**Project report on**

**CI/CD Deployment for Springboot Application**

This document contains sections for:

* [Sprint planning and Task completion](#Sprint_plan)
* [Core concepts used in project](#Core_concepts).
* Technologies used in project.
* [Flow of the Application](#Flow).
* [Demonstrating the product capabilities, appearance, and user interactions.](#Product_capability)

The code for this project is hosted at

<https://github.com/YogeshSairam/Phase5.git>

The project is developed by Yogesh Sairam Pentakota.

## **Sprints planning and Task completion:**

The project is planned to be completed in 2 sprint. Tasks assumed to be completed in the sprints are:

* Creating the flow of the application
* Initializing git repository to track changes as development progresses.
* Writing the program to fulfill the requirements of the project.
* Testing the program with different kinds of User input
* Pushing code to GitHub.
* Creating this specification document highlighting application capabilities, appearance, and user interactions.

## **Core concepts used in project:**

• Deployment: to deploy the local project to the end-users.

• Virtual Machine: use virtual instances to help to build, deploy and manage websites.

• Exception Handling: used to catch problems that arises in the code especially in I/O blocks. • Single Web Page: apply the concept of a website that only contains one HTML page.

• Object-Oriented: used to create and model objects for users and their credentials.

**Technologies Used:**

• AWS EC2 instance: to use the instances as a VM and deploy the application

• Jenkins: to build the project from GitHub.

• GitHub: to upload the source code of the project.

## **Flow of the Application:**



**Project Users Stories : ( Agile and Scrum )**

* As a user I want an automated integration of a Spring boot Application.
* As a user I want an automated deployment of a Spring boot Application.
* As a developer I want to automate the integration of a Spring boot Application for the user.
* As a developer I want to automate the deployment of a Spring boot Application for the user.

1. **SPRINTS**

**Sprint 1**

* Understanding the problem statement of the project .
* Creating the flow chart of the project.
* Creating Maven Project.
* Creating Spring boot Application.
* Adding necessary dependencies.
* Testing at each step for different user inputs.
* Initializing the git repository.
* Pushing the code to the GitHub.
* Creating AWS EC2 instance.
* Downloading MobaXterm.
* Downloading Jenkins.
* Deploying the application on Jenkins.
* Creating the Specification document for deploying the project.

## 

## **Demonstrating the product capabilities, appearance, and user interactions:**

To demonstrate the product capabilities, below are the sub-sections configured to highlight appearance and user interactions for the project:

## **Step 1:** Creating a new project in Eclipse

* Open Eclipse
* Go to File -> New -> Project -> Maven Project -> Next.
* Type in any project name and click on “Finish.”

**Files:**

**Src/main/java/com/SpringTest/SpringApplication.java:**

**package** com.SpringTest;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringJenkinsApplication {

**public** **static** Logger log = LoggerFactory.getLogger(SpringJenkinsApplication.**class**);

**public** **void** init() {

log.info("Spring Boot Application Started.......");

}

**public** **static** **void** main(String[] args) {

log.info("Application Executed .......");

SpringApplication.run(SpringJenkinsApplication.**class**, args);

}

}

**Src/test/java/com/SpringTest/SpringAplicationTest.java:**

package com.SpringTest;

import org.junit.jupiter.api.Test;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class SpringJenkinsApplicationTests {

public static Logger log = LoggerFactory.getLogger(SpringJenkinsApplication.class);

@Test

void contextLoads() {

log.info("Spring Test Case Executing......");

}

}

**META-INF/maven/com.SpringTest/Testing-Spring-Jenkins/pom.properties:**

#Generated by Maven Integration for Eclipse

#Tue May 10 13:03:45 IST 2022

m2e.projectLocation=C\:\\Users\\yogesh\\Desktop\\phase 5 project\\CI-CD-Deployment-for-Springboot-Application

m2e.projectName=Spring-Jenkins

groupId=com.SpringTest

artifactId=Testing-Spring-Jenkins

version=0.0.1-SNAPSHOT

**META-INF/maven/com.SpringTest/Testing-Spring-Jenkins/pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.5.4</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.SpringTest</groupId>

