**CST-323 How to Guide**

Contents

[Part 1: Cloud Platform Deployment Notes for Microsoft Azure 2](#_Toc40101956)

[Part 2: Cloud Platform Deployment Notes for Heroku 8](#_Toc40101957)

[Part 3: Cloud Platform Deployment Notes for Red Hat OpenShift 14](#_Toc40101958)

[Part 4: Cloud Platform Deployment Notes for Amazon AWS 19](#_Toc40101959)

[Part 5: Cloud Platform Deployment Notes for Google Cloud 28](#_Toc40101960)

# Part 1: Cloud Platform Deployment Notes for Microsoft Azure

**Deploy a Spring Framework App or Spring Boot to Microsoft Azure**

For the Spring Framework, create Tomcat 8.5 and MySQL Container in Azure, and for Spring Boot, create a Java 8 SE Runtime for Windows (so you get MySQL):

1. Ensure you have a valid Azure for Students Starter account by checking your subscription type. If you do not have an Azure for Students Starter account, please see your instructor.

**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; for most services you will not need a credit card, and your card will NEVER be charged when using this activity as written).

1. Log into the Azure Portal.
2. Click the '+ Create a resource' icon from the Azure Portal.
3. From the 'Search the Marketplace' search box select Web App + MySQL option. Click the Create button, give your App a name, make sure the Microsoft Azure for Students subscription is selected, and click the Create button. After a few minutes, your application will be created. It is advised that you pin this application to your Dashboard.

**NOTE:** With the Azure for Students Starter account, you can only provision 2 containers in the same App Service Plan (i.e., within the same Region). If you provision more than 2 containers, Microsoft will shut down all your applications. You will then either have to delete some of your applications or simply use a different region when you create your application.

1. Open your application by clicking on it from your Dashboard. Select the Overview menu from the left pane, then click the URL from the right pane to open your application. Make sure the default Microsoft Azure Developer page is displayed. This is an important step! This will ensure that phpMyAdmin is accessible via single sign on. phpMyAdmin DOES NOT require any credentials to log in. If you get prompted for credentials, access your application via its URL and/or simply wait and try again.
2. Click on the Configuration menu from the left pane. Click the General Settings tab. Select the Java stack, select Java 8, and Tomcat from the Java Container options.

Initialize the MySQL Database:

1. Open your application.
2. Under the Settings section click the MySQL In App icon, make sure your Database is enabled, and click the Manage icon to open phpMyAdmin. Import your Database DDL.
3. Under the Development Tools section, click the Console icon.
4. Navigate to the D:\home\data\mysql directory and display the ‘MYSQLCONNSTR\_localdb.txt' file using the type command to get your MySQL Connection Properties. Note the DB connection information to get your DB hostname, post, and credentials.

Possible Configuration for Application:

1. If you have a large application (especially a Spring MVC Framework application), you will need to increase the JVM heap size (which defaults to a very small 16M startup heap size). If your application runs out of memory or does not start due to running out of memory, you will need to complete the following steps.
2. Under the Development Tools section, click the Advanced Tools icon. Click on the Go hyperlink to open the Tool. Click on the CMD menu option under the Debug Console menu. Navigate to the site->wwwroot directory.
3. Create a file named web.config in the wwwroot directory.
4. Add the following content to the web.config file (you might need to change the version of Tomcat in the httpPlatform tag to match the version of Tomcat setup in your Azure Configuration options):

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

<configuration>

<system.webServer>

<handlers>

<remove name="httpPlatformHandlerMain" />

<add name="httpPlatformHandlerMain" path="\*" verb="\*" modules="httpPlatformHandler" resourceType="Unspecified"/>

</handlers>

<httpPlatform processPath="D:\Program Files (x86)\apache-tomcat-8.5.34\bin\startup.bat" requestTimeout="00:04:00" arguments="start" startupTimeLimit="60" startupRetryCount="3" >

<environmentVariables>

<environmentVariable name="CATALINA\_OPTS" value="-Xms256m -Xmx256m -Dport.http=%HTTP\_PLATFORM\_PORT%" />

</environmentVariables>

</httpPlatform>

</system.webServer>

</configuration>

1. Stop your application. Then start your application (do not restart, but stop and start) for the configuration settings to take place.

