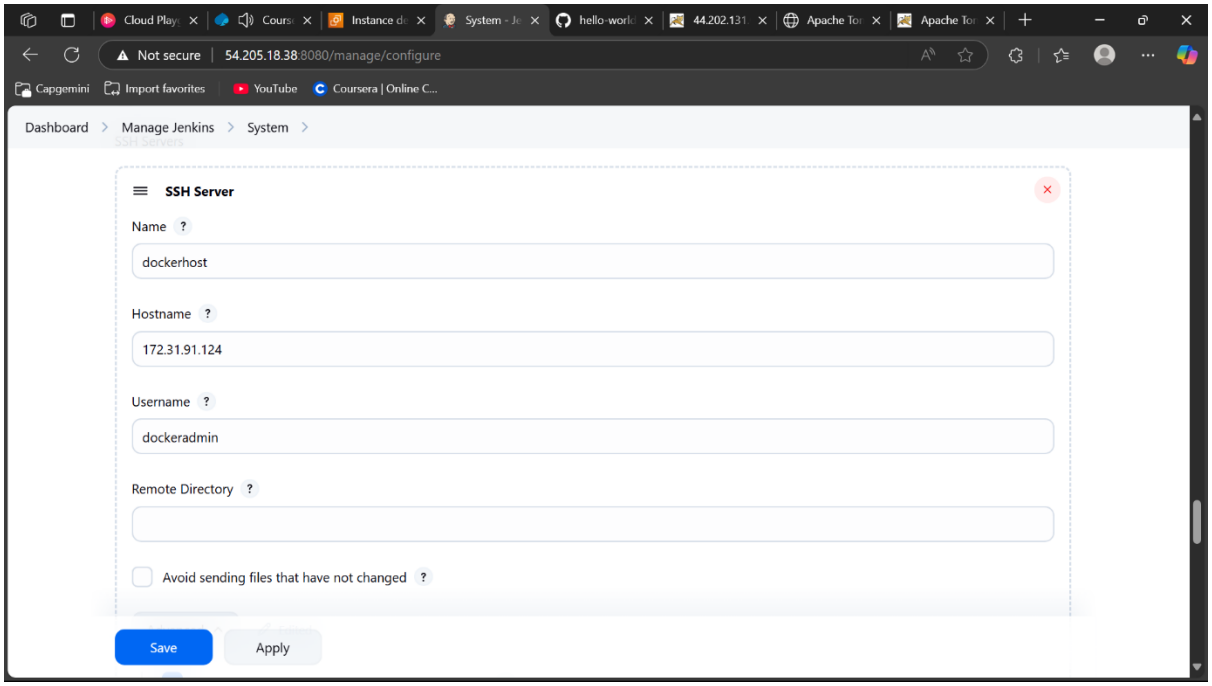


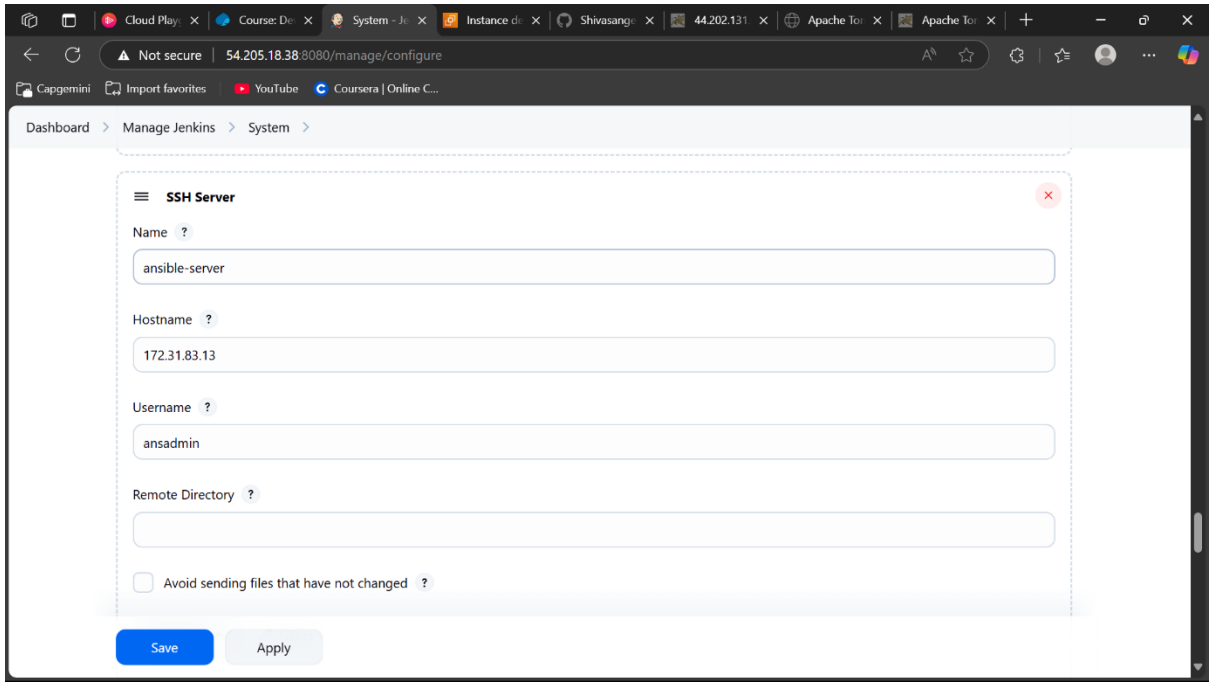


2. **Connecting Ansible to Docker Host:** On the Docker Host, I created a user named ansadmin and granted it sudo privileges. Then, a passwordless SSH connection was set up from the Ansible Controller to the Docker Host. A successful test was performed using the ansible all -m ping command.

