

# TABLE OF CONTENTS

<b>TABLE OF CONTENTS.....</b>	<b>1</b>
<b>PROJECT INFORMATION.....</b>	<b>2</b>
Project Description:.....	2
Developer:.....	2
Project github url:.....	2
Technical Stack:.....	2
<b>STEP-BY-STEP BREAKDOWN.....</b>	<b>3</b>
Step 1: Setup AWS EC2 Instance.....	3
Step 2: Setup Jenkins.....	4
Step 3: Setup GitHub Repository.....	7
Step 4: Run Jenkins pipeline.....	7
<b>Conclusion.....</b>	<b>11</b>

# PROJECT INFORMATION

## Description:

### Project Description:

CI/CD Deployment for Spring Boot Application

As a Full Stack Developer, you have to build a CI/CD pipeline to demonstrate continuous deployment and host the application on AWS EC2 instance

### Developer:

Le Giang Nam

### Project github url:

<https://github.com/rznngnam1402/spring-boot-helloworld>

### Technical Stack:

Programming Language: Java

Version Control System (VCS): Git

Hosting Platform: GitHub, AWS

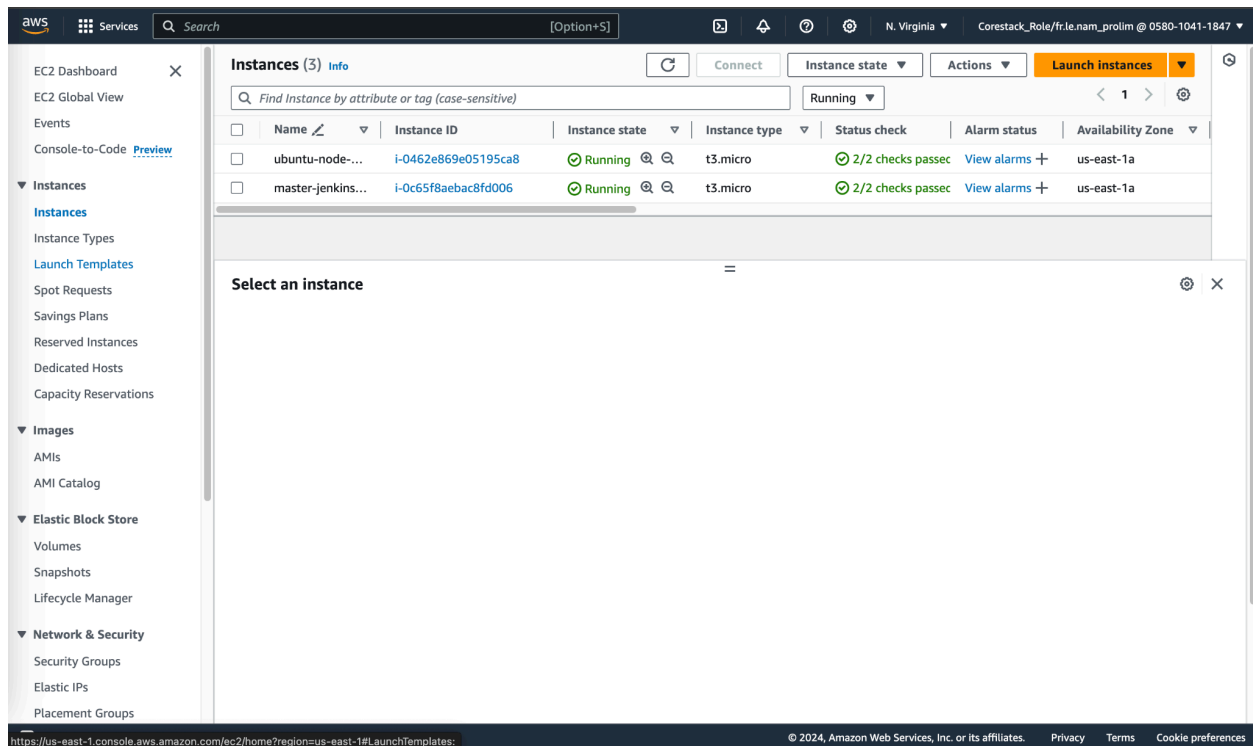
Tool: Jenkins

# STEP-BY-STEP BREAKDOWN

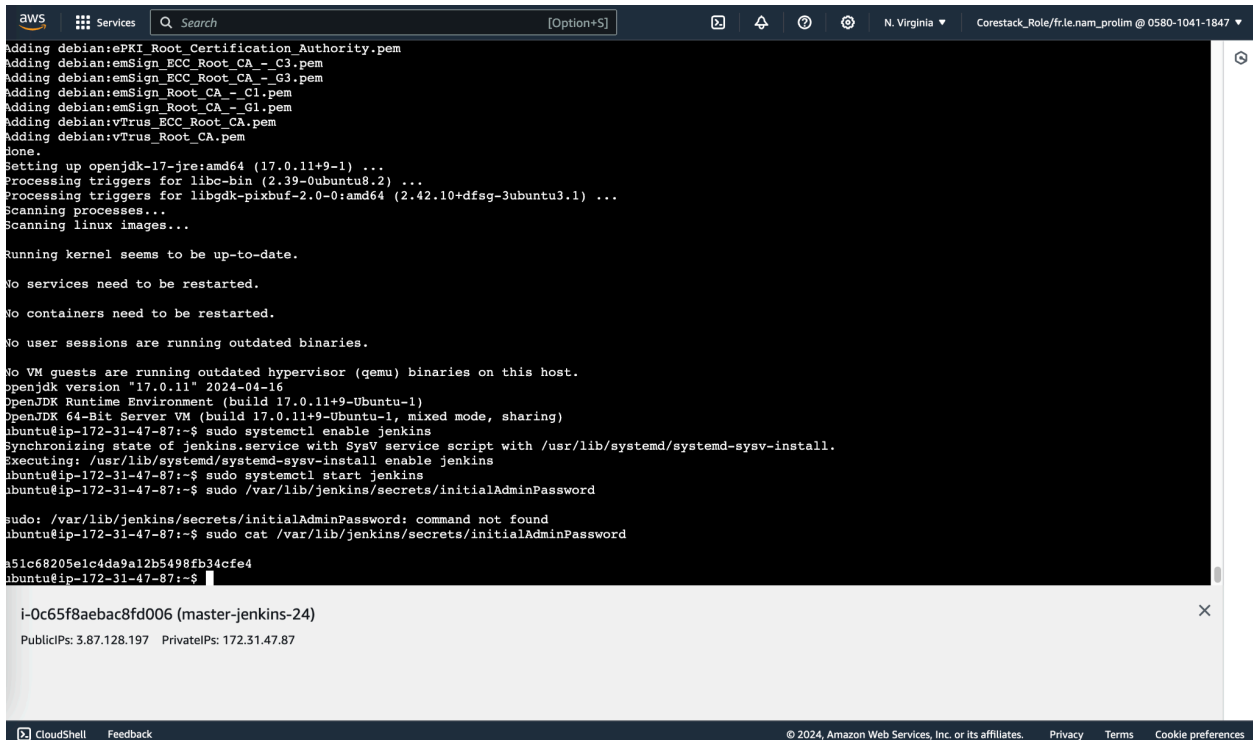
## Step 1: Setup AWS EC2 Instance

1. Go to the AWS Management Console.
2. Navigate to EC2 and launch a new instance.
3. Configure instance details, add storage, and configure security group (allow SSH, HTTP, and any other necessary ports).
4. Launch and download the key pair

For this project, I used two instances to deploy the application because a single instance does not have enough resources to handle both Jenkins and Java. One instance is dedicated to running Jenkins, while the other is used for running the Java Spring Boot application.