Build and Deploy your Application:

1. Open your application.
2. Update your MySQL Database Connection properties in your application (note your hostname will need to be formatted as hostname:port).
3. In Eclipse, run a Maven build script, or right click on your project and use the Export->WAR file menu options to create a WAR file for your project if you are using the Spring Framework. Rename the WAR file with a ZIP extension. For Spring Boot, run a Maven build script to build a JAR file. Rename the JAR file to app.jar.
4. Under the Development Tools section, click the Advanced Tools icon. Click on the Go hyperlink to open the Tool.
5. Click on the CMD menu option under the Debug Console menu. Navigate to the site->wwwroot->webapps->ROOT directory. Drag and drop your application ZIP file onto the right edge of the web page (you will see a small zipper icon when your zip file is being extracted into your ROOT directory). DO NOT drag and drop your zip file to the middle of the page as this will simply copy the file to your ROOT directory. After a few minutes, your application should deployed and accessible with your Azure application URL.

**NOTE:** One option for deployment, rather than using zip files and the Azure Zip Deploy features, is to use an Eclipse Plug-in. The following extension can be installed in Eclipse and used to deploy a Spring application on to Azure:

http://marketplace.eclipse.org/content/azure-toolkit-eclipse

**Deploy a PHP App to Microsoft Azure**

Create PHP Container in Azure:

1. Ensure you have a valid Azure for Students Starter account by checking your subscription type. If you do not have an Azure for Students Starter account, please see your instructor.

**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; for most services you will not need a credit card, and your card will NEVER be charged when using this activity as written).

1. Log into the Azure Portal.
2. Click the '+ Create a resource' icon from the Azure Portal.
3. From the 'Search the Marketplace' search box select Web App + MySQL option. Click the Create button, give your App a name, make sure the Microsoft Azure for Students subscription is selected, and click the Create button. After a few minutes, your application will be created. It is advised that you pin this application to your Dashboard.

**NOTE:** With the Azure for Students Starter account, you can only provision 2 containers in the same App Service Plan (i.e., within the same Region). If you provision more than 2 containers, Microsoft will shut down all your applications. You will then either have to delete some of your applications or simply use a different region when you create your application.

1. Open your application 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. This is an important step! This will ensure that phpMyAdmin is accessible via single sign on. phpMyAdmin DOES NOT require any credentials to log in. If you get prompted for credentials, access your application via its URL and/or simply wait and try again.
2. Click on the Configuration menu from the left pane. Click the General Settings tab. Select the PHP stack and the version of PHP being used in your application development. It should be noted that the version of PHP you select needs to be compliant with the version of Laravel you are using. For example, if you are using Laravel 5.5.28 then your safest know working version of PHP should be 7.1

Initialize the MySQL Database:

1. Open your application.
2. Under the Settings section, click the MySQL In App icon, make sure your Database is enabled, and click the Manage icon to open phpMyAdmin. Import your Database DDL.
3. Under the Development Tools section click the Console icon.
4. Navigate to the D:\home\data\mysql directory and display the ‘MYSQLCONNSTR\_localdb.txt' file using the type command to get your MySQL Connection Properties. Note the DB connection information to get your DB hostname, post, and credentials.

Build and Deploy your Application:

1. Make sure you have built your PHP or PHP Laravel Project with the right version to match the version of PHP you are using.
2. Update your MySQL Database Connection properties in your application (note your hostname will need to be formatted as hostname:port).
3. For the local PHP Laravel codebase, make sure to copy the web.config from the public directory to the root of your project.
4. For the local PHP Laravel codebase, zip up your PHP project into a file named [appname].zip.

**NOTE:** Make sure to zip up all of the hidden files and ensure the .env file is included as this is a required file to run a Laravel application.

1. Under the Development Tools section, click the Advanced Tools icon, select the Go link, and select the Tools->Zip Push Deploy menu.
2. Delete the Azure-created default files from the application (if they exist).
3. Drag and drop your zip file onto the page.

**Deploy an Angular App to Microsoft Azure**

1. Create a new Web App (if new application).
2. Select the + Create a new Resource menu option.
3. 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.
4. After your new application deployment is finished, click the application link to test out. Click go to Resource.
5. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
6. Open the Web App from the Dashboard.
7. Deploy from a Build:
8. Select Advanced Tools and click the Go link.
9. Select the Debug console->CMD menu options.
10. Navigate to the site/wwwroot directory.
11. Delete the default content.
12. Zip up all the files under within the dist\APP\_NAME directory.
13. Drag and drop the zip file onto the right side of the CMD window.
14. Deploy from a GIT CI/CD Build Pipeline:
15. Select the Deployment Center menu option.
16. Select the GitHub CI/CD type (authorize access if necessary) and click the Continue button.
17. Select the GitHub Actions type and click the Continue button.
    1. Fill in the GitHub Repository
    2. Select the master branch.
    3. Select the NodeJS Runtime stack.
    4. Select the Node Version.
    5. Click the Finish button.
    6. In the GitHub repo, modify the GitHub build workflow file in the .guthub/workflows directory.
       1. Change the package: . entry to package: ./dist/[APP\_NAME]
       2. Remove the steps for Unit Testing

