

## DevOps Engineer - Case Study

Submitted By -

Empid:760210

Name: Ashish Singh Baghel

eMail: [Ashish.SBaghel@tcs.com](mailto:Ashish.SBaghel@tcs.com)

TCS wants you to analyze, recommend and perform a DevOps Practices implementation. Automate and streamline sample application with Devops Engineering Practices, including continuous integration, configuration management, continuous delivery, and automated quality assurance. The key objectives of your analysis and recommendations are:

- Create an Spring Boot Application and configure the inbuilt server port to 8885.  
Status: Complete
- Write Junit5 test cases for Spring Boot Application.  
Status: Complete
- Create a GitHub Repository and push the code to GitHub repository.  
Status: Complete
- Integrate Git plugin in jenkins pipeline to checkout the code.  
Status: Complete
- Implement Continuous Integration and Automation of Build, Test and Package.  
Status: Complete
- Implement a tool to analyse the quality of code (Sonar plugin)  
Status: Complete
- Implement Automation of Docker image creation.  
Status: Complete
- Implement authentication on docker registry.  
Status: Complete
- Implement Automation of pushing the docker image to docker registry.  
Status: Complete
- Implement email alerts for bad builds.  
Status: Complete
- Implement auto configuration of tomcat or aws ec2 instance.(using Ansible)  
Status: Complete
- Implement Automation of deployment on tomcat or aws ec2 instance.  
Status: Complete
- Implement Auto trigger of the jenkins cicd pipeline.  
Status: Complete

Note : The evaluation will be done on the provided Jenkin's environment. Once you are ready with the application on your local, please configure and test it according to the Jenkins Environment.

