Intro to Software Engineering
ECSE 321

Kevin Chuong, 260742781
Jeremy Davis, 260744431
Oscar Décéus, 260744646
Elias Al Homsi, 260797449

DELIVERABLE #4

Release Pipeline Plan

Presented to Mr. Daniel Varro
Faculty of Engineering

McGill University March 25, 2018

GitHub link: https://github.mcgill.ca/ECSE321-2018-Winter/Project-13/commit/3e4cf351f26bea340d434338dfb77f1453a23be0

Contents

Release Pipeline Plan

Introduction:

The Release status is fully automated by integrating new code and features to building them on the server **ecse321-12.ece.mcgill.ca** using Jenkins, gradle and bash scripts to automate the process. Once the building is done the application is deployment ready and files are copied to the right location and Linux services are restarted.

Integration

Integration is the process by which new features and code are introduced to the system. The team is currently using GitHub as the version control software. The workflow consists of 3 different branches. Branch development which is cutting edge and where all the new features are tested. Branch master is the current stable version. Lastly, branch release is our software release version which is deployment ready. Jenkins checks for the branch called release for new software to deploy.

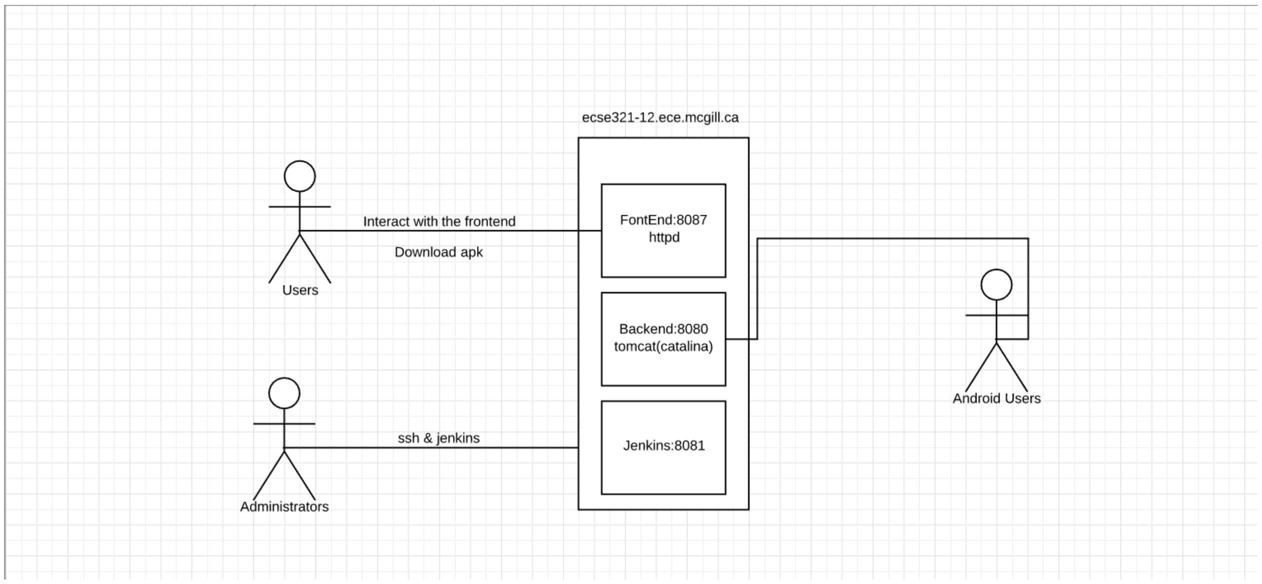
Build

The build step is done by gradle for both the spring application and the Android application. The gradle wrapper is used for the android application to build a signed apk which is then used to put on the website front-end to be available for download for users. The front-end is built by building a production version of the application and therefore, making the JavaScript and css execution far more efficient than the development version. The front-end is deployed on the httpd server.

Deployment

The deployment phase is different for each subsystem of the application. The deployment of the backend includes restarting the tomcat server and running an instance of the backend using port 8080. Ngrok was initially used to test connection on non-exposed ports. Ngrok is an application port exposer that makes it possible to expose the backend through a unique link. The frontend of httpd is running on port 8087 and the vuejs production files that were generated using the command npm run build are then copied into /var/www/html and the right access rights are given using chown and chmod. The main reason behind running the frontend on port 8087 is that there were issues related to exposing port 8087 (SE LINUX security for the service httpd). However, they were fixed using semanage and iptables. The android apk is then copied to the /var/www/HTML to be available for download.