**Deploy a React App to Microsoft Azure**

1. Create a new Web App (if new application).
2. Select the + Create a new Resource menu option.
3. 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.
4. After your new application deployment is finished click the application link to test out. Click go to Resource.
5. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
6. Open the Web App from the Dashboard.
7. Deploy from a Build:
8. Select Advanced Tools and click the Go link.
9. Select the Debug console->CMD menu options.
10. Navigate to the site/wwwroot directory.
11. Delete the default content.
12. Zip up all the files within the build directory.
13. Drag and drop the zip file onto the right side of the CMD window.
14. Deploy from a GIT CI/CD Build Pipeline:
15. Select the Deployment Center menu option.
16. Select the GitHub CI/CD type (authorize access if necessary) and click the Continue button.
17. Select the GitHub Actions type and click the Continue button.
    1. Fill in the GitHub Repository.
    2. Select the master branch.
    3. Select the Node.JS Runtime stack
    4. Select the Node Version.
    5. Click the Finish button.
    6. In the GitHub repo modify the GitHub build workflow file in the .guthub/workflows directory.
    7. Change the package: . entry to package: ./build
    8. Remove the steps for Unit Testing.

# Part 2: Cloud Platform Deployment Notes for Heroku

**Deploy a Spring Framework or Spring Boot App to Heroku**

1. Create a Heroku account.

**NOTE:** You will need to create a Heroku account (this is free. For most services you will not need a credit card; you might be prompted for your credit card, however your card will NEVER be charged when using this activity as written).

1. Create app in Heroku:
   1. Click Create App button from the main page. Give your application a name. Click the Create App button.
   2. 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 GiHub, you can either copy your BitBucket repository to GitHub or use the Heroku CLI, as documented below.
   3. On the Project page, select the Settings Tab, click the Add Buildpack button, for Java click the Java button, click the Save Changes button.
   4. 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, then 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, delete your current JawsDB MySQL database Add-on and add a new JawsDB MySQL database.

1. Connect MySQL Workbench to the instance of MySQL Database. Run your DDL Script to configure the database.
2. You can deploy your application either through the Heroku CLI or using a GitHub repository. Follow either steps 5 or 6 below.
3. To deploy using the Heroku CLI, use the following steps:
   1. Download the Heroku Command Line Interface (CLI) from <https://devcenter.heroku.com>.
   2. If using the Spring Framework:
      * Build a WAR file using either Eclipse or Maven.
      * Deploy your WAR file by using the resources at <https://devcenter.heroku.com/articles/war-deployment>.
      * heroku war:deploy <path\_to\_war\_file> --app <app\_name>
   3. If using the Spring Boot Framework:
      * Build a JAR file using either Eclipse or Maven.
      * heroku jar:deploy <path\_to\_jar\_file> --app <app\_name>
   4. Test the app: https://[APP NAME].herokuapp.com.
4. To deploy using GitHub and a Build Pipeline, use the following steps:
5. Update your App Configuration as necessary in your GitHub repository, or clone the Heroku GIT repository provided by Heroku (go to your App Settings to get the URL), and add your app code to the cloned repository.
6. Add the webapp-runner Maven Plugin to your POM file:

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<version>3.0.2</version>

<executions>

<execution>

<phase>package</phase>

<goals><goal>copy</goal></goals>

<configuration>

<artifactItems>

<artifactItem>

<groupId>com.github.jsimone</groupId>

<artifactId>webapp-runner</artifactId>

<version>8.5.31.0</version>

<destFileName>webapp-runner.jar</destFileName>

</artifactItem>

</artifactItems>

</configuration>

</execution>

</executions>

</plugin>

1. Set the environment variable MAVEN\_CUSTOM\_OPTS in your App Settings to -P[Profile Name] if you are using Maven Profiles.
2. Add a Maven Heroku Procfile to the root of your project:
   * For Java Spring Framework:
     1. web: java $JAVA\_OPTS -jar target/dependency/webapp-runner.jar --port $PORT target/\*.war
   * For Java Boot:
     1. web: java $JAVA\_OPTS -jar --port $PORT target/\*.jar
3. To deploy using your GitHub repository and a Build Pipeline, use the following steps:
   * Update your App Configuration as noted above and push the changes to your GIT repository.
   * Create Heroku Pipeline and add your Spring app to it.
   * Go to your Heroku Pipeline and click on your Spring app, select the Deploy menu option, and make sure the Enable Automatic Deploys is enabled from your master branch.
   * Start a Build by performing either of one the following operations:
     1. Go to your Heroku Pipeline and click on your Spring app, select the Deploy menu option, and then click the Deploy Branch button.
     2. Push a code change to your GitHub repository.
   * Test your app at: https://[APP NAME].herokuapp.com.

**Deploy a PHP App to Heroku**

1. Create a Heroku account.

**NOTE:** You will need to create a Heroku account (this is free. For most services you will not need a credit card; you might be prompted for your credit card, however your card will NEVER be charged when using this activity as written).

1. Create app in Heroku:
   1. Click Create App button from the main page. Give your application a name. Click the Create App button.
   2. 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 GiHub, you can either copy your BitBucket repository to GitHub or use the Heroku CLI, as documented below.
   3. On the Project page, select the Settings Tab, click the Add Buildpack button, for PHP click the PHP button, click the Save Changes button.
   4. 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, delete your current JawsDB MySQL database Add-on and add a new JawsDB MySQL database.

1. Update your App Configuration as necessary:
2. Update the database configuration parameters in your code.
3. For non-Laravel apps, add an empty file (one with just { } as contents) named composer.json to your GitHub repository.
4. You can set the version of PHP using the following entry in your composer.json file: "require": { "php": "^7.1.0" }
5. See Heroku PHP Support located at <https://devcenter.heroku.com/articles/php-support>
6. Connect MySQL Workbench to the instance of MySQL Database. Run your DDL Script to configure the database.
7. You can deploy your application either through the Heroku GIT Repository, or by using your own GitHub or Bitbucket GIT repository. Follow either steps 5 or 6 below.

**NOTE:** Make sure to include all of the hidden files and the vendor folder in your GIT repository (you might need to modify the .gitignore file), and ensure the .env file is included as this is a required file to run a Laravel application. This should also be tested by cloning your GIT repository as a zip file, then deploying the clone files to your local MAMP, and validating that the application functions properly from the cloned repository.

1. To deploy using the Heroku GIT repository, use the following steps:
   1. Clone the Heroku GIT repository provided by Heroku (go to your App Settings to get the URL).
   2. Update your App Configuration as noted above.
   3. Push your code from your local repository to the Heroku GIT repository.
   4. Test the app: https://[APP NAME].herokuapp.com.
2. To deploy using your GitHub repository and a Build Pipeline, use the following steps:
   1. Update your App Configuration as noted above and push the changes to your GitHub repository.
   2. Create Heroku Pipeline and add your PHP app to it.
   3. Go to your Heroku Pipeline and click on your PHP app, select the Deploy menu option, and make sure the Enable Automatic Deploys is enabled from your master branch.
   4. Start a Build by performing either of one the following operations:
      1. Go to your Heroku Pipeline and click on your PHP app, select the Deploy menu option, and then click the Deploy Branch button.
      2. Push a code change to your GitHub repository.
   5. Test your app at: https://[APP NAME].herokuapp.com.

**Deploy an Angular 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:
      1. "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):
      1. "heroku-postbuild": "ng build --base-href ."
   3. Add a new file named Procfile to the repository with the following entry:
      1. 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:
   5. 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]'));
   6. The /route should contain the following code (and specify an APP\_NAME):
      1. res.sendFile(path.join(\_\_dirname + '/dist/[APP\_NAME]/index.html'));

![A screenshot of a social media post

Description automatically generated]()