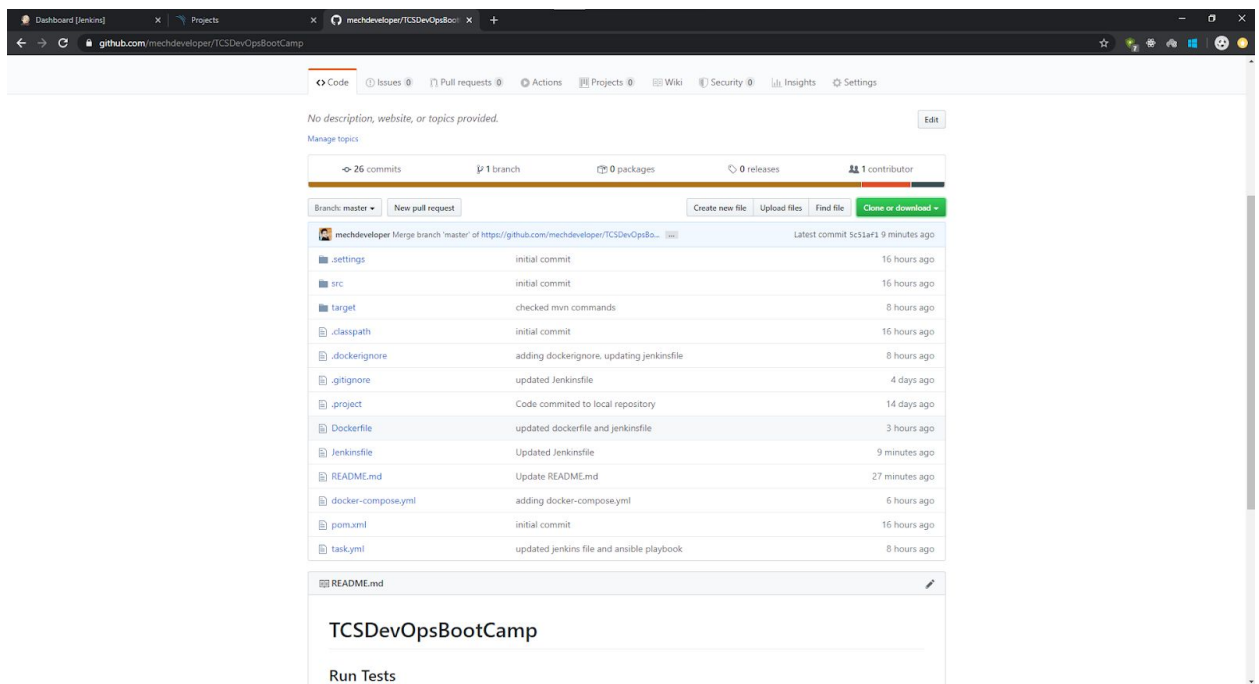
Date of Submission : 11th May 2020

## Setup Details and screenshots :

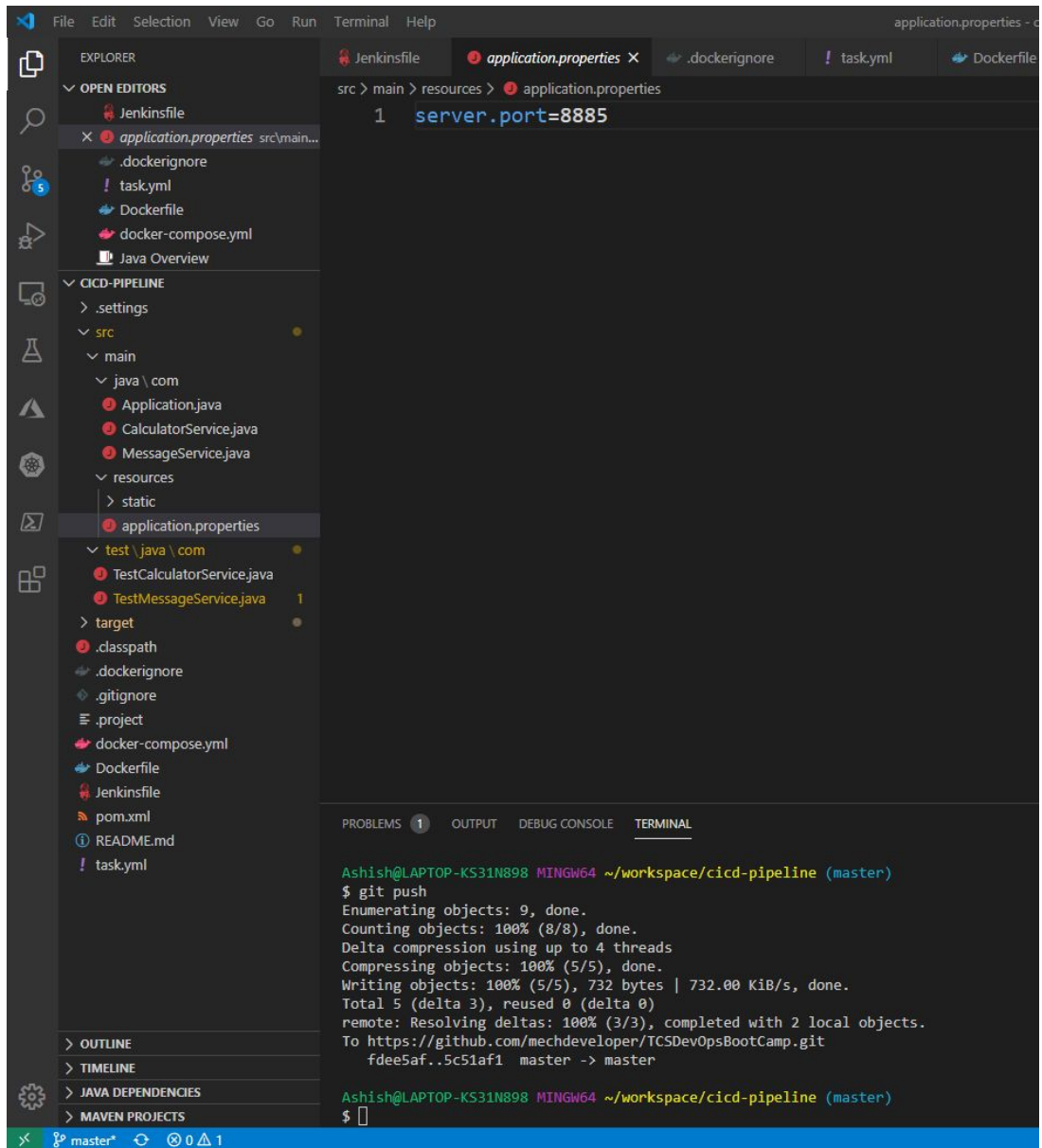
- Java Spring Boot Application -

Github repo for Spring Boot Application -

<https://github.com/mechdeveloper/TCSDevOpsBootCamp.git>



Inbuilt server port 8885 defined in application.properties file -



JUnit5 Testcases -

```

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.TestMessageService
testWelcome
testGreet
testMessage
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.02 s - in com.TestMessageService
[INFO] Running com.TestCalculatorService
testDiff
testMul
testMod
testInc
testDec
testIsGreater
testIsLower
testIsEqual
testSum
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in com.TestCalculatorService
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]

```

The screenshot shows the Visual Studio Code interface with a project named 'cicd-pipeline'. The Explorer panel on the left shows the project structure, including a 'test' directory with 'TestCalculatorService.java' and 'TestMessageService.java'. The main editor displays the code for 'TestCalculatorService.java', which includes imports for JUnit and Spring Boot Test, and a test class with a 'testSum' method. The terminal at the bottom shows the command 'src > test > java > com > TestCalculatorService.java > {} com' and the output of the test run, which matches the text in the first block.

```

TestCalculatorService.java - cicd-pipeline - Visual Studio Code
Jenkinsfile TestCalculatorService.java X .dockerignore task.yml Dockerfile docker-compose.yml
src > test > java > com > TestCalculatorService.java > {} com
1 package com;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.Test;
5 import org.junit.jupiter.api.extension.ExtendWith;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.test.context.junit.jupiter.SpringExtension;
8
9 @ExtendWith(SpringExtension.class)
10 @SpringBootTest
11 public class TestCalculatorService {
12     int num1 = 20;
13     int num2 = 5;
14
15     @Test
16     public void testSum() {
17         CalculatorService calc = new CalculatorService();
18         System.out.println("testSum");
19         Assertions.assertEquals(25, calc.calculateSum(num1, num2));
20     }
21
22

```

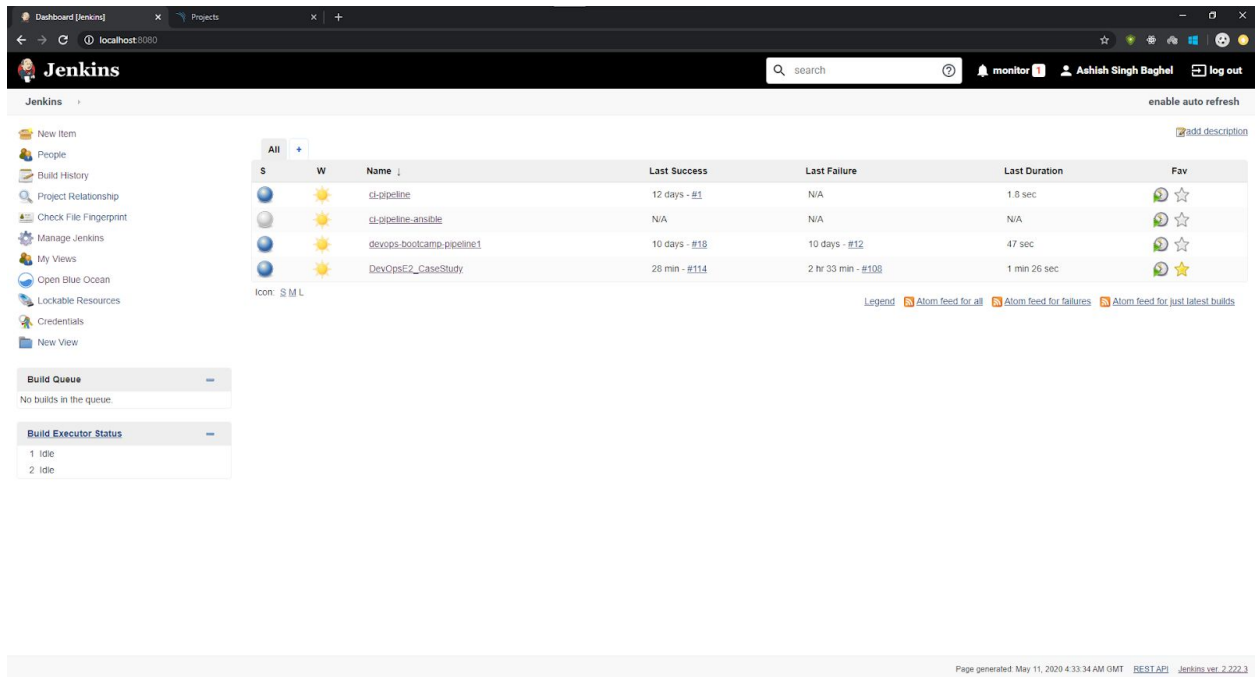
Ashish@LAPTOP-KS31N898 MINGW64 ~/workspace/cicd-pipeline (master)