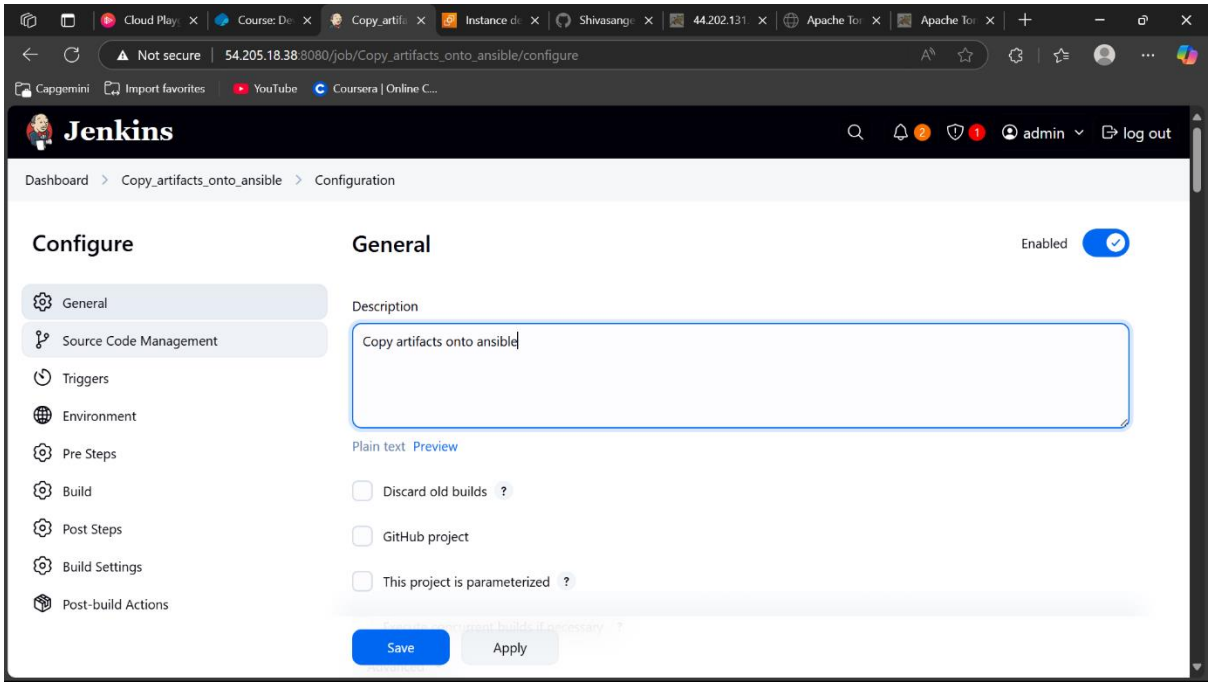


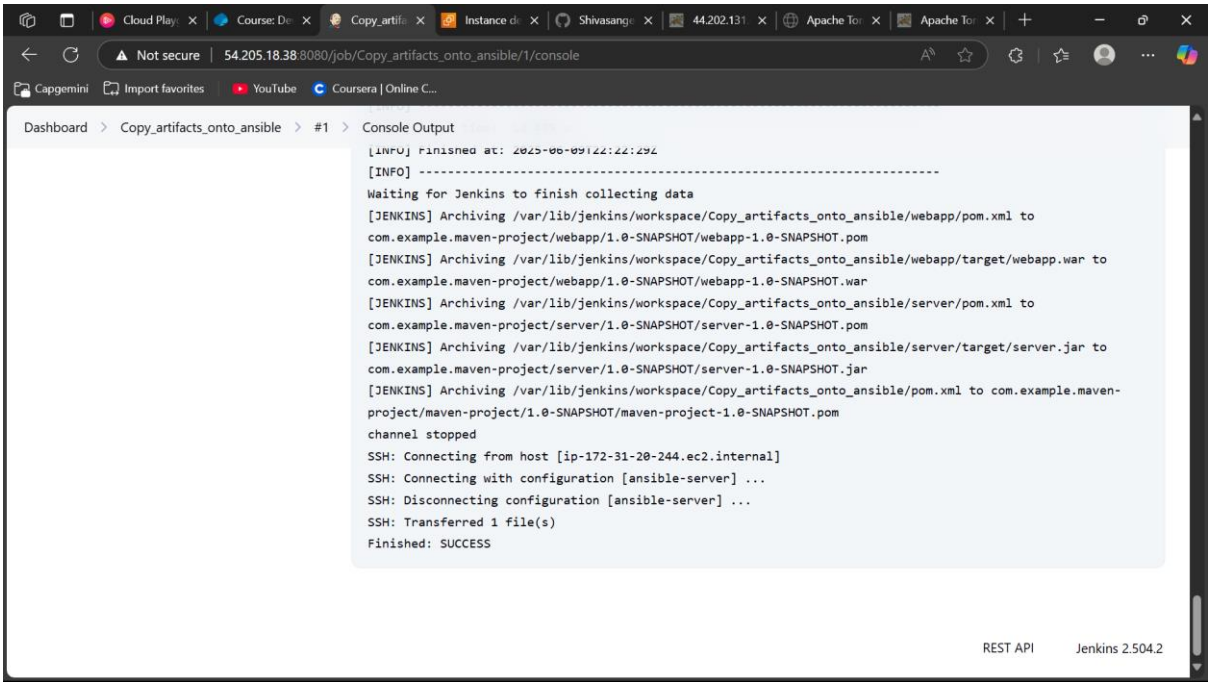
3. **Linking Jenkins to Docker and Ansible:** The "Publish Over SSH" plugin was installed in Jenkins. It was configured with the Docker's and Ansible server's IP address and username to allow Jenkins to send files to it.