1. Deploy from a Build:
   1. Run a build using the ng build --base-href . command.
   2. Push all the code, including the dist directory, to the repository.
   3. Select the Deploy menu from Heroku. Click the Deploy Branch button.
2. Deploy from a GIT CI/CD Build Pipeline:
   1. Push all the code excluding the dist directory to the repository.
   2. 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).
2. Select the New button and select the Create new app menu option. Give your app a Name. Click the Create App button.
3. Select the GitHub deployment method and connection to your GitHub repository.
4. After your new application deployment is finished, click the application link to test out. Click go to Resource.
5. Add new database and initialize with a DDL for your application if needed. Make sure to select a database from the free tier.
6. Open the Web App from the Dashboard.
7. Configure the application:
8. Update package.json to include the express library in the list of dependencies specifiying the version of express used in development:
   1. "express": "^4.17.1"
9. 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):
   1. "heroku-postbuild": "react-scripts build"
10. Add a new file named Procfile to the repository with the following entry:
    1. web: node server.js
11. Add a new file named server.js to the repository that will be used to serve up the React application:
    1. Set the following code to initialize the Express application:
       1. app.use(express.static(\_\_dirname);
       2. app.use(express.static(path.join(\_\_dirname, 'build');
    2. The /route should contain the following code:
       1. res.sendFile(path.join(\_\_dirname, 'build', 'index.html');

![A screenshot of a social media post

Description automatically generated]()

1. Deploy from a Build:
   1. Run a build using the npm run build command.
   2. Push all the code, including the build directory, to the repository.
   3. Select the Deploy menu from Heroku. Click the Deploy Branch button.
2. Deploy from a GIT CI/CD Build Pipeline:
   1. Push all the code, excluding the build directory, to the repository.
   2. Select the Deploy menu from Heroku. Click the Enable Automatic Deploy button. Click the Deploy Branch button.

# Part 3: Cloud Platform Deployment Notes for Red Hat OpenShift

**Deploy a Spring Framework App to Red Hat OpenShift**

Set up and configure the OpenShift JBoss Tomcat MySQL Container (note this is an optional activity per instructors guidance):

1. Create a new Project in OpenShift named CST-323.
2. Click the Add to Project menu.
3. Add the following types to the Project: Wildfly image for the Spring project, and when prompted for a Github Repository URL for your project, enter the URL for your GitHub or Bitbucket repository.
4. Click the Add to Project menu.
5. Add the following Data Store to the Project: MySQL datastore image for the MySQL database.
6. Once you have added both Containers to your Project you will need to group the Wildfly Container with the MySQL Container.

You can initialize the MySQL Database by either using RSH or by using Port Forwarding (this is the easiest solution).

Initialize the MySQL Database by using RSH into the POD (refer to the article: [Migrating Database Applications](https://docs.openshift.com/online/dev_guide/migrating_applications/database_applications.html)):

1. Download the OpenShift Command Line Interface tool from [Red Hat OpenShift Online](https://console.starter-us-east-1.openshift.com/console/command-line) (log in to the Red Hat account you created for this activity).
2. Open a Terminal Window or DOS Box.
3. Navigate to the path were the Command Line Tool was installed.
4. Get the MySQL Pod Name: ./oc get pods
5. Copy your SQL DDL script from your development environment into a directory called /local/db on your local file system.
6. Copy the SQL DDL script to the OpenShift MySQL Pod: ./oc rsync /local/db <mysql\_pod\_name>:/var/lib/mysql/data
7. Remote into the MySQL Pod: ./oc rsh <mysql\_pod>
8. Run the command: cd /var/lib/mysql/data
9. Run the SQL DDL Script: mysql -u root
10. Run the command: source all.sql
11. Grant privileges to petclinic user: grant all privileges on all tables to petclinic
12. Run the command: flush privileges

OR

Initialize MySQl Database using Port Forwarding and MySQL Workbench (refer to the following article: [Connecting to a Database Using Port Forwarding in OpenShift](https://blog.openshift.com/openshift-connecting-database-using-port-forwarding/)):

1. Log into OpenShift: ./oc [OpenShift Online Web Console URL]
2. Get your MySQL Pod Name: ./oc get pods
3. To find your MySQL Database IP, Port, and Credentials by using RSH into the MySQL Database Pod and dumping the environment variables related to MySQL:
4. ./oc rsh [MYSQL\_POD]
5. env | grep MYSQL
6. exit
7. Run a Port Forward:

./oc port-forward [MYSQL\_POD] [MYSQL\_WORKBENCH\_LOCAL\_PORT]:[MYSQL\_PORT]

**NOTE:** This will not be permanent, and will do port forwarding as long as your Terminal Session is running.

1. Run MySQL Workbench connecting to:

127.0.0.1, [MYSQL\_WORKBENCH\_LOCAL\_PORT], your MySQL Database credentials

1. Run the your DDL script generated from MySQL Workbench
2. Exit MySQL Workbench
3. Exit the Terminal Session by entering Ctrl-C

**NOTE:** Because you will not have enough quota during deployment, you will need to change the Container deployment strategy for this project. This can be done by selecting the following menu options: click Applications -> Deployments main menu items, select the name of your application, select the Actions -> Edit drop down menu options, and change the Deployment Strategy Type from Rolling to Recreate.