Diagram:



Description of Integration Phase:

Tools used:

- GitHub
- Eclipse
- Android studio
- Sublime text for web

Work Flow:

The workflow starts by suggesting a new feature that the team is required to build. The modifications are first done on the development branch in full code sprints. The technique being used Test-driven development that is We keep iterating on the development branch until the code has passed initial tests. Once that done the development branch is merged back to the master branch. The code is reviewed and refactored for additional checking and to make sure that it integrates well with other components of the system. Once that done, the code is ready to be deployed and it is merged into the release branch. During the integration phase, automated tests are executed on the code to validate its integrity.

Description of Build Phase:

Note:

All build scripts used are found on GitHub and attached to the end of this report.

Tools used:

- Jenkins
- Gradle
- Npm /vuejs
- Bash programming language

Work Flow:

Once the code is pushed to the release branch on Github. Jenkins has automatic checking on that branch which executes every minute. Jenkins then pulls the code automatically into a clean work environment and does the following steps in this specific order.

1. Building the war file of the spring application. The gradle file responsible for building the spring application and therefore the war file which is required by the tomcat server.
2. Once the gradle build is finished the script launches and does automatic deployment and building. The script builds the website frontend using the command npm run build which generates HTML pages that are far more efficient than the development version.
3. The last step is building the android application using the gradlew (the gradle wrapper). For this, the Android SDK was required under /home/student/Android/SDK which was copied from the development machine.
   Command for building: **sudo ./gradlew assembleRelease**
   This gradle task would generate the apk and sign it under the directory app/release/release.apk

   This concludes the building phase. The script does additional stuff however, they are part of the automatic deployment.

Description of Deployment Phase

Note:

All build scripts used are found on GitHub and attached to the end of this report.

Tools used:

- Jenkins
- Gradle
- Npm /vuejs
- Bash programming language
- Ngrok (a port exposer)
- Httpd
- Tomcat (Catalina server)
- Chown / chmod

Work Flow:

Jenkins was executing the bash script from the previous phase. Jenkins also does deployment whenever the tool finishes building some component. For this part, Jenkins requires special access to the system (root-like access) which was fixed by adding it to the sudoers group.

Jenkins starts deploying in the following phases:

1. **Deploying The backend**
   - **a.** The first step is copying the war file generated by gradle in the previous step to the right location of tomcat/libs/webapps
   - **b.** The tomcat server is stopped using systemctl stop tomcat
   - **c.** For some issues with Tomcat, Catalina is run directly without the help systemd help by running Catalina.sh start **(which configured to run on port 8080 using server.xml)**
   - **d.** The backend should now be running on its designated port of 8080.
   - **e.** Ngrok was run for testing purposes by linking the 8088 port on the local machine to a generated URL. This URL would be then used to access the backend comfortably. The configuration file for Ngrok is called Ngrok.yml. the command for running it without interruption (as nohup service) is
   - **f.** **sudo nohup ./ngrok start backend -config=/home/ecse321/.ngrok2/ngrok.yml log=stdout &**
   - **g.** **successfully completing this stage** the spring should be running on 9e68ed46.ngrok.io **which routes back to port 8088 (for testing only)**

2. **Deploying Android application:**
    a. **Deploying** the android application is simply copying the apk file to the /var/www/html and giving it the right access right. The frontend would later provide a link to download the apk
    b. **sudo cp -rf /home/ecse321/workspace/TreePLE/TreePLE/TreePLEAndroid/app/release/app-release.apk /var/www/html/**

3. **Deploying The frontend**
    a. After npm run build is finished (takes time). The directory /dist is then copied to /var/www/html
    b. Chown is used to giving ownership to the apache user.
    c. Httpd is then restarted using: sudo systemctl restart httpd
    d. The front-end is currently using port 8087 by configuring the apache configuration files. Https could be used for future iterations.

In general, once pushed to the release branch. All those steps are done automatically and a running application is then shown to the user.