## 5. Connect to EC2 Instances:



```
aws
Services
Q Search
[Option+S]
N. Virginia
Corestack_Role/fr.le.nam_prolim @ 0580-1041-1847

Adding debian:ePKI_Root_Certification_Authority.pem
Adding debian:emSign_ECC_Root_CA_-_C3.pem
Adding debian:emSign_ECC_Root_CA_-_G3.pem
Adding debian:emSign_Root_CA_-_C1.pem
Adding debian:emSign_Root_CA_-_G1.pem
Adding debian:vTrus_ECC_Root_CA.pem
Adding debian:vTrus_Root_CA.pem
done.
Setting up openjdk-17-jre:amd64 (17.0.11+9-1) ...
Processing triggers for libc-bin (2.39-0ubuntu8.2) ...
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.10+dfsg-3ubuntu3.1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
openjdk version "17.0.11" 2024-04-16
OpenJDK Runtime Environment (build 17.0.11+9-Ubuntu-1)
OpenJDK 64-Bit Server VM (build 17.0.11+9-Ubuntu-1, mixed mode, sharing)
ubuntu@ip-172-31-47-87:~$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-47-87:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-47-87:~$ sudo /var/lib/jenkins/secrets/initialAdminPassword

sudo: /var/lib/jenkins/secrets/initialAdminPassword: command not found
ubuntu@ip-172-31-47-87:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

a51c68205e1c4da9a12b5498fb34cfe4
ubuntu@ip-172-31-47-87:~$

i-0c65f8aebac8fd006 (master-jenkins-24)
PublicIPs: 3.87.128.197 PrivateIPs: 172.31.47.87

CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
```

## Step 2: Setup Jenkins

1. Install Jenkins, java 17 on my EC2 Instance using these commands:

### Long Term Support release

A [LTS \(Long-Term Support\) release](#) is chosen every 12 weeks from the stream of regular releases as the stable release for that time period. It can be installed from the [debian-stable apt repository](#).

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

```
sudo apt update
sudo apt install fontconfig openjdk-17-jre
java -version
openjdk version "17.0.8" 2023-07-18
OpenJDK Runtime Environment (build 17.0.8+7-Debian-1deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.8+7-Debian-1deb12u1, mixed mode, sharing)
```

## 2. Start Jenkins on the server:

```
ubuntu@ip-172-31-47-87:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Wed 2024-07-24 03:26:31 UTC; 49min ago
     Main PID: 4892 (java)
        Tasks: 45 (limit: 1078)
      Memory: 524.7M (peak: 531.9M)
         CPU: 4min 14.180s
    CGroup: /system.slice/jenkins.service
            └─4892 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

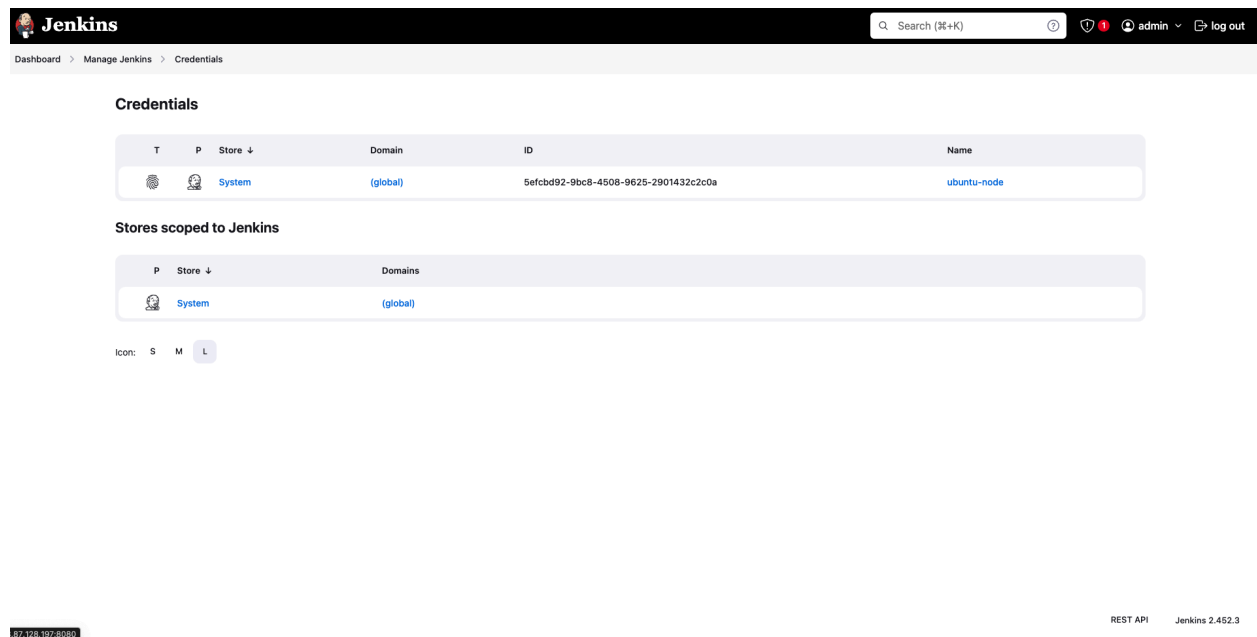
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.117+0000 [id=263] INFO jenkins.InitReactorRunner$1#onAttained: Listed a
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.117+0000 [id=262] INFO jenkins.InitReactorRunner$1#onAttained: Prepared>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.172+0000 [id=263] INFO jenkins.InitReactorRunner$1#onAttained: Started >
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.174+0000 [id=263] INFO jenkins.InitReactorRunner$1#onAttained: Augmente>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.234+0000 [id=261] INFO jenkins.InitReactorRunner$1#onAttained: System c>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.234+0000 [id=261] INFO jenkins.InitReactorRunner$1#onAttained: System c>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.261+0000 [id=261] INFO jenkins.InitReactorRunner$1#onAttained: Loaded a>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.261+0000 [id=261] INFO jenkins.InitReactorRunner$1#onAttained: Loaded a>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.261+0000 [id=262] INFO jenkins.InitReactorRunner$1#onAttained: Configur>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.261+0000 [id=262] INFO jenkins.InitReactorRunner$1#onAttained: Configur>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.755+0000 [id=264] INFO jenkins.InitReactorRunner$1#onAttained: Comple>
Jul 24 03:28:29 ip-172-31-47-87 jenkins[4892]: 2024-07-24 03:28:29.756+0000 [id=73] INFO h.m.UpdateCenter$CompleteBatchJob#run: Completed
lines 1-20/20 (END)
```