Build and Deploy your Application:

1. Select the Builds->Builds main menu items. Click on the name of your Wildlfy Container.
2. Click the Start Build button. Note, you can monitor your build by clicking on the View Log link. Validate that your build was successful.
3. Click the Overview main menu item. Note, you can monitor the build and deployment from this screen.

**Deploy a PHP App to Red Hat OpenShift**

Set up and Configure the OpenShift PHP Container:

1. Create a new Project in OpenShift named CST-323.
2. Click the Add to Project menu.
3. Add the following types to the Project: PHP 7.0 image for a PHP or PHP Laravel project. When prompted for a Github Repository URL for your Project, enter the URL for your 'cst-323' Github or Bitbucket repository.
4. Click the Add to Project menu.
5. Add the following Data Store to the Project: MySQL datastore image for the MySQL database.
6. Once you have added both Containers to your Project you will need to group the PHP Container with the MySQL Container.

You can initialize the MySQL Database by either using RSH or by using Port Forwarding (this is the easiest solution).

Initialize the MySQL Database using RSH into the POD (refer to the article: [Migrating Database Applications](https://docs.openshift.com/online/dev_guide/migrating_applications/database_applications.html)):

1. Download the OpenShift Command Line Interface tool from [Red Hat OpenShift Online](https://console.starter-us-east-1.openshift.com/console/command-line) (log into your Red Hat account you created for this activity).
2. Open a Terminal Window or DOS Box.
3. Navigate to the path were the Command Line Tool was installed.
4. Get the MySQL Pod Name: ./oc get pods
5. Copy your SQL DDL script from your development environment into a directory called /local/db on your local file system.
6. Copy the SQL DDL script to the OpenShift MySQL Pod: ./oc rsync /local/db <mysql\_pod\_name>:/var/lib/mysql/data
7. Remote into the MySQL Pod: ./oc rsh <mysql\_pod>
8. Run the command: cd /var/lib/mysql/data
9. Run the SQL DDL Script: mysql -u root
10. Run the command: source all.sql
11. Grant privileges to petclinic user: grant all privileges on all tables to petclinic
12. Run the command: flush privileges

OR

Initialize MySQl Database using Port Forwarding and MySQL Workbench (refer to the following article: [Connecting to a Database Using Port Forwarding in OpenShift](https://blog.openshift.com/openshift-connecting-database-using-port-forwarding/)):

1. Log into OpenShift: ./oc [OpenShift Online Web Console URL]
2. Get your MySQL Pod Name: ./oc get pods
3. To find your MySQL Database IP, Port, and Credentials by using RSH into the MySQL Database Pod and dumping the environment variables related to MySQL:
4. ./oc rsh [MYSQL\_POD]
5. env | grep MYSQL
6. exit
7. Run a Port Forward:

/oc port-forward [MYSQL\_POD] [MYSQL\_WORKBENCH\_LOCAL\_PORT]:[MYSQL\_PORT]

**NOTE:** This will not be permanent, and will do port forwarding as long as your Terminal Session is running.

1. Run MySQL Workbench connecting to:

127.0.0.1, [MYSQL\_WORKBENCH\_LOCAL\_PORT], your MySQL Database credentials

1. Run the your DDL script that was generated using MySQL Workbench
2. Exit MySQL Workbench
3. Exit the Terminal Session by entering Ctrl-C

**NOTE:** Because you will not have enough quota during deployment, you will need to change the Container deployment strategy for this project. This can be done by selecting the following menu options: click Applications -> Deployments main menu items, select the name of your application, select the Actions -> Edit drop down menu options, and change the Deployment Strategy Type from Rolling to Recreate.

Build and Deploy your Application:

1. Select the Builds->Builds main menu items. Click on the name of your PHP Container.
2. Click the Start Build button. Note, you can monitor your build by clicking on the View Log link. Validate that your build was successful.
3. Click the Overview main menu item. Note, you can monitor the build and deployment from this screen.

# Part 4: Cloud Platform Deployment Notes for Amazon AWS

**Deploy a Spring Framework App to Amazon AWS:**

**NOTE:** You will need to create an AWS account (this is free 12-month service for most services; you will need a credit card and your card will NEVER be charged when using this activity as written).