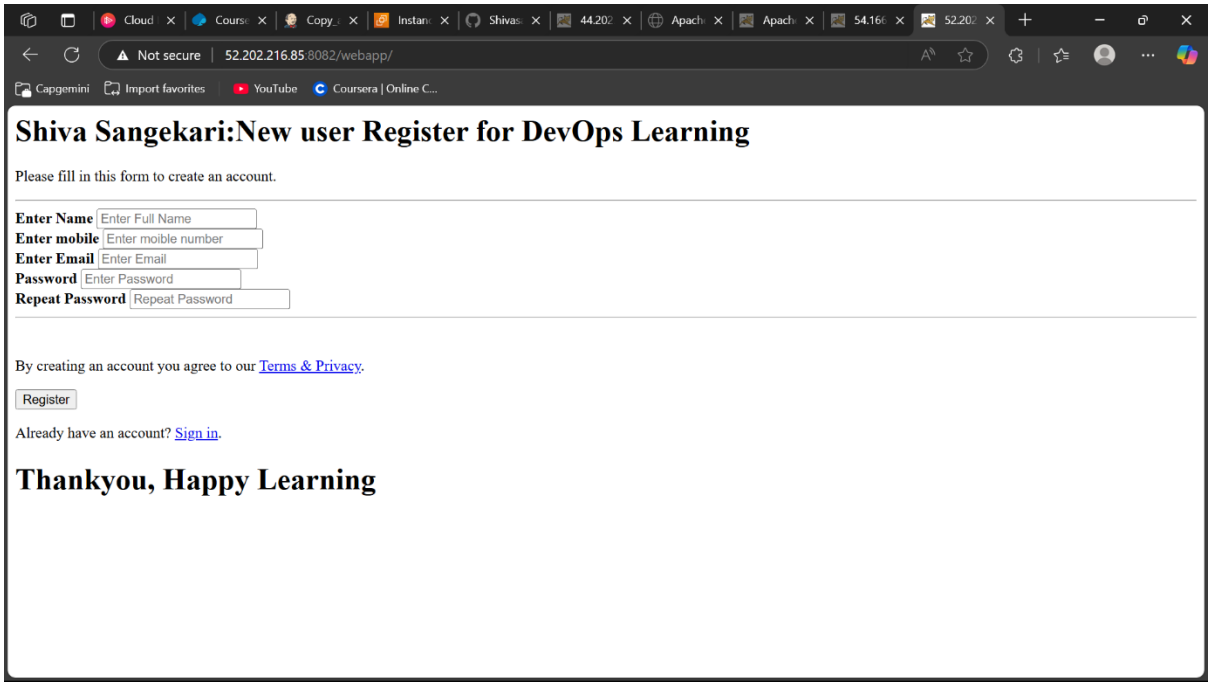




4. **Creating the Main Deployment Job:** A new Jenkins job was created to handle the full CI/CD workflow.
    - As a post-build action, the job was configured to transfer the webapp.war file to the Ansible server.
    - Following the file transfer, a shell command was added to execute the Ansible playbooks.

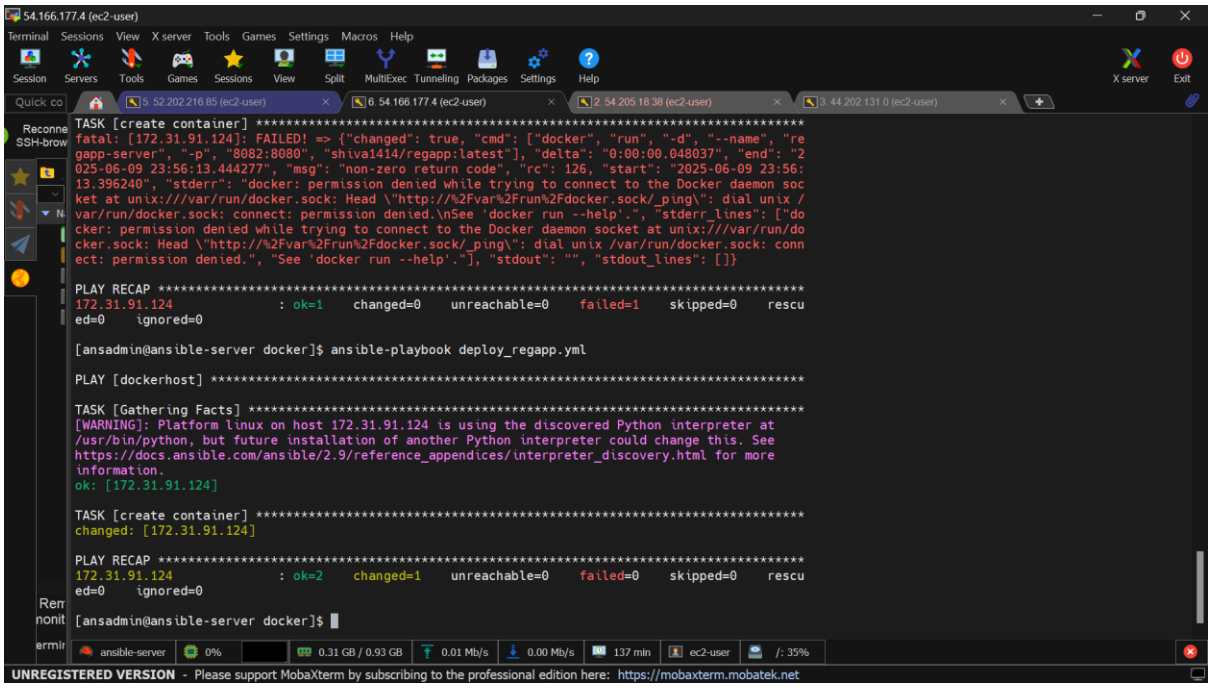

Not secure
54.205.18.38:8080/job/Copy_artifacts_onto_ansible/configure
Jenkins
admin
log out
Dashboard
Copy_artifacts_onto_ansible
Configuration
Configure
General
Source Code Management
Triggers
Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
General
Enabled
Description
Copy artifacts onto ansible
Plain text
Preview
Discard old builds
GitHub project
This project is parameterized
Save
Apply