<artifactId>Testing-Spring-Jenkins</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Spring-Jenkins</name>

<description> Spring Boot -Jenkins</description>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**MANIFEST.MF:**

Manifest-Version: 1.0

Build-Jdk-Spec: 13

Implementation-Title: Spring-Jenkins

Implementation-Version: 0.0.1-SNAPSHOT

Created-By: Maven Integration for Eclipse

**Maven-archiver/pom.properties:**

artifactId=Testing-Spring-Jenkins

groupId=com.SpringTest

version=0.0.1-SNAPSHOT

**mvnw:**

#!/bin/sh

# ----------------------------------------------------------------------------

# Licensed to the Apache Software Foundation (ASF) under one

# or more contributor license agreements. See the NOTICE file

# distributed with this work for additional information

# regarding copyright ownership. The ASF licenses this file

# to you under the Apache License, Version 2.0 (the

# "License"); you may not use this file except in compliance

# with the License. You may obtain a copy of the License at

#

# https://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing,

# software distributed under the License is distributed on an

# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

# KIND, either express or implied. See the License for the

# specific language governing permissions and limitations

# under the License.

# ----------------------------------------------------------------------------

# ----------------------------------------------------------------------------

# Maven Start Up Batch script

#

# Required ENV vars:

# ------------------

# JAVA\_HOME - location of a JDK home dir

#

# Optional ENV vars

# -----------------

# M2\_HOME - location of maven2's installed home dir

# MAVEN\_OPTS - parameters passed to the Java VM when running Maven

# e.g. to debug Maven itself, use

# set MAVEN\_OPTS=-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=8000

# MAVEN\_SKIP\_RC - flag to disable loading of mavenrc files

# ----------------------------------------------------------------------------

if [ -z "$MAVEN\_SKIP\_RC" ] ; then

if [ -f /etc/mavenrc ] ; then

. /etc/mavenrc

fi

if [ -f "$HOME/.mavenrc" ] ; then

. "$HOME/.mavenrc"

fi

fi

# OS specific support. $var \_must\_ be set to either true or false.

cygwin=false;

darwin=false;

mingw=false

case "`uname`" in

CYGWIN\*) cygwin=true ;;

MINGW\*) mingw=true;;

Darwin\*) darwin=true

# Use /usr/libexec/java\_home if available, otherwise fall back to /Library/Java/Home

# See https://developer.apple.com/library/mac/qa/qa1170/\_index.html

if [ -z "$JAVA\_HOME" ]; then

if [ -x "/usr/libexec/java\_home" ]; then

export JAVA\_HOME="`/usr/libexec/java\_home`"

else

export JAVA\_HOME="/Library/Java/Home"

fi

fi

;;

esac

if [ -z "$JAVA\_HOME" ] ; then

if [ -r /etc/gentoo-release ] ; then

JAVA\_HOME=`java-config --jre-home`

fi

fi

if [ -z "$M2\_HOME" ] ; then

## resolve links - $0 may be a link to maven's home

PRG="$0"

# need this for relative symlinks

while [ -h "$PRG" ] ; do

ls=`ls -ld "$PRG"`

link=`expr "$ls" : '.\*-> \(.\*\)$'`

if expr "$link" : '/.\*' > /dev/null; then

PRG="$link"

else

PRG="`dirname "$PRG"`/$link"

fi

done

saveddir=`pwd`

M2\_HOME=`dirname "$PRG"`/..

# make it fully qualified

M2\_HOME=`cd "$M2\_HOME" && pwd`

cd "$saveddir"

# echo Using m2 at $M2\_HOME

fi

# For Cygwin, ensure paths are in UNIX format before anything is touched

if $cygwin ; then

[ -n "$M2\_HOME" ] &&

M2\_HOME=`cygpath --unix "$M2\_HOME"`

[ -n "$JAVA\_HOME" ] &&

JAVA\_HOME=`cygpath --unix "$JAVA\_HOME"`

[ -n "$CLASSPATH" ] &&

CLASSPATH=`cygpath --path --unix "$CLASSPATH"`

fi

# For Mingw, ensure paths are in UNIX format before anything is touched

if $mingw ; then

[ -n "$M2\_HOME" ] &&

M2\_HOME="`(cd "$M2\_HOME"; pwd)`"

