



GRAND CANYON UNIVERSITY™

CST-323 How to Guide

Contents

Part 1: Cloud Platform Deployment Notes for Microsoft Azure	2
Part 2: Cloud Platform Deployment Notes for Heroku	7
Part 3: Cloud Platform Deployment Notes for Amazon AWS	12
Part 4: Cloud Platform Deployment Notes for Google Cloud.....	19

Part 1: Cloud Platform Deployment Notes for Microsoft Azure

Note: You will need to create a GCU Azure Student account as documented in the Student Success Center using your GCU email credentials (this is free, most services will not need a credit card, and your card will NEVER be charged when using this activity as written). DO NOT create a standard Free Azure subscription as this WILL NOT work for this activity.

Important: Ensure you have a valid Azure for Students subscription by checking your subscription type and validate that you have the proper amounts of credits (\$100) allocated to your account. **If you DO NOT have an Azure for Students subscription, please see your instructor. DO NOT proceed with this activity if you do not have an Azure for Students subscription type.** Students should monitor their credit consumption weekly. If desired, to conserve credits, your application can always be stopped.

Deploy a Spring Boot App to Microsoft Azure

1. Log into the Azure Portal with your account credentials.
2. Create a MySQL database in Azure. From the Azure Search bar (the blue search bar at the top of the Azure Portal) search for MySQL. DO NOT search in the Azure Marketplace for MySQL. From the Click the 'Create Azure Database for MySQL server' button. Click the 'Create' button for the Flexible server option. Complete the following database configuration options:
 - a. Select the Azure for Students subscription.
 - b. Select a desired Resource Group or create a new one.
 - c. Enter a desired database name.
 - d. Leave the default region.
 - e. Select the desired version of MySQL (or simply leave the default).
 - f. Leave the Workload type as 'For development or hobby projects'.
 - g. Leave the Authentication method as 'MySQL authentication only'.
 - h. Specify a desired Admin username.
 - i. Specify a desired password (and confirm the password).
 - j. Click the 'Next: Networking' button.
 - k. Leave the Connectivity access to 'Public access'.
 - l. Under the Firewall rules click the 'Add 0.0.0.0 – 255.255.255.255' link. Click the Continue button to confirm.
 - m. Click the 'Review+create' button. Click the 'Create' button.
 - n. The database deployment will be started. Wait until the deployment has finished. This can take 5-10 minutes to complete.
 - o. Click the 'Go to resource button' to access your new database.
3. Initialize the MySQL Database. To access your database go to your Dashboard by selecting it from the top-level hamburger menu and clicking on the database under the resource list. Initialize your database by performing the following steps:
 - a. From the MySQL database menu select the Databases option under Settings. Click the +Add button to create a new database schema. Enter the name of your new database schema and click the 'Save' button. Note, this step maybe skipped if your DDL script creates your database schema for you.
 - b. Click on the Connect option under Settings. Note, your database hostname, username, and password that are listed under the Connection details section.

- c. Start MySQL Workbench and create a new connection using the database connection details from the previous step. Run your DDL script to initialize your database.
4. Configure your application. Complete the following steps to configure your application for deployment onto Azure:
 - a. Update the JDBC string in the application.properties file in your application with the MySQL Hostname, Port, and credentials from your Database Connection details (from the previous step) as shown below.

```
# On Azure
spring.datasource.url=jdbc:mysql://[DB_HOSTNAME]:3306/[DB_SCHEMA]
spring.datasource.username=[DB_USERNAME]
spring.datasource.password=[DB_PASSWORD]
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

- b. Set the Java version to 11 as shown below in the POM.xml as shown below.

```
<packaging>jar</packaging>

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

<dependencies>
```

NOTE: it is critically important that the version of Java 11 and even the minor version of Java be less than or equal to the version of Java 11 that is running in Azure. The version of Java can be determined in Azure or on your local system by running the `java -version` command from a command or terminal window.

5. Create a Web Application in Azure. Click the +Create a resource menu by selecting it from the top-level hamburger menu. Click the Web App from the list of services.
6. Validate your subscription type is correct, create a unique Resource Group Name, fill in a unique App name, select the Code publish option, select Java 11, select the Tomcat 8.5 Java web server stack, select the Windows option, and choose a desired Region. The Create Web App form should look like the following:

Create Web App ...

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Resource Group * ⓘ
[Create new](#)

Instance Details

Need a database? Try the [new Web + Database experience](#).

Name * ☒ .azurewebsites.net

Publish * ☒ Code ☐ Docker Container ☐ Static Web App

Runtime stack *

Java web server stack *

Operating System * ☐ Linux ☒ Windows

Region *
ⓘ Not finding your App Service Plan? Try a different region or select your App Service Environment.

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#)

Windows Plan (East US) * ⓘ
[Create new](#)

Pricing plan **Free F1** (Shared infrastructure)





[Review + create](#) [< Previous](#) [Next : Deployment >](#)

- Click the 'Review+create' button.
- When deployment is finished, pin it your Dashboard, and click the 'Go to resource' button.
- You can open your application from your Dashboard by clicking on it from your Dashboard. Select the Overview menu from the left pane and then click the URL from the right pane to open your application. Make sure the default Microsoft Azure Developer page is displayed.
- Deploy your application. Do a Maven build. Make sure to select the Skip Unit Tests option for your Maven build configuration. Rename the output jar to app.jar.
- Create a web.config file as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="httpPlatformHandler" path="*" verb="*" modules="httpPlatformHandler" resourceType="Unspecified"/>
    </handlers>
    <httpPlatform processPath="%JAVA_HOME%\bin\java.exe"
      arguments="-Djava.net.preferIPv4Stack=true -Dserver.port=%HTTP_PLATFORM_PORT% -jar "%HOME%\site\wwwroot\app.jar""/>
    </httpPlatform>
  </system.webServer>
</configuration>
```

- Go to Advanced Tools, click the Go link, under the Debug console menu select the CMD menu option, and navigate to site/wwwroot directory. Delete all existing content from this directory. Create a zip file containing the web.config and app.jar files (put these in the root directory of your zip file, meaning your zip file should have the 2 files and no folders within the zip file). Drag and drop the zip file to the right edge (like where the file size is shown) of the site/wwwroot page to expand and deploy the zip file contents. The site/wwwroot directory should look as follows.