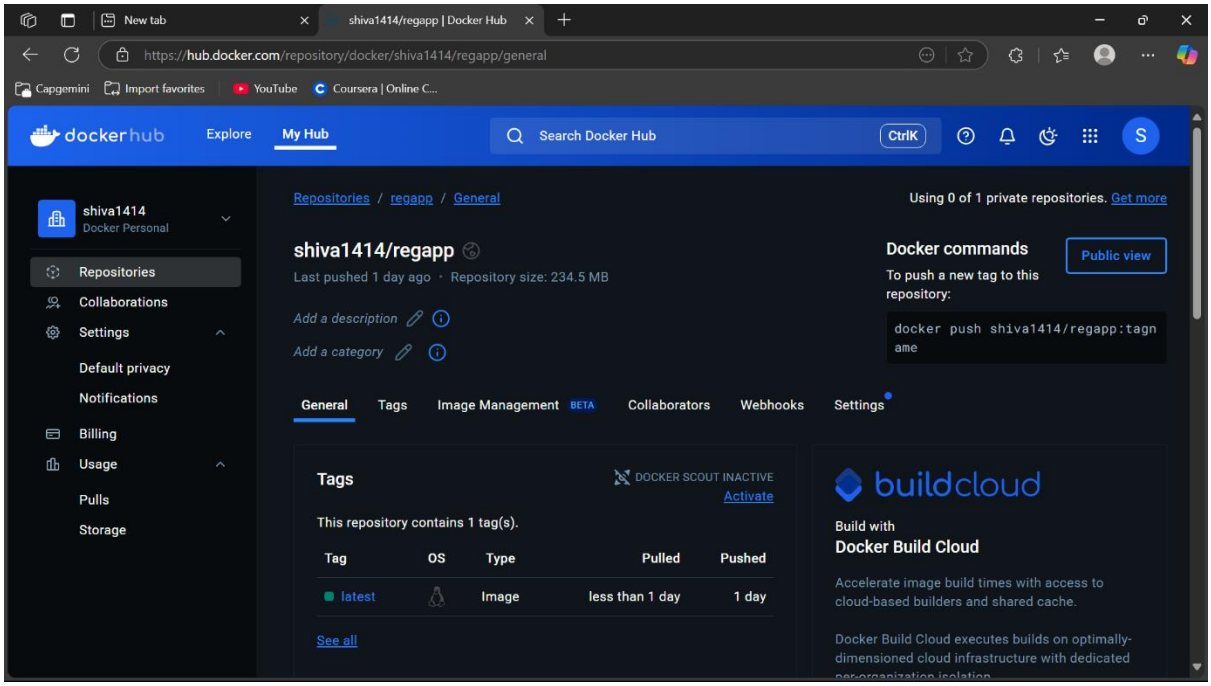

Not secure
54.205.18.38:8080/job/Copy_artifacts_onto_ansible/1/console
Dashboard
Copy_artifacts_onto_ansible
#1
Console Output
[INFO] --
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /var/lib/jenkins/workspace/Copy_artifacts_onto_ansible/webapp/pom.xml to com.example.maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/Copy_artifacts_onto_ansible/webapp/target/webapp.war to com.example.maven-project/webapp/1.0-SNAPSHOT/webapp-1.0-SNAPSHOT.war
[JENKINS] Archiving /var/lib/jenkins/workspace/Copy_artifacts_onto_ansible/server/pom.xml to com.example.maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/Copy_artifacts_onto_ansible/server/target/server.jar to com.example.maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.jar
[JENKINS] Archiving /var/lib/jenkins/workspace/Copy_artifacts_onto_ansible/pom.xml to com.example.maven-project/maven-project/1.0-SNAPSHOT/maven-project-1.0-SNAPSHOT.pom
channel stopped
SSH: Connecting from host [ip-172-31-20-244.ec2.internal]
SSH: Connecting with configuration [ansible-server] ...
SSH: Disconnecting configuration [ansible-server] ...
SSH: Transferred 1 file(s)
Finished: SUCCESS
REST API
Jenkins 2.504.2