[ -n "$JAVA\_HOME" ] &&

JAVA\_HOME="`(cd "$JAVA\_HOME"; pwd)`"

fi

if [ -z "$JAVA\_HOME" ]; then

javaExecutable="`which javac`"

if [ -n "$javaExecutable" ] && ! [ "`expr \"$javaExecutable\" : '\([^ ]\*\)'`" = "no" ]; then

# readlink(1) is not available as standard on Solaris 10.

readLink=`which readlink`

if [ ! `expr "$readLink" : '\([^ ]\*\)'` = "no" ]; then

if $darwin ; then

javaHome="`dirname \"$javaExecutable\"`"

javaExecutable="`cd \"$javaHome\" && pwd -P`/javac"

else

javaExecutable="`readlink -f \"$javaExecutable\"`"

fi

javaHome="`dirname \"$javaExecutable\"`"

javaHome=`expr "$javaHome" : '\(.\*\)/bin'`

JAVA\_HOME="$javaHome"

export JAVA\_HOME

fi

fi

fi

if [ -z "$JAVACMD" ] ; then

if [ -n "$JAVA\_HOME" ] ; then

if [ -x "$JAVA\_HOME/jre/sh/java" ] ; then

# IBM's JDK on AIX uses strange locations for the executables

JAVACMD="$JAVA\_HOME/jre/sh/java"

else

JAVACMD="$JAVA\_HOME/bin/java"

fi

else

JAVACMD="`which java`"

fi

fi

if [ ! -x "$JAVACMD" ] ; then

echo "Error: JAVA\_HOME is not defined correctly." >&2

echo " We cannot execute $JAVACMD" >&2

exit 1

fi

if [ -z "$JAVA\_HOME" ] ; then

echo "Warning: JAVA\_HOME environment variable is not set."

fi

CLASSWORLDS\_LAUNCHER=org.codehaus.plexus.classworlds.launcher.Launcher

# traverses directory structure from process work directory to filesystem root

# first directory with .mvn subdirectory is considered project base directory

find\_maven\_basedir() {

if [ -z "$1" ]

then

echo "Path not specified to find\_maven\_basedir"

return 1

fi

basedir="$1"

wdir="$1"

while [ "$wdir" != '/' ] ; do

if [ -d "$wdir"/.mvn ] ; then

basedir=$wdir

break

fi

# workaround for JBEAP-8937 (on Solaris 10/Sparc)

if [ -d "${wdir}" ]; then

wdir=`cd "$wdir/.."; pwd`

fi

# end of workaround

done

echo "${basedir}"

}

# concatenates all lines of a file

concat\_lines() {

if [ -f "$1" ]; then

echo "$(tr -s '\n' ' ' < "$1")"

fi

}

BASE\_DIR=`find\_maven\_basedir "$(pwd)"`

if [ -z "$BASE\_DIR" ]; then

exit 1;

fi

##########################################################################################

# Extension to allow automatically downloading the maven-wrapper.jar from Maven-central

# This allows using the maven wrapper in projects that prohibit checking in binary data.

##########################################################################################

if [ -r "$BASE\_DIR/.mvn/wrapper/maven-wrapper.jar" ]; then

if [ "$MVNW\_VERBOSE" = true ]; then

echo "Found .mvn/wrapper/maven-wrapper.jar"

fi

else

if [ "$MVNW\_VERBOSE" = true ]; then

echo "Couldn't find .mvn/wrapper/maven-wrapper.jar, downloading it ..."