... / wwwroot + | 2 items |   

	Name	Modified	Size
 	app.jar	11/20/2021, 9:27:44 AM	30868 KB
 	web.config	11/20/2021, 1:58:04 AM	1 KB

13. Go to Overview in azure and restart the app.
14. Click on the app link in the Azure Overview page (it may take a few minutes until app is restarted and running).

Deploy an Angular App to Microsoft Azure

1. Create a new Web App (if new application)
 1. Select the + Create a new Resource menu option.
 2. Select Web App, select to Publish Code, select Node 10.x Runtime stack either Windows (required for Zip deploy) or Linux, and select a desired Region. Click Review + Create. Click Create.
 3. After your new application deployment is finished click the application link to test out. Click go to Resource.
 4. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
2. Open the Web App from the Dashboard
3. Deploy from a Build:
 1. Select Advanced Tools and click the Go link
 2. Select the Debug console->CMD menu options.
 3. Navigate to the site/wwwroot directory.
 4. Delete the default content.
 5. Zip up all the files under within the dist\APP_NAME directory.
 6. Drag and drop the zip file onto the right side of the CMD window.
4. Deploy from a GIT CI/CD Build Pipeline:
 1. Select the Deployment Center menu option.
 2. Select the GitHub CI/CD type (authorize access if necessary) and click the Continue button.
 3. Select the GitHub Actions type and click the Continue button.
 - a. Fill in the GitHub Repository
 - b. Select the master branch
 - c. Select the NodeJS Runtime stack
 - d. Select the Node Version
 - e. Click the Finish button.
 - f. In the GitHub repo modify the GitHub build workflow file in the .github/workflows directory
 - i. Change the package: . entry to package: ./dist/[APP_NAME]
 - ii. Remove the steps for Unit Testing

Deploy a React App to Microsoft Azure

1. Create a new Web App (if new application)
 1. Select the + Create a new Resource menu option.
 2. Select Web App, select to Publish Code, select Node 10.x Runtime stack either Windows (required for Zip deploy) or Linux, and select a desired Region. Click Review + Create. Click Create.
 3. After your new application deployment is finished click the application link to test out. Click go to Resource.
 4. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
2. Open the Web App from the Dashboard
3. Deploy from a Build:
 1. Select Advanced Tools and click the Go link
 2. Select the Debug console->CMD menu options.
 3. Navigate to the site/wwwroot directory.
 4. Delete the default content.
 5. Zip up all the files within the build directory.
 6. Drag and drop the zip file onto the right side of the CMD window.
4. Deploy from a GIT CI/CD Build Pipeline:
 1. Select the Deployment Center menu option.
 2. Select the GitHub CI/CD type (authorize access if necessary) and click the Continue button.
 3. Select the GitHub Actions type and click the Continue button.
 - a. Fill in the GitHub Repository
 - b. Select the master branch
 - c. Select the Node.JS Runtime stack
 - d. Select the Node Version
 - e. Click the Finish button.
 - f. In the GitHub repo modify the GitHub build workflow file in the .github/workflows directory
 - g. Change the package: . entry to package: ./build
 - h. Remove the steps for Unit Testing

Part 2: Cloud Platform Deployment Notes for Heroku

Note: You will need to create a Heroku Student account as documented in the Student Success Center using your GCU email credentials (this is free for most services however you will need a credit card but your card will NEVER be charged when using this Activity as written). DO NOT create a standard Free Heroku subscription as this WILL NOT work for this Activity.

Important: Ensure you have a valid Heroku account by checking your account and validate that you have the proper amounts of credits (\$156) allocated to your account. **If you DO NOT have a proper Heroku account and credits, please see your instructor. DO NOT proceed with this Activity if you do not have credits in your Heroku account.** It should be noted that a small Spring Boot application and MySQL database will consume about \$6/month of credits. Students should monitor their credit consumption weekly. If desired to conserve credits your application can always be stopped (see instructions below).

Deploy a Spring Boot App to Heroku

1. Create a Heroku account. Please follow the Heroku setup instructions found in the Student Success Center.

2. Add a file named Procfile to the root of the project with the following entry:

```
web: java -Dserver.port=$PORT $JAVA_OPTS -jar target/*.jar
```

NOTE: although not fully tested this file may now be optional on Heroku for Spring Boot

3. If desired you can set the web application context to something other than root at / by adding the following to the application.properties file:

```
server.servlet.context-path=[CONTEXT_PATH]/
```

4. Do a manual Maven build to ensure that everything builds properly.
5. Test the Maven build by going to localhost:8080/ in your browser.
6. For Heroku if you are using a version greater than Java 8 in your test application you will need to update the Maven POM file to use Java 15. Finally, add a file named system.properties to the root of your project with the following setting. Make sure to test your Maven build and application locally.

```
java.runtime.version=15
```

7. Create a new project in Heroku and use GitHub deployment.
 - a. Click Create App button from the main page. Give your application a name. Click the Create App button.
 - b. On the Project page, select the Resources Tab, under the Add-ons search for JawsDB MySQL, select JawsDB MySQL from the search list, select the Free plan, click the Provision button.