i-0c65f8aebac8fd006 (master-jenkins-24)

PublicIPs: 3.87.128.197 PrivateIPs: 172.31.47.87

## 3. Configure Jenkins

- Access Jenkins on 3.82.15.132/8080 and install suggested plugins
- Create credentials for the second instance



## 4. Set up Jenkins pipeline

Use this script to set up java spring boot for deployment on the second instance:

```

pipeline {
    agent any

    environment {
        SECOND_INSTANCE_IP = '3.82.15.132'
        REPO_URL = 'https://github.com/rzngnam1402/spring-boot-helloworld'
        SECOND_INSTANCE_SSH_KEY = '5efcbd92-9bc8-4508-9625-2901432c2c0a'
        APP_NAME = 'spring-boot-helloworld'
        JAR_NAME = 'hello-world-0.0.1-SNAPSHOT.jar'
    }

    stages {
        stage('Clone Repository') {
            steps {
                withCredentials([sshUserPrivateKey(credentialsId: SECOND_INSTANCE_SSH_KEY, keyFileVariable: 'SSH_KEY_FILE')])
            }
            script {
                sh """
                ssh -i \${SSH_KEY_FILE} ubuntu@\${SECOND_INSTANCE_IP} " \
                if [ ! -d "/home/ubuntu/\${APP_NAME}" ]; then \
                git clone \${REPO_URL} /home/ubuntu/\${APP_NAME}; \
                else \
                cd /home/ubuntu/\${APP_NAME} && git fetch && git checkout main && git pull origin main; \
                fi"
                """
            }
        }

        stage('Build Application') {
            steps {
                withCredentials([sshUserPrivateKey(credentialsId: SECOND_INSTANCE_SSH_KEY, keyFileVariable: 'SSH_KEY_FILE')])
            }
            script {
                sh """
                ssh -i \${SSH_KEY_FILE} ubuntu@\${SECOND_INSTANCE_IP} " \
                cd /home/ubuntu/\${APP_NAME} \
                && mvn clean package -X"
                """
            }
        }

        stage('Deploy Application') {
            steps {
                withCredentials([sshUserPrivateKey(credentialsId: SECOND_INSTANCE_SSH_KEY, keyFileVariable: 'SSH_KEY_FILE')])
            }
            script {
                sh """
                ssh -i \${SSH_KEY_FILE} ubuntu@\${SECOND_INSTANCE_IP} "

                nohup java -jar /home/ubuntu/\${APP_NAME}/target/\${JAR_NAME} > /home/ubuntu/app.log 2>&1 &

                "
                """
            }
        }
    }

    post {
        success {
            echo 'Build and deployment succeeded!'
        }
        failure {
            echo 'Build and deployment failed!'
        }
    }
}