fi

if [ -n "$MVNW\_REPOURL" ]; then

jarUrl="$MVNW\_REPOURL/io/takari/maven-wrapper/0.5.6/maven-wrapper-0.5.6.jar"

else

jarUrl="https://repo.maven.apache.org/maven2/io/takari/maven-wrapper/0.5.6/maven-wrapper-0.5.6.jar"

fi

while IFS="=" read key value; do

case "$key" in (wrapperUrl) jarUrl="$value"; break ;;

esac

done < "$BASE\_DIR/.mvn/wrapper/maven-wrapper.properties"

if [ "$MVNW\_VERBOSE" = true ]; then

echo "Downloading from: $jarUrl"

fi

wrapperJarPath="$BASE\_DIR/.mvn/wrapper/maven-wrapper.jar"

if $cygwin; then

wrapperJarPath=`cygpath --path --windows "$wrapperJarPath"`

fi

if command -v wget > /dev/null; then

if [ "$MVNW\_VERBOSE" = true ]; then

echo "Found wget ... using wget"

fi

if [ -z "$MVNW\_USERNAME" ] || [ -z "$MVNW\_PASSWORD" ]; then

wget "$jarUrl" -O "$wrapperJarPath"

else

wget --http-user=$MVNW\_USERNAME --http-password=$MVNW\_PASSWORD "$jarUrl" -O "$wrapperJarPath"

fi

elif command -v curl > /dev/null; then

if [ "$MVNW\_VERBOSE" = true ]; then

echo "Found curl ... using curl"

fi

if [ -z "$MVNW\_USERNAME" ] || [ -z "$MVNW\_PASSWORD" ]; then

curl -o "$wrapperJarPath" "$jarUrl" -f

else

curl --user $MVNW\_USERNAME:$MVNW\_PASSWORD -o "$wrapperJarPath" "$jarUrl" -f

fi

else

if [ "$MVNW\_VERBOSE" = true ]; then

echo "Falling back to using Java to download"

fi

javaClass="$BASE\_DIR/.mvn/wrapper/MavenWrapperDownloader.java"

# For Cygwin, switch paths to Windows format before running javac

if $cygwin; then

javaClass=`cygpath --path --windows "$javaClass"`

fi

if [ -e "$javaClass" ]; then

if [ ! -e "$BASE\_DIR/.mvn/wrapper/MavenWrapperDownloader.class" ]; then

if [ "$MVNW\_VERBOSE" = true ]; then

echo " - Compiling MavenWrapperDownloader.java ..."

fi

# Compiling the Java class

("$JAVA\_HOME/bin/javac" "$javaClass")

fi

if [ -e "$BASE\_DIR/.mvn/wrapper/MavenWrapperDownloader.class" ]; then

# Running the downloader

if [ "$MVNW\_VERBOSE" = true ]; then

echo " - Running MavenWrapperDownloader.java ..."

fi

("$JAVA\_HOME/bin/java" -cp .mvn/wrapper MavenWrapperDownloader "$MAVEN\_PROJECTBASEDIR")

fi

fi

fi

fi

##########################################################################################

# End of extension

##########################################################################################

export MAVEN\_PROJECTBASEDIR=${MAVEN\_BASEDIR:-"$BASE\_DIR"}

if [ "$MVNW\_VERBOSE" = true ]; then

echo $MAVEN\_PROJECTBASEDIR

fi

MAVEN\_OPTS="$(concat\_lines "$MAVEN\_PROJECTBASEDIR/.mvn/jvm.config") $MAVEN\_OPTS"

# For Cygwin, switch paths to Windows format before running java

if $cygwin; then

[ -n "$M2\_HOME" ] &&

M2\_HOME=`cygpath --path --windows "$M2\_HOME"`

[ -n "$JAVA\_HOME" ] &&

JAVA\_HOME=`cygpath --path --windows "$JAVA\_HOME"`

[ -n "$CLASSPATH" ] &&

CLASSPATH=`cygpath --path --windows "$CLASSPATH"`

[ -n "$MAVEN\_PROJECTBASEDIR" ] &&

MAVEN\_PROJECTBASEDIR=`cygpath --path --windows "$MAVEN\_PROJECTBASEDIR"`

fi

# Provide a "standardized" way to retrieve the CLI args that will

# work with both Windows and non-Windows executions.

MAVEN\_CMD\_LINE\_ARGS="$MAVEN\_CONFIG $@"

export MAVEN\_CMD\_LINE\_ARGS

WRAPPER\_LAUNCHER=org.apache.maven.wrapper.MavenWrapperMain

exec "$JAVACMD" \

$MAVEN\_OPTS \

-classpath "$MAVEN\_PROJECTBASEDIR/.mvn/wrapper/maven-wrapper.jar" \

"-Dmaven.home=${M2\_HOME}" "-Dmaven.multiModuleProjectDirectory=${MAVEN\_PROJECTBASEDIR}" \