NOTE: If you fail to connect too many times to the database Heroku may lock you out from connecting to your database. If you repeatedly cannot connect to your database and are sure your configuration is correct, then delete your current JawsDB MySQL database Add-on and add a new JawsDB MySQL database.

8. Create a connection using MySQL Workbench to the instance of MySQL Database by using the database configuration (host, username, password, and database/schema) from the JawsDB MySQL settings, which can be found by clicking on the Resources Tab and then by clicking on the JawsDB MySQL icon. Once connected to the JawsDB MySQL database, run your DDL Script from MySQL Workbench to initialize the database.

NOTE: DO NOT copy and paste the host, username, password, and database/schema directly from the JawsDB configuration web page into MySQL Workbench because you might copy tab or other hidden characters that are not visible. First copy and paste the host, username, password, and database/schema from the JawsDB configuration into Windows Notepad or Mac TextEdit (as a standard text file) or other safe text editor to strip any invalid characters from configuration settings.

9. On the Project page, select the Deploy Tab, and link your application to your GitHub repository (BitBucket is not supported on Heroku). If you are not using GitHub, you can copy your BitBucket repository to GitHub or use the Heroku CLI.
10. On the Project page, select the Settings Tab, click the Add Buildpack button, for Java click the Java button, click the Save Changes button.
11. Update the database configuration (see instructions in step 8) in the Spring Boot application.properties files.

NOTE: If you are encrypting database credential property values then you will need to set a secret named JASYPT_ENCRYPTOR_PASSWORD with your Jasper encryption secret key in a Heroku environment variable.

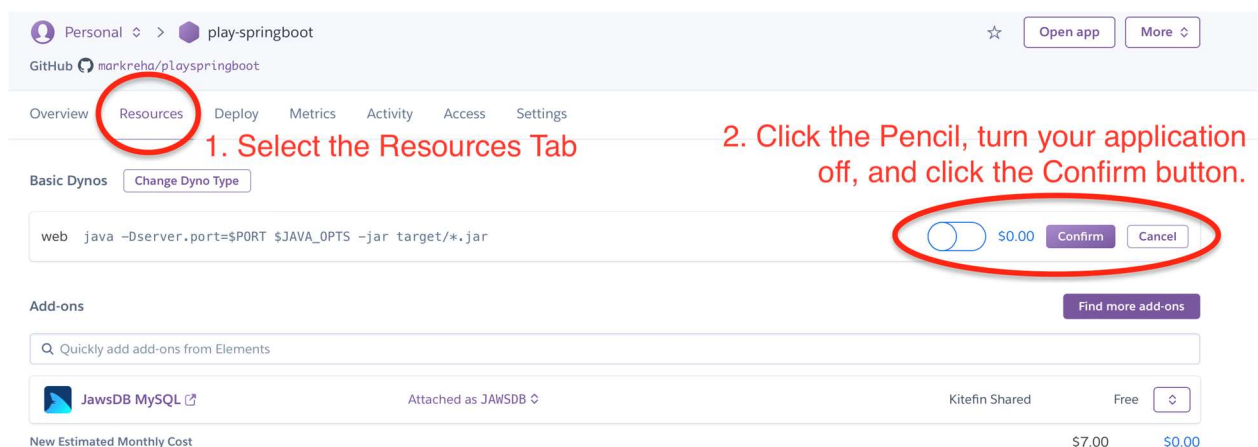
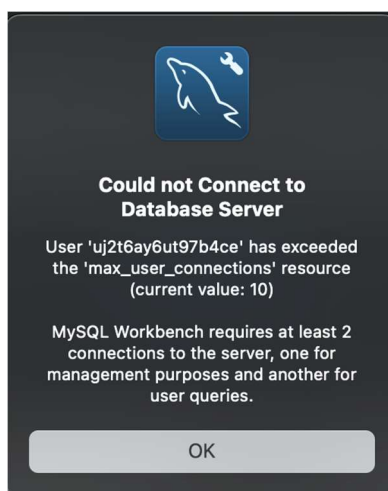
12. Click the Deploy tab. Click the Manual Deploy button to manually do a build. Click the Enable Automatic Deploys to enable the CI/CD build pipeline.
13. Click the Open app button to run the application.
14. If your app did not start, then use the Heroku CLI and run the following commands:

```
heroku login
```

```
heroku ps:scale web=1 -a [APP_NAME]
```

15. If your app started but you cannot read any data from your database validate that the case of your table names and column names match between what is displayed in MySQL Workbench and your database code.

