CI/CD Deployment for SpringBoot Application.

(Sprint Work and Project Specification)

Developer: *Tushar Khillare*

- Version History:

1.	**Author Name**	**Tushar Ashok Khillare.**
2.	**Purpose**	Specification of Project and Sprint Work.
3.	**Date**	19 January 2022.
4.	**Version**	1.0
5.	**Contact**	*tusharkhillare2015@gmail.com*

Contents.

1. Modules in the project

- CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams
- In this project to deploy CI/CD first we will be creating Spring Boot application in eclipse and commit it into the git hub account.
- Main thing which we must take care of is Docker File as we have to create an image of the war file using Jenkins
- Once GitHub is ready with all the file configuring the Jenkins with details of the git hub account and docker export to create a image file in Docker hub.
- Once done with the image go to AWS EC2 and launch an instinct and run the docker file in AWS ECS instance.
- The spring boot application can be access using the public IP address from anywhere.

2. Core concepts

- Spring Boot
- GitHub
- Jenkins
- Docker
- AWS EC2

3. Important URLs:

Repository Name	Phase-5-CI-CD-Deployment
GitHub Link	https://github.com/tuskhillare/Phase-5-CI-CD-Deployment.git

- ***Host server app to view the deployed application:***
  http://3.91.182.20:7000/

4. Sprint Wise Work

Sr no	Sprint Number	Sprint Module
1.	1	Created an EC2 instance and started Jenkins on it: An ec2 instance which has configured security group that allows users to access it through http protocol on port 80 and then install Java, Jenkins, maven, git and docker on it and make it accessible on port 8080 (default port of Jenkins server).
2.	2	Made a springboot application: A springboot application that has following request URLs. a. /greeting -Which will show a sample greeting page. b. / -sample index page c. /hello another sample page. d. It has Dockerfile with required dependency JDK-8
3.	3	Upload it on GitHub: Tracked the springboot application with version controlling system i.e., Git and then connected it with remote repository on Source code management system i.e., GitHub in the main branch.
4.	4	Make a host ec2 instance: An ec2 instance which has configured security group that allows users to access it through http protocol on port 80 and install docker on it to run docker container.

- **Created a Jenkins Pipeline job:**

  Created a pipeline job with steps:

  1) SCM pull: pulls the source code from GitHub of the created spring boot app.
  2) Maven package: packages the source code pulled by step 1.
  3) Docker Build: makes an image of the package created by Maven package with dockerfile in that spring boot application.
  4) Docker push: push the created image to dockerhub.
  5) SSH login to Host ec2 instance through Jenkins credentials that has private key for the SSH.
  6) Pull the docker image uploaded by Docker push step and run it as docker container and while running map the internal 8081 port (default spring boot app server port) to that ec2 instance's port 7000.
  7) Running application can accessed by ec2 public Ip: 7000 port.

5. Application Unique Selling Points

- The application is deployed using the "CI/CD" refers to continuous delivery and/or continuous deployment.
- **Effective, simple, and powerful** – Whenever there is any commit into the GitHub account it will automatically build and deploy the latest image.
- **Instant feedback** - No more waiting or extra work to code, push, integration and other thing.
- **Actionable insights** – Just push the code into the GitHub and CI/CD will take care of everything automatically.
- The application is designed with modularity in mind. Making work of the user very easy by providing the automation of the application.
- CI-CD product feature velocity is high. The high velocity improves the time spent investigating and patching defects.

End-user involvement and feedback during continuous development leads to usability improvements. User can add new requirements based on customer's needs on a daily basis.

6. Conclusion

The purpose of this Application was to deploy a CI/CD project which introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment. Taken together, these connected practices are often referred to as a "CI/CD pipeline" continuous delivery and/or continuous deployment and are supported by development and operations teams working together in an agile way with either a DevOps or site reliability engineering (SRE) approach