## - Jenkins Setup -

Running jenkins on docker container.

Github repo with docker-compose.yml file to run jenkins as docker container -

<https://github.com/mechdeveloper/jenkins-docker.git>

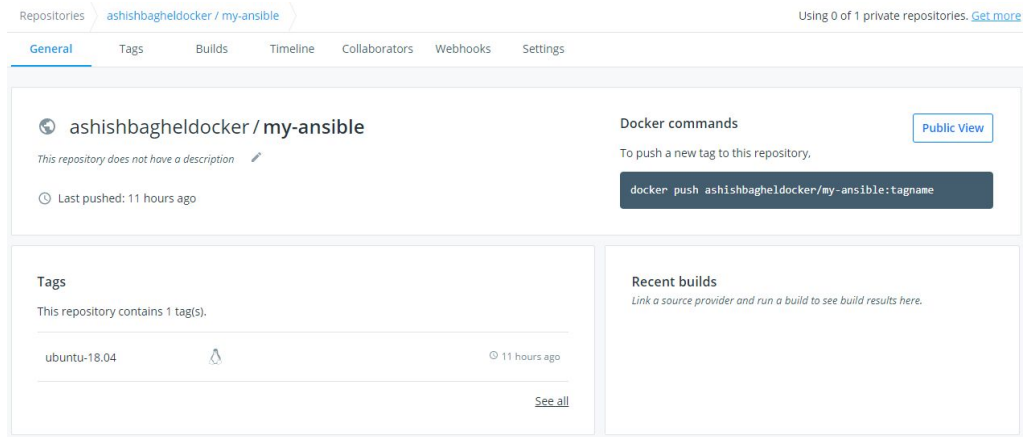


The screenshot shows the Jenkins Dashboard interface. The top navigation bar includes the Jenkins logo, a search bar, and user information (Ashish Singh Baghel). The main content area displays a table of builds with columns for status, icon, name, last success, last failure, last duration, and favorite status. Below the table, there are sections for 'Build Queue' (showing no builds) and 'Build Executor Status' (showing 1 idle and 2 busy executors). The footer indicates the page was generated on May 11, 2020, at 4:33:34 AM GMT, with the Jenkins version 2.222.3.

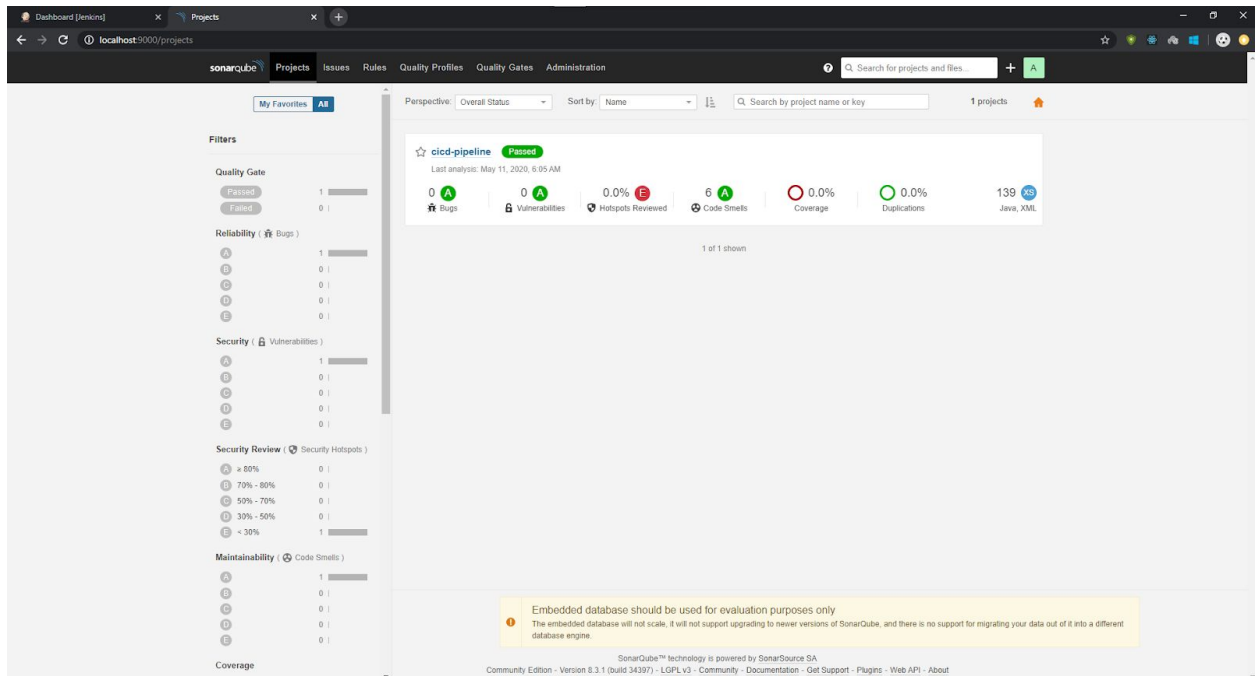
S	W	Name	Last Success	Last Failure	Last Duration	Fav
●	☀	ci-pipeline	12 days - #1	N/A	1.8 sec	☆
●	☀	ci-pipeline-ansible	N/A	N/A	N/A	☆
●	☀	devops-bootcamp-pipeline1	10 days - #18	10 days - #12	47 sec	☆
●	☀	DevOpsE2_CaseStudy	28 min - #114	2 hr 33 min - #108	1 min 26 sec	☆