16. NOTE: the JawsDB MySQL database service now restricts a max number of 10 connections to your database. It is possible once the Spring Boot application is deployed that there will be no available database connections. As shown in the figure below the following error might be displayed when accessing your database in MySQL Workbench. To work around this issue, you can have two options: 1) Stop your application, which will free up enough connections for you to access your database with MySQL Workbench. Then once you have finished any database updates using MySQL Workbench you can simply re-enable your application again. This is shown in the figure below. 2) Set the following property in the application.properties file to reduce the max number of connections in the database connection pool (NOTE: this has not been tested with Spring Boot 3.x): `spring.datasource.hikari.maximumPoolSize=5`



Deploy an Angular App to Heroku

1. Create a new Web App (if new application)
 - a. Select the New button and select the Create new app menu option. Give your app a Name. Click the Create App button.
 - b. Select the GitHub deployment method and connection to your GitHub repository.
 - c. After your new application deployment is finished click the application link to test out. Click go to Resource.
 - d. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
2. Open the Web App from the Dashboard
3. Configure the application:
 - a. Update package.json to include the express library in the list of dependencies specifying the version of express used in development:
 - i. "express": "^4.17.1"
 - b. Update package.json to include a Heroku post install step to build the project in the list of scripts (this step is only required for CI/CD builds):
 - i. "heroku-postbuild": "ng build --base-href ."
 - c. Add a new file named Procfile to the repository with the following entry:
 - i. web: node server.js
 - d. Add a new file named server.js to the repository that will be used to serve up the React application:
 - e. Set the following code to initialize the Express application (and specify an APP_NAME):
 - i. app.use(express.static(__dirname));
 - ii. app.use(express.static(__dirname + '/dist/[APP_NAME]'));
 - f. The /route should contain the following code (and specify an APP_NAME):
 - i. res.sendFile(path.join(__dirname + '/dist/[APP_NAME]/index.html'));

```
1  const express = require('express');
2  const path = require('path');
3  const port = process.env.PORT || 8080;
4  const app = express();
5
6  app.use(express.static(__dirname));
7  app.use(express.static(__dirname + '/dist/playangular'))
8
9  app.get('/ping', function (req, res)
10 {
11   return res.send('pong');
12 });
13
14 app.get('/*', function (req, res)
15 {
16   res.sendFile(path.join(__dirname + '/dist/playangular/index.html'));
17 });
18
19 app.listen(port, function ()
20 {
21   console.info('Angular Server App listening on port ' + port);
22 });
```
4. Deploy from a Build:
 - a. Run a build using the ng build --base-href . command.

- b. Push all the code including the dist directory to the repository.
 - c. Select the Deploy menu from Heroku. Click the Deploy Branch button.
- 5. Deploy from a GIT CI/CD Build Pipeline:
 - a. Push all the code excluding the dist directory to the repository.
 - b. Select the Deploy menu from Heroku. Click the Enable Automatic Deploy button. Click the Deploy Branch button.

Deploy a React App to Heroku

1. Create a new Web App (if new application)
 1. Select the New button and select the Create new app menu option. Give your app a Name. Click the Create App button.
 2. Select the GitHub deployment method and connection to your GitHub repository.
 3. After your new application deployment is finished click the application link to test out. Click go to Resource.
 4. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
2. Open the Web App from the Dashboard
3. Configure the application:
 1. Update package.json to include the express library in the list of dependencies specifying the version of express used in development:
 - a. `"express": "^4.17.1"`
 2. Update package.json to include a Heroku post install step to build the project in the list of scripts (this step is only required for CI/CD builds):
 - b. `"heroku-postbuild": "react-scripts build"`
 3. Add a new file named Procfile to the repository with the following entry:
 - c. `web: node server.js`
 4. Add a new file named server.js to the repository that will be used to serve up the React application:
 - d. Set the following code to initialize the Express application:
 - i. `app.use(express.static(__dirname));`
 - ii. `app.use(express.static(path.join(__dirname, 'build'));`
 - e. The /route should contain the following code:
 - i. `res.sendFile(path.join(__dirname, 'build', 'index.html'));`

```

1  const express = require('express');
2  const path = require('path');
3  const port = process.env.PORT || 8080;
4  const app = express();
5
6  app.use(express.static(__dirname));
7  app.use(express.static(path.join(__dirname, 'build')));
8
9  app.get('/ping', function (req, res)
10 {
11     return res.send('pong');
12 });
13
14 app.get('/*', function (req, res)
15 {
16     res.sendFile(path.join(__dirname, 'build', 'index.html'));
17 });
18
19 app.listen(port, function ()
20 {
21     console.info('React Server App listening on port ' + port);
22 });

```

5. Deploy from a Build:
 - a. Run a build using the npm run build command.
 - b. Push all the code including the build directory to the repository.
 - c. Select the Deploy menu from Heroku. Click the Deploy Branch button.
6. Deploy from a GIT CI/CD Build Pipeline:
 - a. Push all the code excluding the build directory to the repository.
 - b. Select the Deploy menu from Heroku. Click the Enable Automatic Deploy button. Click the Deploy Branch button.

Part 3: Cloud Platform Deployment Notes for Amazon AWS

Note: You will need to create an AWS account (this is free 12-month service for most services and you will get **750 hours per month**; you will need a credit card and your card will NEVER be charged when using this Activity as written, which assumes you will **ONLY deploy a single** application and database to AWS).

Deploy a Spring Boot App to Amazon AWS:

1. Go to All Services and under Compute click on the Elastic Beanstalk option or type elastic beanstalk in the search bar. Once you see Elastic Beanstalk click on that option.
2. Select Applications from the left pane. Click on the 'Create a new Application' button. Complete the following steps:
 - Give the Application a name and click the Create button.
 - Click on the 'Create a new environment' button.
 - Select the 'Web server environment' option. Click the Select button.
 - Under the 'Platform' section select the Java platform option and the Coretto 11 option.
 - Under the 'Application code' section Select the 'Sample application' option.