Work Plan and Individual work description:

What's done:

1. The automated pipeline (Oscar (gradle android), Jeremy (gradle Spring), Kevin (tomcat server), Elias (bash script))
2. Displaying trees on map, and frontend Programming (Elias)
3. Unit Testing the Backend (Oscar, Jeremy, Kevin, Elias)
4. A starting template for the android application (Elias)

To Do:

1. Android application functionalities: including login/ map and marking trees.
2. Making the web-frontend more beautiful
3. More test cases for the backend
4. Potential mysql usage

Deployment scripts:

Spring gradle:

```
buildscript {
	ext {
		springBootVersion = '1.5.9.RELEASE'
	}
	repositories {
		mavenCentral()
	}
	dependencies {
		classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
	}
}

apply plugin: 'java' apply plugin: 'eclipse'
apply plugin: 'org.springframework.boot'
apply plugin: 'war'

group = 'ca.mcgill.ecse321' version
= '0.0.1-SNAPSHOT'
sourceCompatibility = 1.8



war {
	baseName = 'eventregistration'
version = '0.0.1-SNAPSHOT'
}
```

```
repositories {
            mavenCentral()
}

configurations {
            providedRuntime
}

dependencies {              compile('org.modelmapper:modelmapper:1.1.1')
compile('org.springframework.boot:spring-boot-starter-web')
compile("org.springframework.boot:spring-boot-starter-actuator")
compile('org.springframework:spring-webmvc:4.2.6.RELEASE')
compile('com.google.guava:guava:23.5-jre')
compile('com.thoughtworks.xstream:xstream:1.4.7')

            // Use MySQL Connector-J
   compile 'mysql:mysql-connector-java'

            // https://mvnrepository.com/artifact/org.mindrot/jbcrypt
compile group: 'org.mindrot', name: 'jbcrypt', version: '0.3m'

            providedRuntime('org.springframework.boot:spring-boot-starter-tomcat')

            testCompile('org.springframework.boot:spring-boot-starter-test')
testCompile group: 'junit', name: 'junit', version: '4.12'
}

eclipse {
   classpath {
       containers.remove('org.eclipse.jdt.launching.JRE_CONTAINER')
       containers
'org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/J
avaSE-1.8'
   }
}
```

Android Gradlew Script:

```
#!/usr/bin/env bash

############################################################################## ##
##  Gradle start up script for UN*X
##
##############################################################################

# Add default JVM options here. You can also use JAVA_OPTS and GRADLE_OPTS to pass JVM options to
this script.
DEFAULT_JVM_OPTS=""

APP_NAME="Gradle"
APP_BASE_NAME=`basename "$0"`

# Use the maximum available, or set MAX_FD != -1 to use that value.
MAX_FD="maximum"

warn ( ) {
echo "$*"
}
```

```
die ( ) {
echo
echo "$*"
echo
   exit 1
}

# OS specific support (must be 'true' or 'false').
cygwin=false
msys=false
darwin=false case
"`uname`" in
CYGWIN* )
cygwin=true
   ;;
  Darwin* )
darwin=true
   ;;
  MINGW* )
   msys=true
   ;;
esac

# Attempt to set APP_HOME
# Resolve links: $0 may be a link
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"`"/$link"
   fi
done SAVED="`pwd`" cd "`dirname
\"$PRG\"`/" >/dev/null
APP_HOME="`pwd -P`" cd "$SAVED"
>/dev/null

CLASSPATH=$APP_HOME/gradle/wrapper/gradle-wrapper.jar

# Determine the Java command to use to start the JVM.
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
   if [ ! -x "$JAVACMD" ] ; then         die "ERROR: JAVA_HOME is set to an
invalid directory: $JAVA_HOME

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
   fi
else
   JAVACMD="java"
   which java >/dev/null 2>&1 || die "ERROR: JAVA_HOME is not set and no 'java' command could be
found in your PATH.


Please set the JAVA_HOME variable in your environment to match the location

of your Java installation."

fi

# Increase the maximum file descriptors if we can. if

[ "$cygwin" = "false" -a "$darwin" = "false" ] ; then

MAX_FD_LIMIT=`ulimit -H -n`

   if [ $? -eq 0 ] ; then        if [ "$MAX_FD" = "maximum" -o

"$MAX_FD" = "max" ] ; then            MAX_FD="$MAX_FD_LIMIT"

      fi

      ulimit -n $MAX_FD

if [ $? -ne 0 ] ; then

         warn "Could not set maximum file descriptor limit: $MAX_FD"

      fi

   else

      warn "Could not query maximum file descriptor limit: $MAX_FD_LIMIT"

   fi fi


# For Darwin, add options to specify how the application appears in the dock if

$darwin; then

   GRADLE_OPTS="$GRADLE_OPTS \"-Xdock:name=$APP_NAME\"
\"Xdock:icon=$APP_HOME/media/gradle.icns\""

fi


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

$cygwin ; then

   APP_HOME=`cygpath --path --mixed "$APP_HOME"`
```