```


## Step 3: Setup GitHub Repository




1. Create new repo
2. Push source code to github: <https://github.com/rznngnam1402/spring-boot-helloworld>

## Step 4: Run Jenkins pipeline


1. Select Build Now
2. See changes in Build Output

The screenshot displays the Jenkins web interface in a browser. The address bar shows the URL `3.87.128.197:8080/job/App-pipeline/8/`. The Jenkins logo and a search bar are at the top. The breadcrumb navigation indicates the path: `Dashboard > App-pipeline > #8`. On the left, a sidebar lists various options: Status, Changes, Console Output, Edit Build Information (highlighted), Delete build '#8', Timings, Pipeline Overview, Pipeline Console, Restart from Stage, Replay, Pipeline Steps, Workspaces, and Previous Build. The main content area shows the details for **Build #8 (Jul 24, 2024, 3:45:12 AM)**, which is marked as successful with a green checkmark. It includes a 'Keep this build forever' button, an 'Add description' link, and a timestamp 'Started 39 min ago' with 'Took 11 sec'. A section titled 'This run spent:' lists the following metrics: 20 ms waiting, 11 sec build duration, and 11 sec total from scheduled to completion. The bottom of the page shows the REST API endpoint `3.87.128.197:8080/job/App-pipeline/8/configure` and the version `Jenkins 2.452.3`.





 < **Build #8**


  





Pipeline







Details

 Manually run by admin  
 Started 42 min ago  
 Queued 2 ms  
 Took 11 sec






 < **Build #8**


   




Success 43 min ago in 11 sec

 Clone Repository  
 Build Application  
 Deploy Application  
 **Post Actions**

Stage 'Post Actions'

 Started 43 min ago  
 Queued 0 ms  
 Took 0.1 sec  
 Success  
 [View as plain text](#)

 **Build and deployment succeeded!**  
Print Message

42 ms   

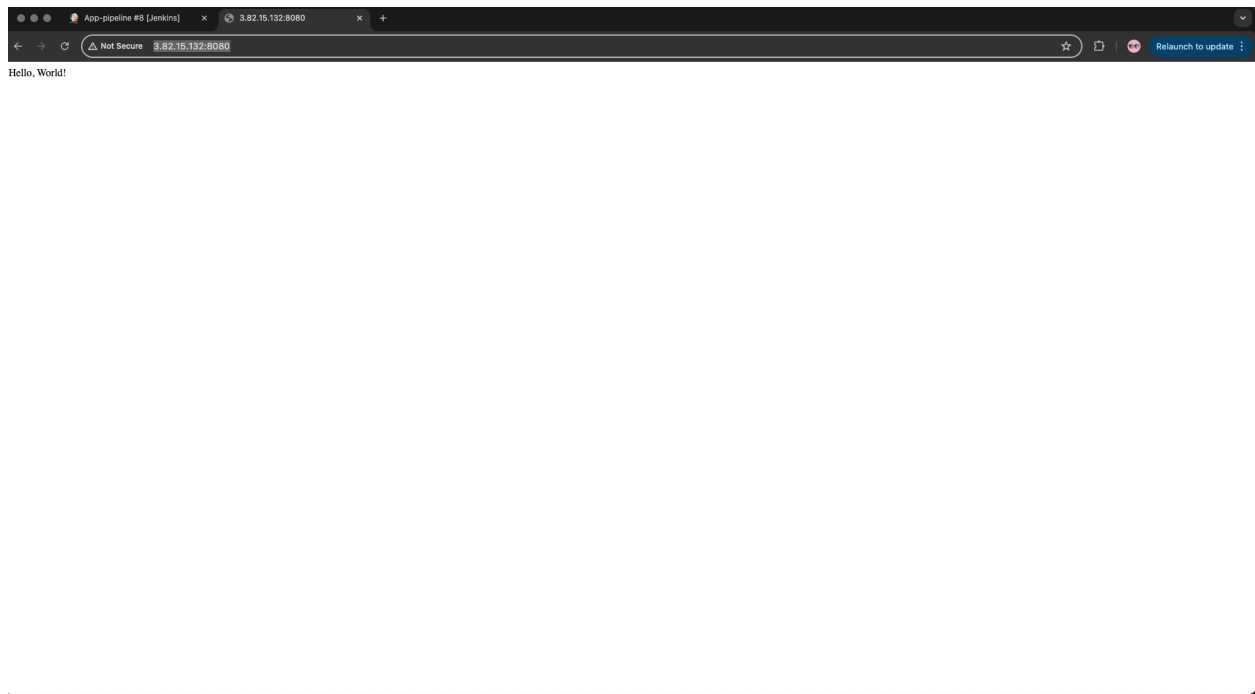
Build and deployment succeeded!



- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#8'
- Timings
- Pipeline Overview
- Pipeline Console
- Restart from Stage
- Replay
- Pipeline Steps**
- Workspaces
- Previous Build

Step	Arguments	Status
Start of Pipeline - (11 sec in block)		✓
node - (10 sec in block)		✓
node block - (10 sec in block)		✓
withEnv - (10 sec in block)	REPO_URL, JAR_NAME, APP_NAME, SECOND_INSTANCE_IP, SECOND_INSTANCE_SSH_KEY	✓
withEnv block - (10 sec in block)		✓
stage - (1.9 sec in block)	Clone Repository	✓
stage block (Clone Repository) - (1.8 sec in block)		✓
withCredentials - (1.7 sec in block)		✓
withCredentials block - (1.6 sec in block)		✓
script - (1.5 sec in block)		✓
script block - (1.4 sec in block)		✓
sh - (1.4 sec in self)	ssh -i \$(SSH_KEY_FILE) ubuntu@\$(SECOND_INSTANCE_IP) " if [ ! -d "/home/ubuntu/spring-boot-helloworld" ]; then git clone https://github.com/zrنگnam1402/spring-boot-helloworld /home/ubuntu/spring-boot-helloworld; else cd /home/ubuntu/spring-boot-helloworld && git fetch && git checkout main && git pull origin main; fi"	✓
stage - (7.7 sec in block)	Build Application	✓
stage block (Build Application) - (7.6 sec in block)		✓
withCredentials - (7.5 sec in block)		✓
withCredentials block - (7.5 sec in block)		✓
script - (7.4 sec in block)		✓
script block - (7.4 sec in block)		✓
sh - (7.3 sec in self)	ssh -i \$(SSH_KEY_FILE) ubuntu@\$(SECOND_INSTANCE_IP) " cd /home/ubuntu/spring-boot-helloworld && mvn clean package -X"	✓
stage - (0.88 sec in block)	Deploy Application	✓
stage block (Deploy Application) - (0.81 sec in block)		✓
withCredentials - (0.74 sec in block)		✓
withCredentials block - (0.67 sec in block)		✓
script - (0.62 sec in block)		✓
script block - (0.56 sec in block)		✓
sh - (0.52 sec in self)	ssh -i \$(SSH_KEY_FILE) ubuntu@\$(SECOND_INSTANCE_IP) " nohup java -jar /home/ubuntu/spring-boot-helloworld/target/hello-world-0.0.1-SNAPSHOT.jar > /home/ubuntu/app.log 2>&1 & "	✓
stage - (0.15 sec in block)	Declarative: Post Actions	✓
stage block (Declarative: Post Actions) - (0.1 sec in block)		✓
echo - (42 ms in self)	Build and deployment succeeded!	✓

3. When built successfully, access <http://3.82.15.132:8080/> to see deployed application



# Conclusion

In this project, I gained valuable experience in setting up and managing a deployment pipeline for a Spring Boot application using Jenkins and AWS EC2 instances. Key takeaways from this project include:

1. **Resource management:** I learned the importance of resource allocation and management. By utilizing separate EC2 instances for Jenkins and the Spring Boot application, I effectively mitigated resource constraints and ensured smooth operation of both systems.
2. **Continuous integration and deployment (CI/CD):** Implementing a CI/CD pipeline helped me understand the nuances of automating build and deployment processes.
3. **Debugging and error Handling:** Through troubleshooting issues like deployment errors and script failures, I developed problem-solving skills and learned how to debug and resolve issues effectively. This included managing log files and understanding error messages to pinpoint and fix problems.
4. **Hands-on experience:** The project provided hands-on experience with AWS EC2 instances, SSH commands, Maven build processes, and Jenkins job configurations.