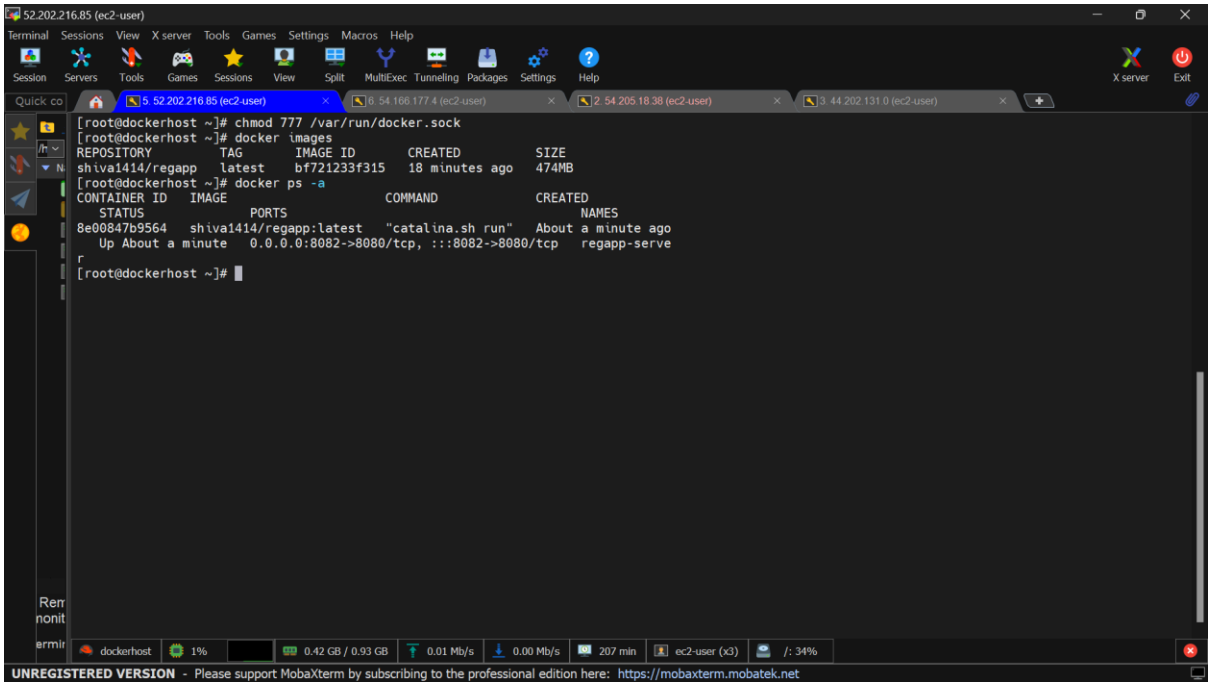
5. **Running the Ansible Playbooks:**

   - The first playbook built a Docker image of the application and pushed it to my account on Docker Hub.
   - The second playbook pulled the latest image from Docker Hub and ran it as a new container on the Docker Host, mapping port 8082 on the host to port 8080 in the container.





Final Deployment Verification: The Jenkins job completed successfully. I checked the Docker Host and confirmed the new container was running. The application was then accessible in a web browser using the server's public IP on port 8082.