## SpringBoot Application Jenkinsfile explained -

- Git checkout (source code checkout from github)
  - mvn clean (cleanup)
  - mvn package (build, test and package)
  - SonarQube analysis
  - Build Docker image (withCredentials, using dockerhub credentials to mask username and password for dockerhub)
  - Push image to dockerhub
  - Ansible create EC2 Instance (runs tasks.yml file to create EC2 instance along with its security group to allow ssh and http port 80 access over internet)
- Note: Pulling my own ansible docker image from dockerhub and running the ansible commands to spin up resources in AWS cloud -



- EC2 instance Install docker (ssh into EC2 instance using ip and privatekey)
  - EC2 instance Start docker (ssh into EC2 instance using ip and privatekey)
  - EC2 instance initiate docker swarm mode for docker stack deployment (ssh into EC2 instance using ip and privatekey)
  - EC2 instance copy docker-compose.yml for deployment of application on EC2 Instance
  - EC2 instance Deploy application as a stack of service in EC2 Docker swarm (ssh into EC2 instance using ip and privatekey)
  - Entire pipeline is enclosed in try catch block to catch any build step failure and report/send notification via email.
- 
- Sonarqube server Setup -
    - Running Sonarqube server on docker container -
    - Github repo with docker-compose.yml file to run sonarqube server as docker container - <https://github.com/mechdeveloper/sonarqube-docker.git>

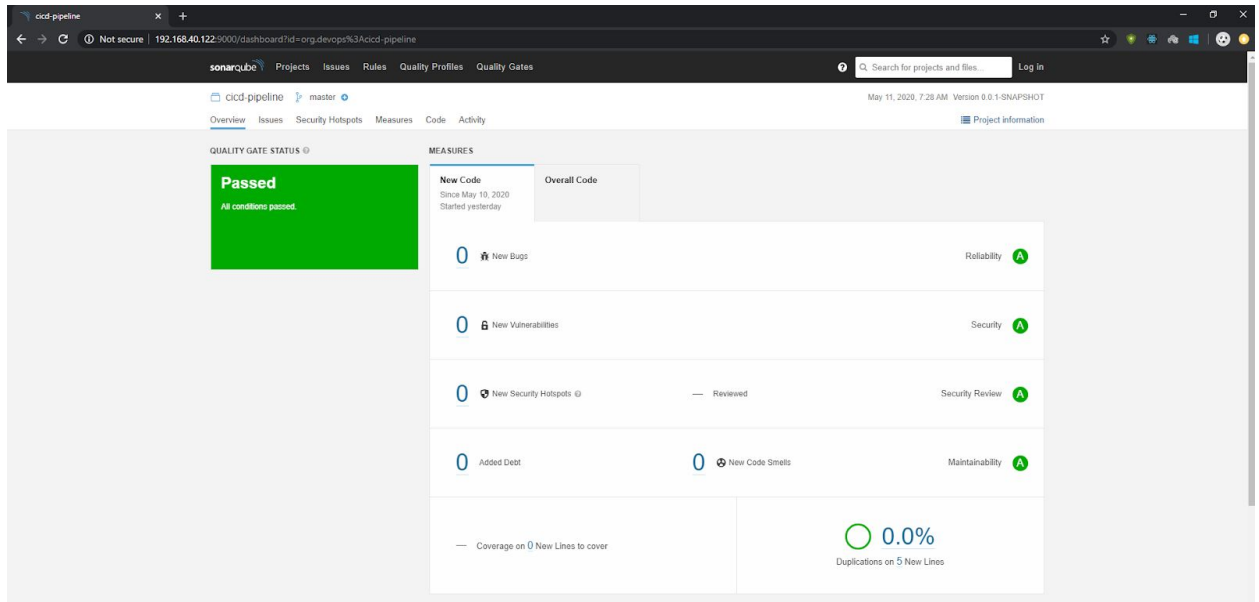


## SonarQube build success -

```
[INFO] ----- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=13ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=20ms
[INFO] CPD Executor 1 file had no CPD blocks
[INFO] CPD Executor Calculating CPD for 2 files
[INFO] CPD Executor CPD calculation finished (done) | time=10ms
[INFO] Analysis report generated in 102ms, dir size=97 KB
[INFO] Analysis report compressed in 43ms, zip size=21 KB
[INFO] Analysis report uploaded in 98ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://192.168.40.122:9000/dashboard?id=org.devops%3Acicd-pipeline
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://192.168.40.122:9000/api/ce/task?id=AXICNI1GYqcBmUEPWcqf
[INFO] Analysis total time: 8.721 s
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.860 s
[INFO] Finished at: 2020-05-11T05:28:17Z
[INFO] -----
```

## SonarQube Dashboard after code analysis -





- Jenkins configuration and Integration explained

Credentials configuration on Jenkins -

The screenshot shows the Jenkins 'Credentials' page. The table below lists the configured credentials:

T	P	Store	Domain	ID	Name
🔑	Jenkins	(global)	git-creds	mechdeveloper*****	(Credentials for github repo)
🔑	Jenkins	(global)	docker-creds	ashishbaghel*****	(Credentials for docker hub)
🔑	Jenkins	(global)	docker-password	password for docker hub	
🔑	Jenkins	(global)	sonarqube-token	sonarqube token for authentication from jenkins	
🔑	Jenkins	(global)	AWS_ACCESS_KEY_ID	aws access key id	
🔑	Jenkins	(global)	AWS_SECRET_ACCESS_KEY	aws access key	
🔑	Jenkins	(global)	devops-ec2-key	ec2-user (aws key for ec2 instances)	
🔑	Jenkins	(global)	Master Github Server Token	Master Github Token	