- Click the 'Configure more options' button.
- Under the 'Database' section click the Edit button.
- Select the mysql, the version of MySQL that you have built your application with, and the db.t2.micro options. Set the username and password for MySQL. Set the Database deletion policy to Delete (IMPORTANT to ensure your data is not archived when your database is deleted and to avoid storage charges) Click the Save button.
- Click the 'Create environment' button.
- It will take quite a few minutes before your application is ready. From the left pane click the Environments link and wait until the Health of your environments shows as OK.
- From the left pane under your environment name click the Configuration link. Under the Database configuration click the link to URL of your database. This will open the RDS service. Click on your DB under the DB identifier column.
 1. **Important:** Under the 'Connectivity & security' section make sure the Public Accessibility setting is set to Yes.
 2. **Important:** Under the 'Security group rules' section click on the 'EC2 Security Group - Inbound' security group. Under the 'Inbound rules' tab click the 'Edit inbound rules' button. Click the 'Add rule' button. Add \a rule for 'All TCP with a source type of 'Anywhere-IPv4' set to 0.0.0.0/0, and click the 'Save Rule' button.

Inbound rules (2)								Manage tags	Edit inbound rules
Filter security group rules								< 1 >	⚙
<input type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Source		
<input type="checkbox"/>	–	sgr-07a5621f8bfac3764	IPv6	MYSQL/Aurora	TCP	3306	::/0		
<input type="checkbox"/>	–	sgr-06d62e03c698334...	IPv4	MYSQL/Aurora	TCP	3306	0.0.0.0/0		

Security group	Type	Rule
rds-launch-wizard (sg-091e78a2a7ca6b2b0)	CIDR/IP - Inbound	0.0.0.0/0
rds-launch-wizard (sg-091e78a2a7ca6b2b0)	CIDR/IP - Outbound	0.0.0.0/0

3. **Important:** Note the database Endpoint and Port under Connectivity" & security' section as this will be your database hostname and port for connecting to your database.
4. Create a connection using MySQL Workbench to the instance of MySQL Database by using the database configuration (endpoint/hostname, username, password, and database/schema) from the AWS RDS MySQL settings. Once connected to the AWS RDS MySQL database, run your DDL Script from MySQL Workbench to initialize the database.
5. Update the database configuration (see instructions in previous step) in the Spring Boot application.properties files.

```
# On AWS
spring.datasource.url=jdbc:mysql://[YOUR DB ENDPOINT]:3306/[YOUR DB SCHEMA]
spring.datasource.username=[YOUR DB USERNAME]
spring.datasource.password=[YOUR DB PASSWORD]
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

- Configure and do a Maven build using the version of Java 11 for a JAR application.
Note: Elastic BeanStalk's load balancer uses path "/" by default for health checks. If you don't have that path defined in your controller, your application will keep failing health checks and will show Severe as status in dashboard. You can either have "/" endpoint in your controller or edit the load balancer setting later to use different path instead.
- From the left pane under your environment name click the Configuration link. Under the Software section click the Edit button. Under the 'Environment properties' add the following variable:
SERVER_PORT with a value of 5000. Click the Apply button. Wait for the changes to be deployed.
- From the left pane click the Environments link. Click the link for your application environment. Click the 'Upload and deploy' button. Upload the application JAR file. Click the Deploy button. Wait for the changes to be deployed.
- From the left pane click the click the Environments link. Click the application URL to access your application.
Note: if your database queries do not work properly make sure the case of your Table and Column names in your code match the case observed in the Tables and Columns in MySQL Workbench when connected to the AWS RDS database.

Deploy an Angular App to Amazon AWS

1. Log into AWS and select Elastic Beanstalk from the Compute Services.
2. Create a new Web App (if new application)
 - a. Click the Create a new application button.
 - b. Give your application a name. Click the Create button.
 - c. Click the Create a new environment button.
 - i. Web server environment
 - ii. Platform of Node.js
 - iii. Platform branch of Node.js 10 running on 64bit Linux
 - iv. Sample application
 - v. Click the Create button
 - vi. Once the environment is up and running test the Sample application to ensure all is working properly
 - vii. Add new database and initialize with a DDL for your application if needed (make sure to select a database from the free tier)
3. Configure the application:

- a. Update package.json to include the express library in the list of dependencies specifying the version of express used in development:
 - i. "express": "^4.17.1"
- b. Update package.json to include a Heroku post install step to build the project in the list of scripts (this step is only required for CI/CD builds):
 - i. "heroku-postbuild": "ng build --base-href ."
- c. Add a new file named Procfile to the repository with the following entry:
 - i. web: node server.js
- d. Add a new file named server.js to the repository that will be used to serve up the React application:
 - i. Set the following code to initialize the Express application (and specify an APP_NAME):
 1. app.use(express.static(__dirname));
 2. app.use(express.static(__dirname + '/dist/[APP_NAME]'));
 - ii. The /route should contain the following code (and specify an APP_NAME):
 1. res.sendFile(path.join(__dirname + '/dist/[APP_NAME]/index.html'));