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


  # We build the pattern for arguments to be converted via cygpath
  ROOTDIRSRAW=`find -L / -maxdepth 1 -mindepth 1 -type d 2>/dev/null`
SEP=""    for dir in $ROOTDIRSRAW ; do
    ROOTDIRS="$ROOTDIRS$SEP$dir"
    SEP="|"
done
  OURCYGPATTERN="(^($ROOTDIRS))"
  # Add a user-defined pattern to the cygpath arguments
if [ "$GRADLE_CYGPATTERN" != "" ] ; then
    OURCYGPATTERN="$OURCYGPATTERN|($GRADLE_CYGPATTERN)"
  fi
  # Now convert the arguments - kludge to limit ourselves to /bin/sh
  i=0
  for arg in "$@" ; do
    CHECK=`echo "$arg"|egrep -c "$OURCYGPATTERN" -`
    CHECK2=`echo "$arg"|egrep -c "^-"`                    ### Determine if an option


    if [ $CHECK -ne 0 ] && [ $CHECK2 -eq 0 ] ; then         ### Added a condition
eval `echo args$i`=`cygpath --path --ignore --mixed "$arg"`
    else
      eval `echo args$i`="\"$arg\""
    fi
    i=$((i+1))
  done
case $i in
(0) set -- ;;
```

```
(1) set -- "$args0" ;;
(2) set -- "$args0" "$args1" ;;
(3) set -- "$args0" "$args1" "$args2" ;;
(4) set -- "$args0" "$args1" "$args2" "$args3" ;;
(5) set -- "$args0" "$args1" "$args2" "$args3" "$args4" ;;
(6) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" ;;
(7) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" ;;
(8) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" "$args7" ;;
(9) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" "$args7" "$args8" ;;
  esac
fi


# Split up the JVM_OPTS And GRADLE_OPTS values into an array, following the shell quoting and
substitution rules function splitJvmOpts() {
  JVM_OPTS=("$@")
}
eval splitJvmOpts $DEFAULT_JVM_OPTS $JAVA_OPTS $GRADLE_OPTS
JVM_OPTS[${#JVM_OPTS[*]}]="-Dorg.gradle.appname=$APP_BASE_NAME"


exec "$JAVACMD" "${JVM_OPTS[@]}" -classpath "$CLASSPATH"
org.gradle.wrapper.GradleWrapperMain "$@"
```

Start.sh Bash script:

```
#!/bin/bash
# building the backend and deploying it sudo
sudo cp /home/ecse321/workspace/TreePLE/TreePLE/TreePLE-Spring/build/libs/eventregistration-
0.0.1-SNAPSHOT.war /opt/tomcat/webapps/ROOT.war
sudo systemctl stop tomcat sudo bash
```

```
/home/ecse321/catalina.sh stop sudo bash

/home/ecse321/catalina.sh start


# building android app and deploying it

cd /home/ecse321/workspace/TreePLE/TreePLE/TreePLE-Android sudo

./gradlew assembleRelease

sudo cp -rf /home/ecse321/workspace/TreePLE/TreePLE/TreePLE-Android/app/release/app-release.apk
/var/www/html/


# building front-end and deploying it sudo npm run build --prefix

/home/ecse321/workspace/TreePLE/TreePLE/TreePLE-Web/ sudo cp -rf

/home/ecse321/workspace/TreePLE/TreePLE/TreePLE-Web/dist/* /var/www/html/ sudo

chown apache:apache -R /var/www/

sudo systemctl restart httpd
```

Thank you for reading,
Sincerely,
Project 13