Stores scoped to Jenkins

P	Store	Domains
🔑	Jenkins	(global)

- git-creds : Github credentials
- docker-creds : Dockerhub credentials
- sonarqube-token : Token for sonarqube server authentication from jenkins server
- AWS\_ACCESS\_KEY\_ID : AWS access key ID to allow AWS access for EC2 configuration
- AWS\_SECRET\_ACCESS\_KEY: AWS secret access key to allow AWS access for EC2 configuration
- devops-ec2-key: private key configured in AWS to enable ssh into E2 Instances
- Master Github Token: Github integration for auto trigger of builds



## - Email Notification Jenkins:

**E-mail Notification**

SMTP server

Default user e-mail suffix

☒ Use SMTP Authentication

User Name

Password  [Change Password](#)

Use SSL ☒

Use TLS ☐

SMTP Port

Reply-To Address

Charset

☐ Test configuration by sending test e-mail

[Save](#) [Apply](#)

Enabled email Notification via gmail SMTP server -

Created app password in personal gmail account for integration with Jenkins

### ← App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. [Learn more](#)

**Your app passwords**

Name	Created	Last used	
Jenkins	May 10	3:44 AM	

Select the app and device you want to generate the app password for.

Select app ▼ Select device ▼

[GENERATE](#)

Failed Builds are notified via email -



## - SonarQube Server configuration in Jenkins -

**SonarQube servers**

Environment variables

☒ Enable injection of SonarQube server configuration as build environment variables  
If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

Server URL

Default is <http://localhost:9000>

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

List of SonarQube installations

## SonarQube Token generated for Jenkins authentication -

**Tokens**

If you want to enforce security by not providing credentials of a real SonarQube user to run your code scan or to invoke web services, you can provide a User Token as a replacement of the user login. This will increase the security of your installation by not letting your analysis user's password going through your network.

**Generate Tokens**

Name	Last use	Created	
Jenkins	< 1 hour ago	May 10, 2020	<input type="button" value="Revoke"/>

SonarQube server URL in Jenkins is IPv4 address of local machine as SonarQube is running on a Docker container similarly Jenkins is also running on its own Docker container both cannot talk to each other over network as they have their own isolated Docker network. SonarQube Token is required for Jenkins to access the SonarQube server over the IP address of local machine

## - Docker Hub Image Registry-

Created Docker Hub account to push Docker images created from Jenkins pipeline -

dockerhub

Search for great content (e.g., mysql)

Explore

Repositories

Organizations

Get Help

ashishbagheldoc...

Repositories

ashishbagheldocker / devops-e2-casestudy

Using 0 of 1 private repositories. [Get more](#)

General

Tags

Builds

Timeline

Collaborators

Webhooks

Settings

ashishbagheldocker / devops-e2-casestudy

This repository does not have a description

Last pushed: an hour ago

Docker commands






Public View

To push a new tag to this repository,

```
docker push ashishbagheldocker/devops-e2-casestudy:tagname
```

Tags

This repository contains 22 tag(s).

109		3 hours ago
107		3 hours ago
110		3 hours ago
108		3 hours ago
104		4 hours ago

Recent builds

Link a source provider and run a build to see build results here.

## - Github Integration on Jenkins for auto trigger of builds

GitHub

GitHub Servers

GitHub Server

Name

Main GitHub Server

API URL

https://api.github.com

Credentials

Master Git Hub Token

Add

☒ Manage hooks

Add GitHub Server

Test connection

Advanced...

Delete

## Created Personal Access token on Github for Jenkins integration -

[Settings](#) / [Developer settings](#)

[GitHub Apps](#)  
[OAuth Apps](#)  
**Personal access tokens**

### Edit personal access token

If you've lost or forgotten this token, you can regenerate it, but be aware that any scripts or applications using this token will need to be updated.

[Regenerate token](#)

**Note**  

Master Jenkins Dev Server

What's this token for?

**Select scopes**  
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> write:packages	Upload packages to github package registry
<input type="checkbox"/> read:packages	Download packages from github package registry
<input type="checkbox"/> delete:packages	Delete packages from github package registry

## - Jenkins Global Tool Configuration

### SonarQube Scanner installation -

#### Configured to install automatically

**SonarQube Scanner**  
SonarQube Scanner Installations

Add SonarQube Scanner

SonarQube Scanner

Name

☒ Install automatically

Install from Maven Central

Version

[Delete Installer](#)

### Maven Installation -

#### Configured to install automatically

**Maven**

Maven installations

[Add Maven](#)