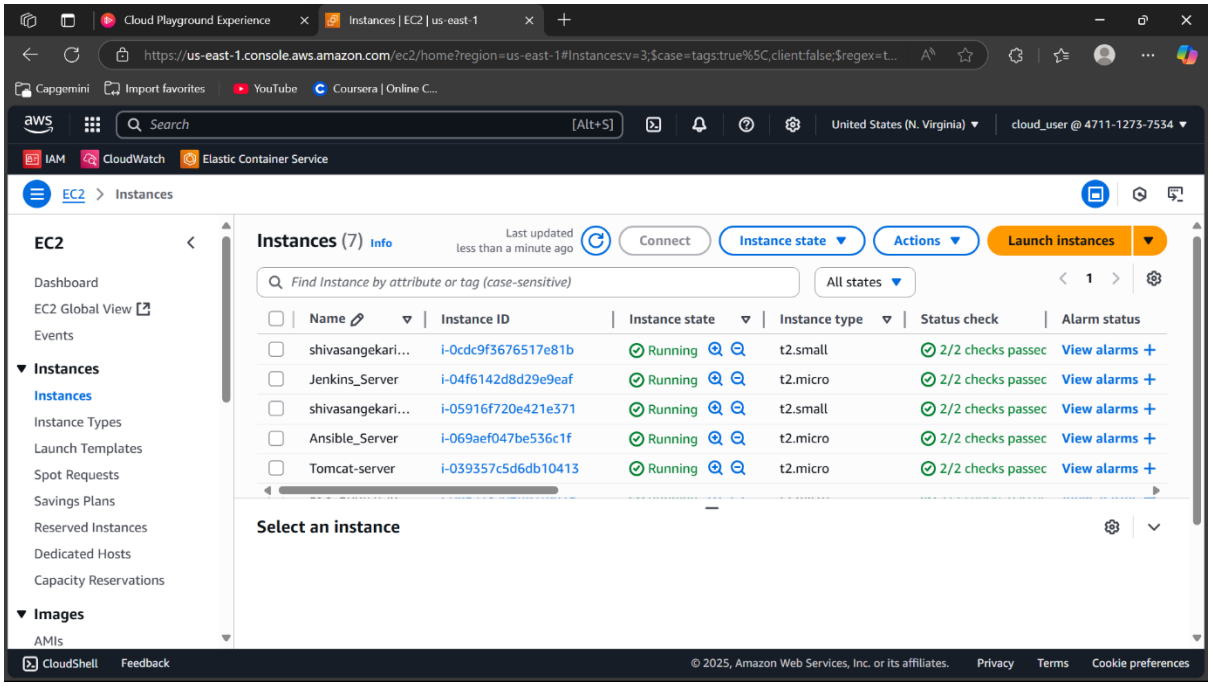


Step 4: Preparing the Environment for Kubernetes

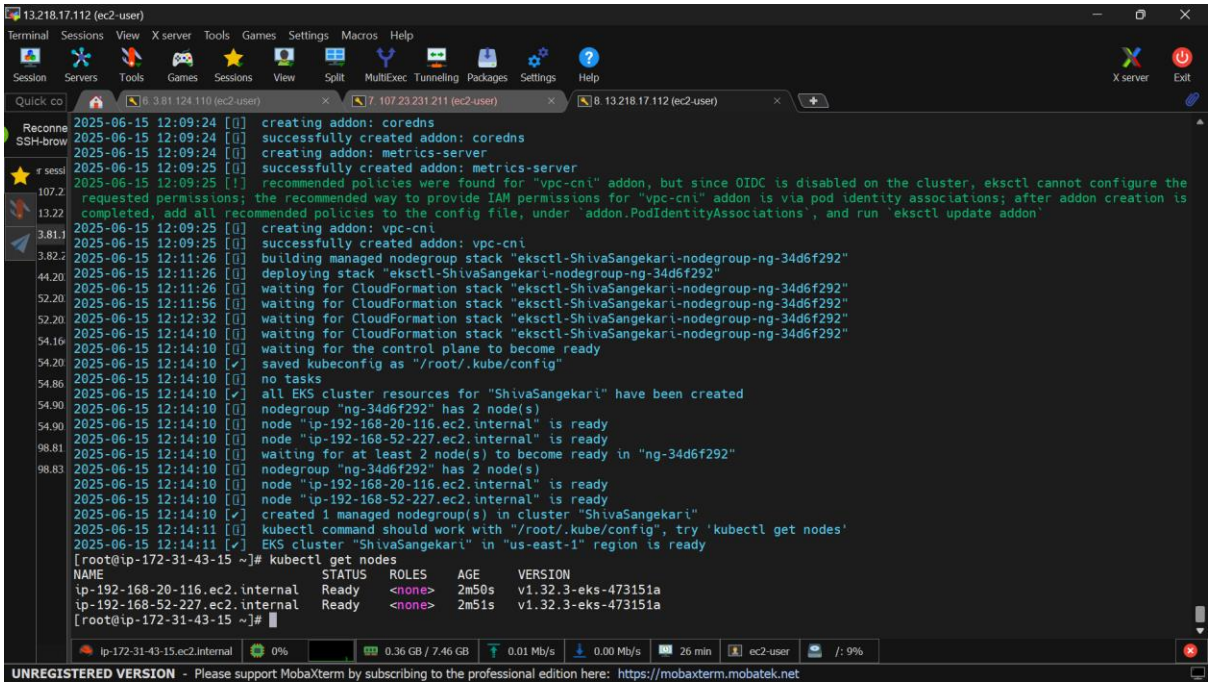
The final step was to set up the tools needed to manage the application using Kubernetes.