Create and Configure an AWS MySQL Database:

1. Log into AWS and select Services from the main menu.
2. Select RDS.
3. Under the Create database section, click the Create database button.
4. Select the MySQL engine option and the 5.6 version edition radio button. Check the 'Enable options for free tier'. Click the Next button.
5. Fill out the Specify DB details form:
6. From Settings enter DB instance identifier enter instance name (i.e., mydatabaseinstance).
7. From Settings, enter Master username and password.
8. Click the Next button.
9. Under Network and Security section, set the Public Accessibility setting to Yes.
10. Leave all defaults in the Configure advanced settings form.
11. From the RDS Dashboard, select your database instance.
12. Your database URL is listed under the Connect section under the Endpoint value.
13. Make your database accessible from a Java application by clicking the Security groups link under the Details section for the database.
14. In the Security Group setup, select the Inbound Tab. Click the Edit button. Under the Source dropdown, select the Anywhere option.
15. In MySQL Workbench, set up a connection using the AWS Database Endpoint URI and credentials. Create the 'cst-323' schema and tables by running the DDL script created from your development environment.

Create and Configure an AWS Tomcat Application:

1. Select Elastic Beanstalk service.
2. Click the 'Create new Application' link from the top right menu.
3. Give your Application Name (i.e., TestApp). Click the Create button.
4. Create your Application Environment by clicking the 'Create one now' link.
5. Select the 'Web server environment' and click the Select button.
6. Fill out the following fields in the Creating web server environment form:
7. From Environment information Domain: Give your Application a name (i.e., test-app).
8. From Base configuration: Select Tomcat from the Preconfigured platform options. Upload a WAR file of your Java application.
9. Click the Create Environment button. Wait for environment to be built.
10. From the Elastic Beanstalk application screen, click the App URL to validate application is running properly.

Deploying Manually:

1. Create a WAR file using Maven.
2. Log into AWS and select Services from the main menu.
3. Select Elastic Beanstalk. Select your Application.
4. Click the Upload and Deploy button. Upload your WAR file and give your build a label. Click the Deploy button.

Deploying using an AWS Code Pipeline:

1. Update app code in source control: Set up config folder and Maven Profile.
2. Add a buildspec.yml to the root of your application code.
3. Log into AWS and select Services from the main menu.
4. Select the CodePipeline service.
5. Click the Create Pipeline button.
6. Give your pipeline a name (i.e., TestAppPipeline). Click the Next step button.
7. Select GitHub from the Source provider dropdown. Click the Connect to GitHub button and select your repo and master branch. Click the Next step button.
8. Select AWS CodeBuild from the Build provider dropdown. Select the Create a new build project option. Give your build a name. Select Ubuntu operating system with the Java OpenJDK 8 runtime.
9. Create a Service Role with a name (i.e., testapp-build-role)
10. Click the Save build project button. Click the Next step button.
11. Select AWS Elastic Beanstalk from the Deployment provider dropdown. Choose your Application and Environment from the dropdowns. Click the Next step button.
12. Create an AWS Service Role. Click the Next step button.
13. Click the Create pipeline button.
14. To build and deploy your application:
15. Select the CodePipeline service from the Services dashboard. Open the Pipeline.
16. Either make a change to code in GitHub, or click the Release change button to start a build and deployment.
17. To access your application:
18. Select the Elastic Beanstalk service from the Services dashboard. Open your Application.
19. Test your application: https://[APP NAME].[AWS REGION].elasticbeanstalk.com/

**Deploy a PHP App to Amazon AWS**

**NOTE:** You will need to create an AWS account (this is free 12-month service; you will need a credit card and your card will NEVER be charged when using this activity as written).

Create and configure an AWS MySQL database:

1. Log into AWS and select Services from the main menu.
2. Select RDS.
3. Under the Create database section, click the Create database button.
4. Select the MySQL engine option and the 5.6 version edition radio button. Check the 'Enable options for free tier'. Click the Next button.
5. Fill out the Specify DB details form:
6. From Settings enter DB instance identifier enter instance name (i.e., mydatabaseinstance).
7. From Settings, enter Master username and password.
8. Click the Next button.
9. Leave all defaults in the Configure advanced settings form.
10. From the RDS Dashboard select your database instance.
11. Your database URL is listed under the Connect section under the Endpoint value.
12. Make your database accessible from a Java application by clicking the Security groups link under the Details section for the database.
13. In the Security Group setup select the Inbound Tab. Click the Edit button. Under the Source dropdown select the Anywhere option.
14. In MySQL Workbench setup a connection using the AWS Database Endpoint URI and credentials. Create the 'cst-323' schema and tables by running the DDL script created from your development environment.