Maven

Name

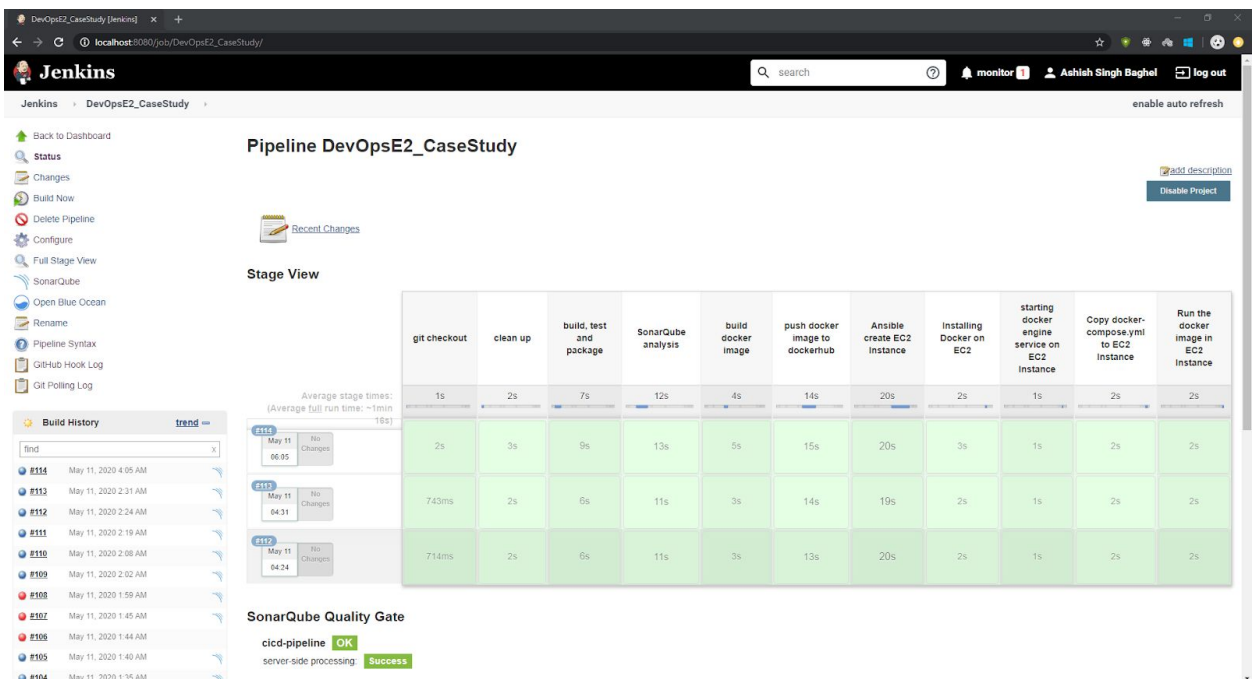
☒ Install automatically

[Install from Apache](#)

Version

[Delete installer](#)

- **Jenkins Build Pipeline** to trigger build and deployment of Spring Boot application



Entire pipeline is written as Groovy Script

**Pipeline**

Definition

Script

```

1 node {
2   try {
3     stage('git checkout'){
4       git credentialsId: 'git-creds', url: 'https://github.com/mechdeveloper/TCSDevOpsBootCamp.git'
5     }
6     stage('clean up'){
7       def mavenHome = tool name: 'maven3', type: 'maven'
8       def mavenCMD = "${mavenHome}/bin/mvn"
9       sh "${mavenCMD} clean"
10    }
11    stage('build, test and package'){
12      def mavenHome = tool name: 'maven3', type: 'maven'
13      def mavenCMD = "${mavenHome}/bin/mvn"
14      sh "${mavenCMD} package"
15    }
16  }
17 }

```

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

## Automatic Build triggers

The screenshot shows the 'Build Triggers' configuration page in Jenkins. It includes several checkboxes for build triggers: 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling', and 'Poll SCM' (which is checked). Below these is a 'Schedule' field with a text area and a note: 'No schedules so will only run due to SCM changes if triggered by a post-commit hook'. There is also an unchecked checkbox for 'Ignore post-commit hooks'. Further down, there are checkboxes for 'Disable this project', 'Quiet period', and 'Trigger builds remotely (e.g., from scripts)' (which is checked). At the bottom, there is an 'Authentication Token' field with the value 'DevOpsE2' and a text box containing instructions on how to use the token to trigger a build remotely via a Jenkins URL.

- Auto configuration of AWS EC2 Instances via Ansible -
- Running ansible command on a docker container with ansible installed -

```
// Run Ansible playbook here
stage ('Ansible create EC2 Instance'){
    withCredentials([string(credentialsId: 'AWS_ACCESS_KEY_ID', variable: 'AWS_ACCESS_KEY_ID'), string(credentialsId: 'AWS_SECRET_ACCESS_KEY', variable: 'AWS_SECRET_ACCESS_KEY')]){
        def ansibleCMD = "ansible-playbook task.yml"
        sh "docker run --rm -v \$(pwd):/ansible/playbooks --env AWS_ACCESS_KEY_ID=\${AWS_ACCESS_KEY_ID} --env AWS_SECRET_ACCESS_KEY=\${AWS_SECRET_ACCESS_KEY} ansibleCMD"
    }
}
```

This Build step runs task.yml playbook to configure EC2 Instance -  
task.yml explained -

- First task sets up a new security group to allow access over port 80 and 22
- Second task launches the AWS EC2 instance with key devops-ec2-key to allow ssh access to server via this key.
- Third task waits until ssh starts working for the AWS EC2 instance just created via ansible playbook.



```
task.yml > {} 0 > [ ] tasks > {} 2 > [ ] wait_for
1  ## Ansible Playbook
2  # 1) Creates a custom security group
3  # 2) Creates a new EC2 Instance
4  # 3) Waits for ssh to become active on Instance
5  #
6  # Version 1.0      Ashish Singh Baghel      May/07/2020
7  #
8  - name: ec2-launcher
9    hosts: localhost
10   gather_facts: false
11   connection : local
12
13  vars:
14    region: us-east-2
15    instance_type: t2.micro
16    ami: ami-097834fcb3081f51a
17    key_name: devops-ec2-key
18    env: test
19
20  tasks:
21    - name: Setting up Security/Firewall Group
22      ec2_group:
```


## - AWS Configuration -

Created non root Administrator user with admin access to AWS for Jenkins Integration via Access Key ID and and Secret Access Key

Users > Administrator

## Summary

---

**User ARN**    arn:aws:iam::146396079818:user/Administrator 

**Path**    /

**Creation time**    2020-05-03 01:29 UTC+0200


---

**Permissions**   Groups (1)   Tags   Security credentials   Access Advisor

---

▼ Permissions policies (1 policy applied)

**Add permissions**

Policy name ▼
Attached from group
▶  AdministratorAccess

---

▶ Permissions boundary (not set)

---

Created private key pair for AWS Region us-east-2 to enable access to EC2 machines deployed in that region

EC2 > Key pairs

### Key pairs (1/1)

<input checked="" type="checkbox"/>	Name ▼	Fingerprint
<input checked="" type="checkbox"/>	devops-ec2-key	0f:a5:06:90:89:d1:2a:97:41:27:0f:20:98:91:8b:4a:bf:9d:1c:43