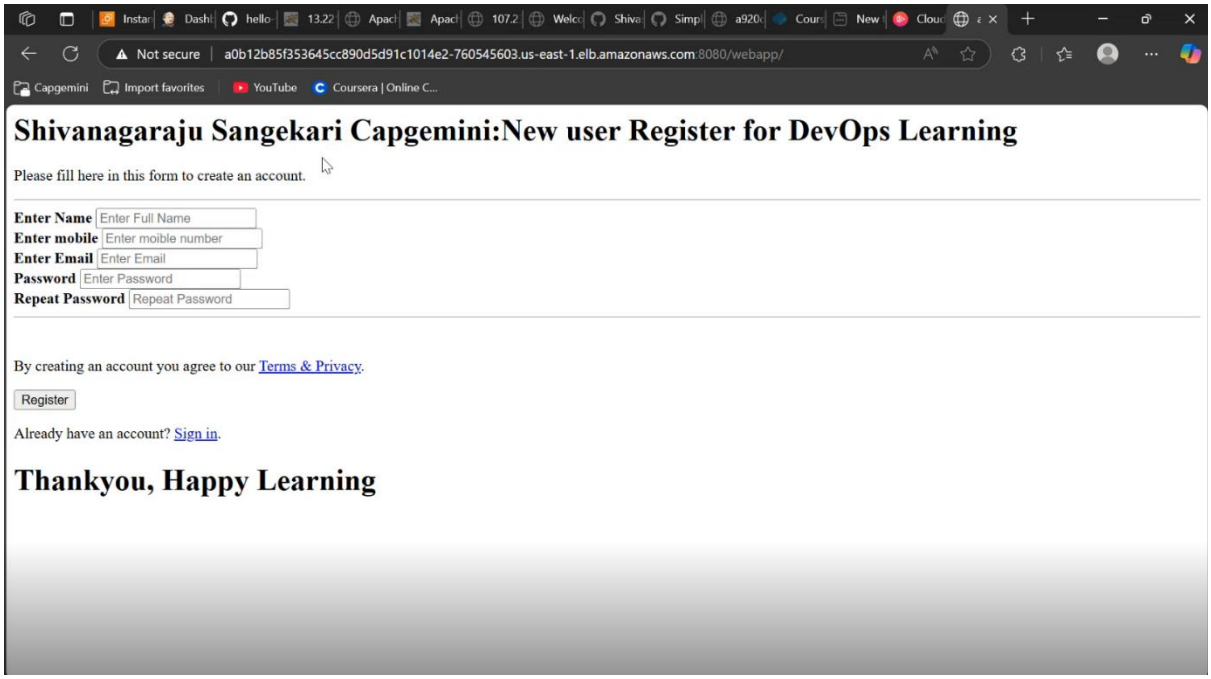
1. **EKS Bootstrap Server:** A new EC2 instance was launched to act as a bootstrap server for managing the Amazon EKS cluster.

2. **Kubernetes Tool Installation:** The necessary command-line tools (aws-cli, kubectl, and eksctl) were installed on the bootstrap server.
3. **EKS Cluster Creation:** An IAM role with administrator access was created and attached to the bootstrap server. This granted eksctl the permissions needed to create AWS resources. eksctl was then used to create the EKS cluster successfully.



In below you can see cluster has been created and you can also see after executing kubectl get nodes it showing two nodes created and status is ready





Above pic is been accessed using External link of load balancer, these load balancer is created when I created service manifest file in eks

Conclusion

Project Outcomes

This project was a valuable, hands-on experience. I learned how to move from just knowing about DevOps tools to using them in a practical way. A complete CI/CD pipeline was built from scratch, which showed how the entire software delivery process can be automated. A key outcome was seeing how tools like Jenkins, Docker, and Ansible integrate to create an efficient workflow. Using Docker was especially important, as it helps ensure an application will run reliably anywhere. This project has given me a strong foundation for learning more advanced topics in the future.

Skills Gained

- Setting up and configuring servers on AWS EC2.
- Building automated CI/CD pipelines in Jenkins.
- Using Maven to build and package Java applications.
- Creating containerized applications with Docker.
- Writing Ansible playbooks for server automation.
- Integrating different DevOps tools into a single workflow.

- Troubleshooting and problem-solving to fix technical issues.

Challenges Faced

One of the main challenges I faced was with user permissions. For instance, the Ansible playbook failed initially because the user did not have permission to access the Docker service. I had to carefully examine the error logs to understand and fix the problem by giving the correct permissions. This experience taught me how important it is to be patient and methodical when troubleshooting. Overcoming these challenges has made me more confident in my technical abilities.