```

1  const express = require('express');
2  const path = require('path');
3  const port = process.env.PORT || 8080;
4  const app = express();
5
6  app.use(express.static(__dirname));
7  app.use(express.static(__dirname + '/dist/playangular'))
8
9  app.get('/ping', function (req, res)
10 {
11   return res.send('pong');
12 });
13
14 app.get('/*', function (req, res)
15 {
16   res.sendFile(path.join(__dirname + '/dist/playangular/index.html'));
17 });
18
19 app.listen(port, function ()
20 {
21   console.info('Angular Server App listening on port ' + port);
22 });

```

4. Deploy from a Build:

- a. Run a build using the ng build --base-href . command.
- b. Zip up all the code and files within the dist directory but do not include the e2e, node_modules, and src directories.
- c. Click on the Upload and deploy button.
- d. Click the Choose file button and select your zip file.
- e. Click the Deploy button.

5. Deploy from a GIT CI/CD Build Pipeline:

- a. Configure code and setup build pipeline (if not already completed):
 - i. Add a buildspect.yml to the root of your application code for Node 10 application.

Deploy a React App to Amazon AWS

1. Log into AWS and select Elastic Beanstalk from the Compute Services.
2. Create a new Web App (if new application)
 - a. Click the Create a new application button.
 - b. Give your application a name. Click the Create button.
 - c. Click the Create a new environment button.
 - i. Web server environment
 - ii. Platform of Node.js
 - iii. Platform branch of Node.js 10 running on 64bit Linux
 - iv. Sample application
 - v. Click the Create button
 - vi. Once the environment is up and running test the Sample application to ensure all is working properly
 - vii. Add new database and initialize with a DDL for your application if needed (make sure to select a database from the free tier)
3. Configure the application:
 - a. Update package.json to include the express library in the list of dependencies specifying the version of express used in development:
 - i. "express": "^4.17.1"
 - b. Update package.json to include a Heroku post install step to build the project in the list of scripts (this step is only required for CI/CD builds):
 - i. "heroku-postbuild": "react-scripts build"
 - c. Add a new file named Procfile to the repository with the following entry:
 - i. web: node server.js
 - d. Add a new file named server.js to the repository that will be used to serve up the React application:
 - i. Set the following code to initialize the Express application:
 1. app.use(express.static(__dirname));
 2. app.use(express.static(path.join(__dirname, 'build'));
 - ii. The /route should contain the following code:
 1. res.sendFile(path.join(__dirname, 'build', 'index.html'));


```

1  const express = require('express');
2  const path = require('path');
3  const port = process.env.PORT || 8080;
4  const app = express();
5
6  app.use(express.static(__dirname));
7  app.use(express.static(path.join(__dirname, 'build')));
8
9  app.get('/ping', function (req, res)
10 {
11     return res.send('pong');
12 });
13
14 app.get('/*', function (req, res)
15 {
16     res.sendFile(path.join(__dirname, 'build', 'index.html'));
17 });
18
19 app.listen(port, function ()
20 {
21     console.info('React Server App listening on port ' + port);
22 });

```

4. Deploy from a Build:
 - a. Run a build using the npm run build command.
 - b. Zip up all the code and files within the build directory but do not include the node_modules and src directories.
 - c. Click on the Upload and deploy button.
 - d. Click the Choose file button and select your zip file.
 - e. Click the Deploy button.
5. Deploy from a GIT CI/CD Build Pipeline:
 - a. Configure code and setup build pipeline (if not already completed):
 - b. Add a buildspect.yml to the root of your application code for Node 10 application.

```

1  version: 0.2
2
3  # Build the code
4  phases:
5    install:
6      runtime-versions:
7        nodejs: 10
8    pre_build:
9      commands:
10       - echo Installing source NPM dependencies...
11       - npm install
12    build:
13      commands:
14       - echo Build started on `date`
15       - npm run build
16    post_build:
17      commands:
18       - echo Build completed on `date`
19
20  # Include only the files required for your application to run.
21  artifacts:
22    files:
23     - server.js
24     - Procfile
25     - package.json
26     - build/**/*
27     - public/**/*

```

- c. Log into AWS and select Services from the main menu.
- d. Select the CodePipeline service.
- e. Click the Create Pipeline button.
- f. Give your pipeline a name (i.e. TestAppPipeline). Click the Next step button.
- g. Select GitHub from the Source provider dropdown. Click the Connect to GitHub button and select your repository and master branch. Click the Next step button.
- h. Select AWS CodeBuild from the Build provider dropdown. Select the Create a new build project option. Give your build a name. Select Linux operating system with defaults and using a buildspec.
- i. Click the Create Project button.
- j. Click the Next step button.
- k. Select AWS Elastic Beanstalk from the Deployment provider dropdown. Choose your Application and Environment from the dropdowns. Click the Next step button.
- l. Click the Create pipeline button.
- m. To build and deploy your application:
 - i. Select the CodePipeline service from the Services dashboard. Open the Pipeline.
 - ii. Either make a change to code in GitHub or click the Release change button to start a build and deployment.

Part 4: Cloud Platform Deployment Notes for Google Cloud

Note: You will need to create a Google Cloud Platform account (this requires a credit card and gives you \$300 credit to explore their cloud platform over a 90-day period. Your card will NEVER be charged when using this Activity as written. You should delete your Google Cloud Platform account once you are done with this Activity, and you are not using this cloud platform for your CLC Milestone Assignment.

Deploy a Spring Boot App to Google Cloud

1. Create an account on Google Cloud.
2. Create a new App Engine project and application using the following steps:
 - a. Select App Engine from the Main Menu.
 - b. Click the 'Select a Project' dropdown list and then click the New Project icon.
 - c. Give your Project a Name and click the Create button.
 - d. From the Welcome to App Engine screen click the Create Application button.
 - e. Select a Region from the US and click the Next button.
3. Create a new MySQL Database using the following steps:
 - a. Select SQL menu item from the Main Menu. Click the Create Instance button.
 - b. Click the Choose MySQL button. If prompted click the Enable API button.
 - c. Fill out the Instance ID (database name), root password, desired version of MySQL that matches your development version, region, and single zone options.
 - d. Expand the Show Configuration Options. Select the Machine Type of Shared Core with 1vCPU and the smallest database available. Make sure Public IP is enabled. Click the Create Instance button. NOTE: it is extremely important that these options are set to avoid being charged by Google for your database usage.
 - e. The database instance can take quite a few minutes to complete.
 - f. Note your Public IP Address.
 - g. From the left pane select the Users menu, click the Add User Account button, and then create a new user [DB_USERNAME]/[DB_PASSWORD] with the Allow any host option. Click the Add button.
 - h. Select the Database menu, enter your new Database (your schema) name and click the Create button.
 - i. Get your public IP Address by going to your browser and in the search bar enter 'My IP'. Note your IPv4 Address for the next step.
 - j. Select the Connections menu and under Authorization Networks click Add Network button, name of GCU, network of your IP Address (from previous step), click Done and Save buttons.
 - k. The database updates can take quite a few minutes to complete.

- l. Set up a MySQL Workbench connection using the databases IP address (listed in the Overview menu) and your database credentials (setup from the prior step).
 - m. Connect to the database in MySQL Workbench and run your DDL script.
 - n. From the main Google menu go to APIs & Services, click on the Library menu, search for Google Cloud SQL, click on each one, and make sure both the Cloud SQL and sqladmin API are enabled.
4. Configure, build, test, and deploy the Spring Boot test application using the following steps:
- a. Open up a Cloud Shell from the Activate Cloud Shell icon in the top menu. From the Cloud Shell, create a working directory, change into that working directory, and perform the following operations.

Note: once you have a Cloud Shell open if you click on the Pencil icon from the Cloud Shell menu this will open a tree view of your code, which allows you to edit some of your configuration files. Once you are in the editor you can also upload files into your project.

- b. Run the following command from the Cloud Shell:
git clone [URL to your Test App Repo]

Note: you will need to make your GIT repository public in order for the GIT clone command to work.

- c. Click the Open Editor button and make the following changes to the application:
 - Update the POM file with the following changes:

- Set Maven to build using Java 11:

```
<properties>
|   <java.version>11</java.version>
</properties>
```

- Add the following Maven dependencies (should be last in your dependencies):

```
<!-- For Google App Engine -->
<dependency>
  <groupId>com.google.cloud.sql</groupId>
  <artifactId>mysql-socket-factory</artifactId>
  <version>1.0.5</version>
</dependency>
<dependency>
  <groupId>com.google.api-client</groupId>
  <artifactId>google-api-client</artifactId>
  <version>1.23.0</version>
</dependency>
<dependency>
  <groupId>com.google.api-client</groupId>
  <artifactId>google-api-client-appengine</artifactId>
  <version>1.21.0</version>
</dependency>
<dependency>
  <groupId>jakarta.xml.bind</groupId>
  <artifactId>jakarta.xml.bind-api</artifactId>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.3.3</version>
  <scope>runtime</scope>
</dependency>
```

- Add the following Maven plugin:

```
<!-- For Google App Engine -->
<plugin>
  <groupId>com.google.cloud.tools</groupId>
  <artifactId>appengine-maven-plugin</artifactId>
  <version>2.2.0</version>
  <configuration>
    <version>1</version>
    <projectId>GCLLOUD_CONFIG</projectId>
  </configuration>
</plugin>
```

• Create a new directory named appengine under src/main. Create new file in src/main/appengine named app.yaml with the following entries:

```
runtime: java11
handlers:
- url: /*
  script: this field is required, but ignored
manual_scaling:
  instances: 1
```

• Update the application.properties file with the MySQL connection configuration (JDBC Connection string, database username, and database password). The JDBC Connection string is formatted as follows:

```
spring.datasource.url=jdbc:mysql://google/[DB_SCHEMA]?socketFactory=com.google.cloud.sql.mysql.SocketFactory&cloudSqlInstance=[PROJECT_ID]:[DB_REGION]:[DB_INSTANCE_NAME]
```

```
# On Google
spring.datasource.url=jdbc:mysql://google/[DB_SCHEMA]?socketFactory=com.google.cloud.sql.mysql.SocketFactory&cloudSqlInstance=[PROJECT_ID]:[DB_REGION]:[DB_INSTANCE_NAME]
spring.datasource.username=[DB_USERNAME]
spring.datasource.password=[DB_PASSWORD]
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

5. Test locally in Cloud Shell by navigating to the root of your project and running the following commands:

- mvn clean package
- cd target
- java -jar [JAR_NAME]
- Make sure there are no errors at startup
- Test by clicking on the Web Preview icon in the Shell and use port 8080 option

Note: if your database queries do not work properly make sure the case of your Table and Column names in your code match the case observed in the Tables and Columns in MySQL Workbench when connected to the Google Cloud RDS database.

6. Deploy in Cloud Shell by navigating to the root of your project by running the following command:

- mvn clean package appengine:deploy

Note: This will take a few minutes to complete. If this command fails and in order for this to work properly you may need to wait for Google to finish provisioning storage to upload your build artifact. If the deployment fails, try running the deploy command multiple times.