## AWS EC2 Dashboard before deployment region us-east-2

EC2

Resources

You are using the following Amazon EC2 resources in the US East (Ohio) Region:

Running instances

0

Elastic IPs

0

Dedicated Hosts

0

Snapshots

0

Volumes

0

Load balancers

0

Key pairs

1

Security groups

1

Placement groups

0

Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. [Learn more](#)

## Triggering the build from jenkins -

Jenkins

search

monitor

Ashish Singh Baghel

log out

Jenkins

DevOpsE2\_CaseStudy

enable auto refresh

Back to Dashboard

Status

Changes

Build Now

Delete Pipeline

Configure

Full Stage View

SonarCube

Open Blue Ocean

Rename

Pipeline Syntax

Git Polling Log

Pipeline DevOpsE2\_CaseStudy

add description

Disable Project

Recent Changes

3 commits

Stage View

git checkout

clean up

build, test and package

SonarQube analysis

build docker image

push docker image to dockerhub

Ansible create EC2 instance

Installing Docker on EC2

starting docker engine service on EC2 instance

Copy docker-compose.yml to EC2 Instance

Run the docker image in EC2 Instance

Average stage times: (Average full run time: ~1min 18s)

1s

2s

6s

12s

4s

14s

20s

2s

1s

2s

2s

1min 7s

Build History

trend

find

May 11, 2020 5:27 AM

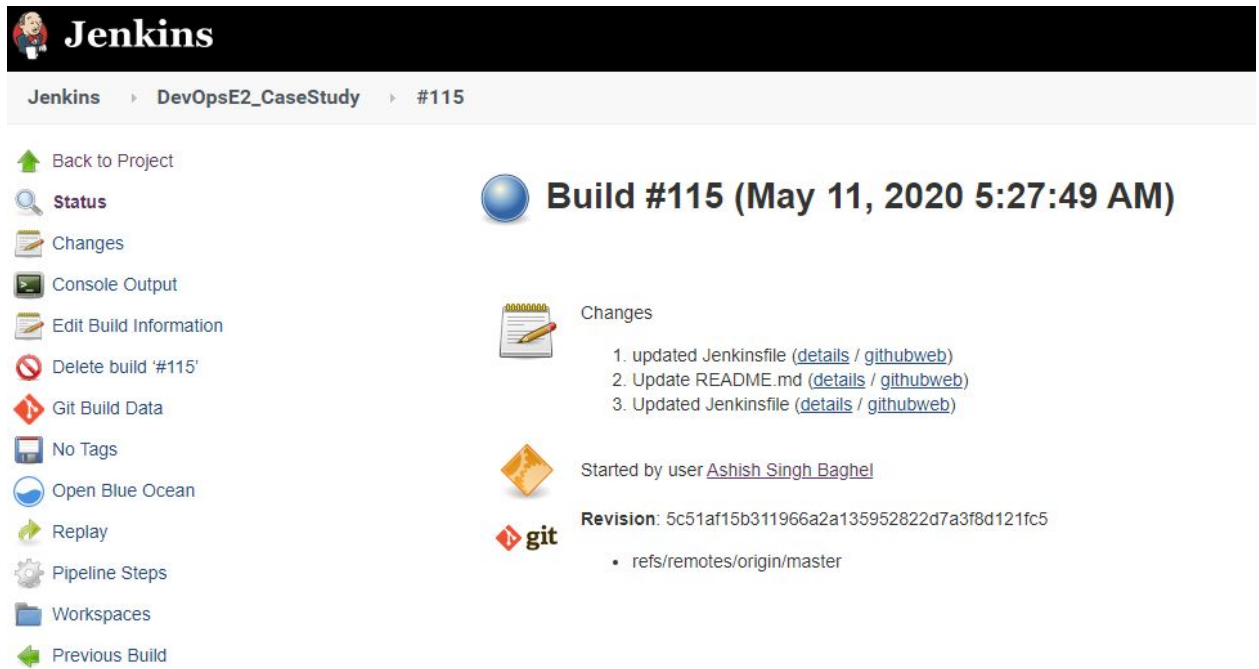
May 11, 2020 4:05 AM

May 11, 2020 2:31 AM

May 11, 2020 2:24 AM

May 11, 2020 2:19 AM

## Jenkins Build Successful -



The screenshot shows the Jenkins web interface for a successful build. The top navigation bar includes 'Jenkins', 'DevOpsE2\_CaseStudy', and '#115'. A sidebar on the left lists various actions: 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete build '#115'', 'Git Build Data', 'No Tags', 'Open Blue Ocean', 'Replay', 'Pipeline Steps', 'Workspaces', and 'Previous Build'. The main content area displays 'Build #115 (May 11, 2020 5:27:49 AM)'. Below this, the 'Changes' section lists three updates to the Jenkinsfile and README.md. The 'Started by user' section identifies the user as 'Ashish Singh Baghel'. The 'Revision' section shows the commit hash '5c51af15b311966a2a135952822d7a3f8d121fc5' and the branch 'refs/remotes/origin/master'.

**Jenkins**   ▸   DevOpsE2\_CaseStudy   ▸   #115

Back to Project

Status

Changes

Console Output

Edit Build Information

Delete build '#115'

Git Build Data

No Tags

Open Blue Ocean

Replay

Pipeline Steps

Workspaces

Previous Build

**Build #115 (May 11, 2020 5:27:49 AM)**

Changes

1. updated Jenkinsfile ([details](#) / [githubweb](#))
2. Update README.md ([details](#) / [githubweb](#))
3. Updated Jenkinsfile ([details](#) / [githubweb](#))