${WRAPPER\_LAUNCHER} $MAVEN\_CONFIG "$@"

**Pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.5.4</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.SpringTest</groupId>

<artifactId>Testing-Spring-Jenkins</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Spring-Jenkins</name>

<description> Spring Boot -Jenkins</description>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**OUTPUTS:**

# Eclipse





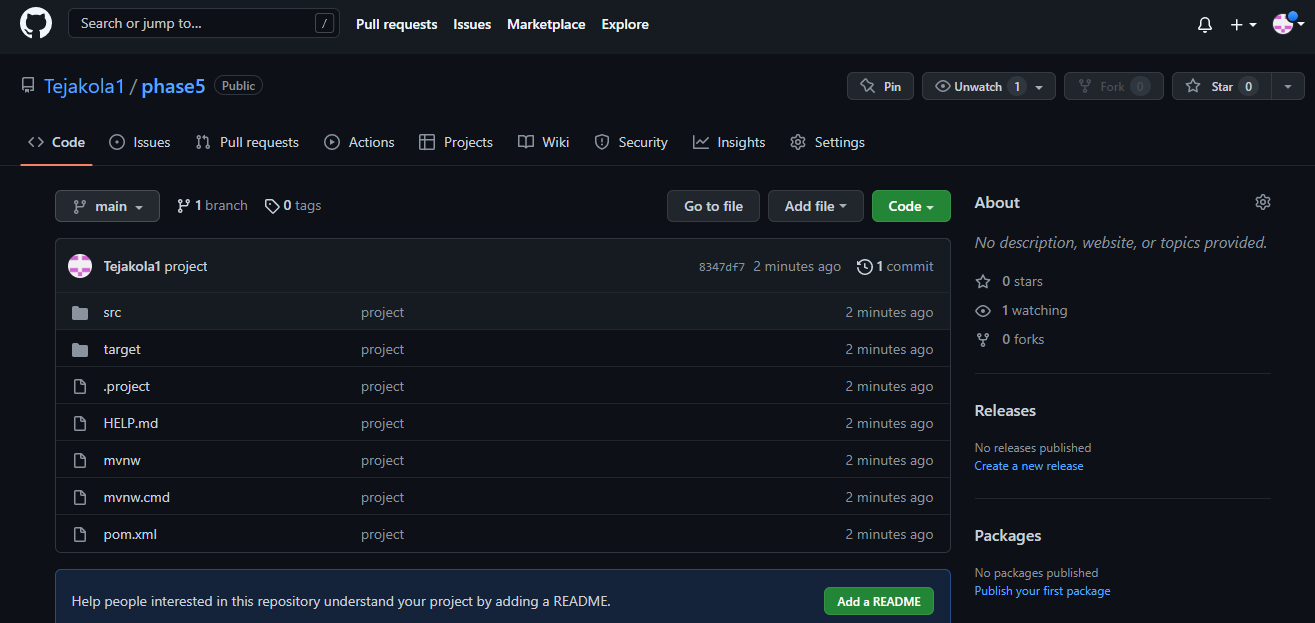
# I create Simple Spring boot Application to build a CI/CD pipeline to demonstrate continuous deployment:

## 





# Git Hub: Application Deploy on git Hub



**Github repository:**

<https://github.com/YogeshSairam/Phase5.git>

# Jenkins : Run Jenkins



**CREATE THREE JENKINS PROJECT DEPLOYMENT:-**

**DEPLOYMENT-TESTING-PRODUCTION**





**Final Output:-**

**Last host the application on AWS EC2 instance:**



## **Step 2:** **Pushing the code to GitHub repository**

* Open your command prompt and navigate to the folder where you have created your files.

*cd <folder path>*

* Initialize repository using the following command:

*git init*

* Add all the files to your git repository using the following command:

*git add .*

* Commit the changes using the following command:

*git commit . -m <commit message>*

* Push the files to the folder you initially created using the following command:

*git push -u origin master.*