7. Test by going to the target URL output in the console window.

Deploy an Angular App to Google Cloud

1. Log into Google Cloud.
2. Create a new Web App (if new application):
 - a. Click on the list of projects in the top menu bar. Click the New Project button.
 - b. Give your project a name, click the Create button, then select a US region and NodeJS for framework.
 - c. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
3. Configure the application:
 - a. Add a Google Cloud app.yaml file to the root of your repository.
 - b. Set runtime to nodejs and the environment to flex and the desired minimum manual scaling settings (see the Google Cloud web site for examples).

```
1 runtime: nodejs
2 env: flex
3
4 manual_scaling:
5   instances: 1
6 resources:
7   cpu: 0.25
8   memory_gb: 0.50
9   disk_size_gb: 10.00
```
4. Deploy from a Build:
 - a. Activate the Cloud Shell for your project:
 - b. Create a working directory or navigate to an existing working directory.
 - i. **Note:** these steps assume the use of the Google Cloud Shell but an alternative is to download and install the Google Cloud SDK (at <https://cloud.google.com/sdk>).
 - c. Clone the GIT Repository using the 'git clone [GIT REPO URL]' command. Navigate to the root of your project code.
 - d. Run the 'npm install' command.
 - e. Run the 'npm run build --prod' command (if you make any changes run a 'git pull [GIT REPO URL]' command and run the 'npm run build --prod' command again).
 - f. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts, replace APP_NAME accordingly:
 - g. Update: "start": "serve -s dist/[APP NAME]",
 - h. Add: "prestart": "npm install -g serve",
 - i. Run the 'gcloud app deploy --project [PROJECT ID]' command.
 - j. Access the application from the URL noted in the build screen or select the App Engine > Versions menu options from the main Google Cloud page.
5. Deploy from a GIT CI/CD Build Pipeline:
 - a. Select Cloud Build from the Google Cloud main menu.
 - b. Enable the App Engine Admin API by going to the API's & Services-> Library menu options, search for App Engine Admin API, and enable the API.
 - c. Click the Settings menu. Enable App Engine Service account permissions.

- d. Select the Triggers menu. Click the Connect repository button. Select GitHub option. Click the Continue button. Click the Install Google Cloud Build button to enable access to desired repositories for specified projects.
- e. Create a Google Cloud Build file named cloudbuild.yaml and add this to the root of the project in GitHub.

```

1  steps:
2
3      # Install node packages
4      - name: 'gcr.io/cloud-builders/npm'
5        args: [ 'install' ]
6
7      # Build productive files
8      - name: 'gcr.io/cloud-builders/npm'
9        args: [ 'run', 'build', '--prod' ]
10
11     # Deploy to google cloud app engine
12     - name: 'gcr.io/cloud-builders/gcloud'
13       args: ['app', 'deploy', '--version=prod']

```

- f. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts:
 - g. Update: "start": "serve -s build",
 - h. Add: "prestart": "npm install -g serve",
 - i. To build and deploy your application:
 - j. Select the Cloud Build > Triggers menu open options.
 - i. Either make a change to code in GitHub and select the Cloud Build > Dashboard menu open options or select the Cloud Build > Triggers menu and click the Run Trigger button to start a build and deployment.

Deploy a React App to Google Cloud

1. Log into Google Cloud.
2. Create a new Web App (if new application):
3. Click on the list of projects in the top menu bar. Click the New Project button.
 - a. Give your project a name, click the Create button, then select a US region and NodeJS for framework.
 - b. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
 - c. Configure the application:
 - d. Add a Google Cloud app.yaml file to the root of your repository.
 - e. Set runtime to nodejs and the environment to flex and the desired minimum manual scaling settings (see the Google Cloud web site for examples).

```

1 runtime: nodejs
2 env: flex
3
4 manual_scaling:
5   instances: 1
6 resources:
7   cpu: 0.25
8   memory_gb: 0.50
9   disk_size_gb: 10.00

```

4. Deploy from a Build:

- a. Activate the Cloud Shell for your project:
- b. Create a working directory or navigate to an existing working directory.
 - i. NOTE: these steps assume the use of the Google Cloud Shell but an alternative is to download and install the Google Cloud SDK (at <https://cloud.google.com/sdk>).
- c. Clone the GIT Repository using the 'git clone [GIT REPO URL]' command. Navigate to the root of your project code.
- d. Run the 'npm install' command.
- e. Run the 'npm run build' command (if you make any changes run a 'git pull [GIT REPO URL]' command and run the 'npm run build' command again).
- f. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts:
- g. Update: "start": "serve -s build",
- h. Add: "prestart": "npm install -g serve",
- i. Run the 'gcloud app deploy --project [PROJECT ID]' command.
- j. Access the application from the URL noted in the build screen or select the App Engine > Versions menu options from the main Google Cloud page.

5. Deploy from a GIT CI/CD Build Pipeline:

- a. Select Cloud Build from the Google Cloud main menu.
- b. Enable the App Engine Admin API by going to the APIs & Services > Library menu options, search for App Engine Admin API, and enable the API.
- c. Click the Settings menu. Enable App Engine Service account permissions.
- d. Select the Triggers menu. Click the Connect repository button. Select GitHub option. Click the Continue button. Click the Install Google Cloud Build button to enable access to desired repositories for specified projects.
- e. Create a Google Cloud Build file named cloudbuild.yaml and add this to the root of the project in GitHub.

```

1 steps:
2
3   # Install node packages
4   - name: 'gcr.io/cloud-builders/npm'
5     args: [ 'install' ]
6
7   # Build productive files
8   - name: 'gcr.io/cloud-builders/npm'
9     args: [ 'run', 'build', '--prod' ]
10
11  # Deploy to google cloud app engine
12  - name: 'gcr.io/cloud-builders/gcloud'
13    args: ['app', 'deploy', '--version=prod']

```


- f. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts:
 - i. Update: "start": "serve -s build",
 - ii. Add: "prestart": "npm install -g serve",
- g. To build and deploy your application:
 - i. Select the Cloud Build > Triggers menu open options.
 - ii. Either make a change to code in GitHub and select the Cloud Build > Dashboard menu open options or select the Cloud Build > Triggers menu and click the Run Trigger button to start a build and deployment.