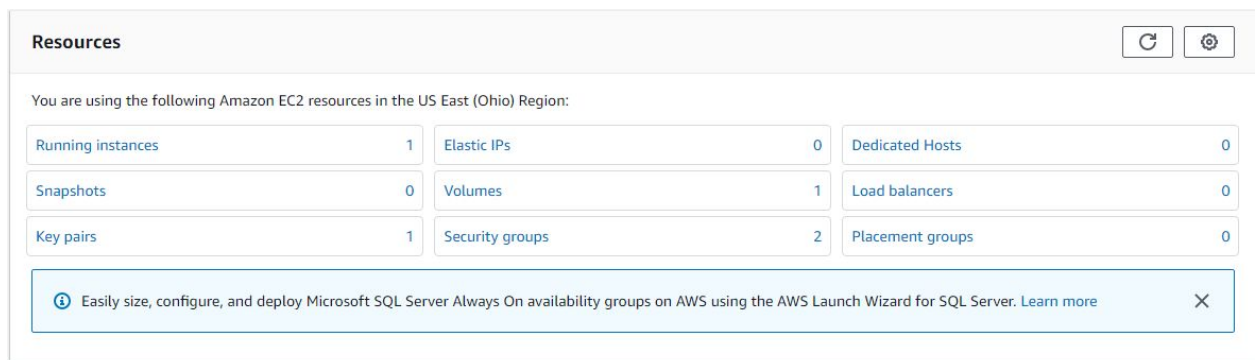
Started by user [Ashish Singh Baghel](#)

**Revision:** 5c51af15b311966a2a135952822d7a3f8d121fc5

- refs/remotes/origin/master

## Checking AWS EC2 Dashboard after successful build from Jenkins pipeline build

-



The screenshot shows the 'Resources' section of the AWS Management Console. It displays a table of Amazon EC2 resources used in the US East (Ohio) Region. The resources include 1 Running instance, 0 Elastic IPs, 0 Dedicated Hosts, 0 Snapshots, 1 Volumes, 0 Load balancers, 1 Key pairs, 2 Security groups, and 0 Placement groups. A notification banner at the bottom encourages using the AWS Launch Wizard for SQL Server.

**Resources**

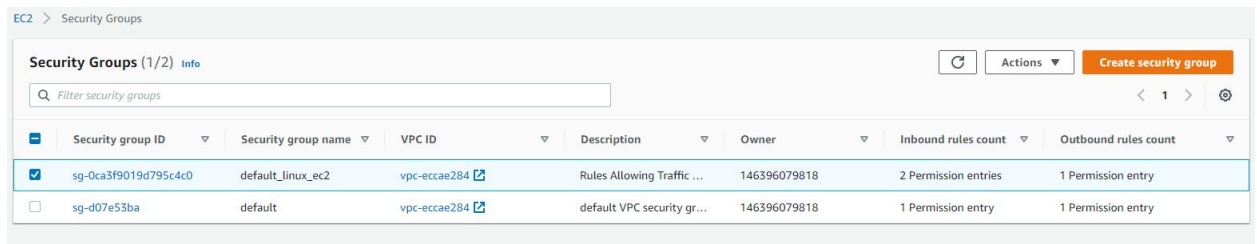
You are using the following Amazon EC2 resources in the US East (Ohio) Region:

Running instances	1	Elastic IPs	0	Dedicated Hosts	0
Snapshots	0	Volumes	1	Load balancers	0
Key pairs	1	Security groups	2	Placement groups	0

Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. [Learn more](#)

1 New running instance along with 1 New Volume and a brand new Security Group for our Instance -

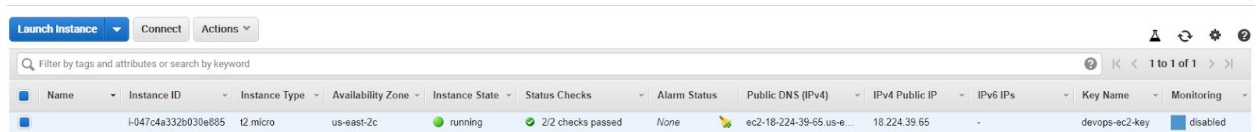
Checking the Newly created security group for our ec2 instance -



The screenshot shows the AWS Management Console 'Security Groups' page. It lists two security groups: 'default\_linux\_ec2' and 'default'. The 'default\_linux\_ec2' group is selected, showing its ID, name, VPC ID, description, owner, and rule counts.

Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count	Outbound rules count
sg-0ca3f9019d795c4c0	default_linux_ec2	vpc-eccae284	Rules Allowing Traffic ...	146396079818	2 Permission entries	1 Permission entry
sg-d07e53ba	default	vpc-eccae284	default VPC security gr...	146396079818	1 Permission entry	1 Permission entry

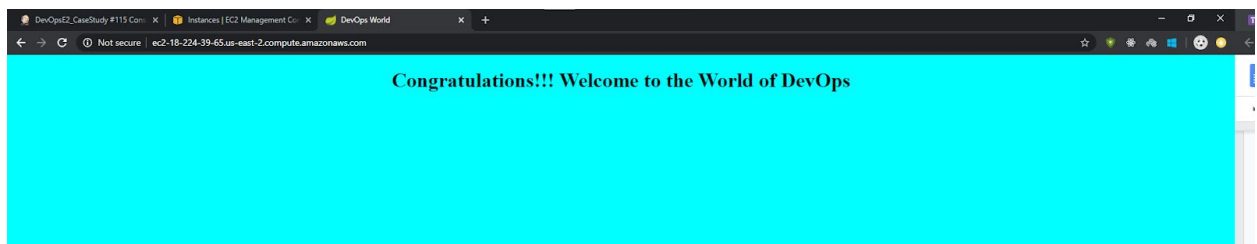
Checking new Running EC2 Instance -



The screenshot shows the AWS Management Console 'Instances' page. It displays a single running EC2 instance with its details, including instance ID, type, availability zone, state, status checks, alarm status, public DNS, IPv4 public IP, IPv6 IPs, key name, and monitoring status.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name	Monitoring
	i-047c4a332b030e685	t2.micro	us-east-2c	running	2/2 checks passed	None	ec2-18-224-39-65.us-e...	18.224.39.65	-	devops-ec2-key	disabled

- Accessing public DNS name of the newly created AWS EC2 instance



Our Application is successfully deployed via end to end automated jenkins pipeline.