Create and Configure the AWS PHP Application:

1. Update app code in source control:
2. It is possible to set environment specific configuration in your .env file for a PHP Laravel project.
3. Log into AWS and select Services from the main menu.
4. Select Elastic Beanstalk service.
5. Click the 'Create new Application' link from the top right menu.
6. Give your Application Name (i.e., TestApp). Click the Create button.
7. Create your Application Environment by clicking the 'Create one now' link.
8. Select the 'Web server environment' and click the Select button.
9. Fill out the following fields in the Creating web server environment form:
10. From Environment Information Domain: Give your Application a name (i.e., test-app).
11. From Base configuration: Select PHP from the Preconfigured platform options. Upload a ZIP file of your PHP application.
12. Click the Create Environment button. Wait for environment to be built.
13. From the Elastic Beanstalk application screen click the App URL to validate application is running properly.

Deploy Manually:

1. Create a ZIP file with all your code (make sure to update APP\_ENV to amazon in .env).
2. Log into AWS and select Services from the main menu.
3. Select Elastic Beanstalk. Select your Application.
4. Click the Upload and Deploy button. Upload your ZIP file and give your build a label. Click the Deploy button.

Deploy using an AWS Code Pipeline:

1. Add a buildspec.yml to the root of your application code.
2. Log into AWS and select Services from the main menu.
3. Select the CodePipeline service.
4. Click the Create Pipeline button.
5. Give your pipeline a name (i.e., TestAppPipeline). Click the Next step button.
6. Select GitHub from the Source provider dropdown. Click the Connect to GitHub button, and select your repo and master branch. Click the Next step button.
7. Select AWS CodeBuild from the Build provider dropdown. Select the Create a new build project option. Give your build a name. Select Ubuntu operating system with the Base runtime.
8. Create a Service Role with a name (i.e., testapp-build-role)
9. Click the Save build project button. Click the Next step button.
10. Select AWS Elastic Beanstalk from the Deployment provider dropdown. Choose your Application and Environment from the dropdowns. Click the Next step button.
11. Create an AWS Service Role. Click the Next step button.
12. Click the Create pipeline button.
13. To build and deploy your application:
14. Select the CodePipeline service from the Services dashboard. Open the Pipeline.
15. Either make a change to code in GitHub, or click the Release change button to start a build and deployment.
16. To access your application:
17. Select the Elastic Beanstalk service from the Services dashboard. Open your Application.
18. Test your application: https://[APP NAME].[AWS REGION].elasticbeanstalk.com/

**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).
   1. Click the Create a new application button.
   2. Give your application a name. Click the Create button.
   3. Click the Create a new environment button.
      1. Web server environment
      2. Platform of Node.js
      3. Platform branch of Node.js 10 running on 64bit Linux
      4. Sample application
      5. Click the Create button
      6. Once the environment is up and running, test the Sample application to ensure all is working properly
      7. 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:
   1. Update package.json to include the express library in the list of dependencies specifiying the version of express used in development:
      1. "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):
      1. "heroku-postbuild": "ng build --base-href ."
   3. Add a new file named Procfile to the repository with the following entry:
      1. 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:
      1. 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]'));
      2. The /route should contain the following code (and specify an APP\_NAME):
         1. res.sendFile(path.join(\_\_dirname + '/dist/[APP\_NAME]/index.html'));

![A screenshot of a social media post

Description automatically generated]()

1. Deploy from a Build:
   1. Run a build using the ng build --base-href . command.
   2. Zip up all the code and files within the dist directory but do not include the e2e, node\_modules, and src directories.
   3. Click on the Upload and deploy button.
   4. Click the Choose file button and select your zip file.
   5. Click the Deploy button.
2. Deploy from a GIT CI/CD Build Pipeline:
   1. Configure code and set up build pipeline (if not already completed):
      1. Add a buildspec.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).
   1. Click the Create a new application button.
   2. Give your application a name. Click the Create button.
   3. Click the Create a new environment button.
      1. Web server environment
      2. Platform of Node.js
      3. Platform branch of Node.js 10 running on 64bit Linux
      4. Sample application
      5. Click the Create button
      6. Once the environment is up and running, test the Sample application to ensure all is working properly
      7. 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:
   1. Update package.json to include the express library in the list of dependencies specifiying the version of express used in development:
      1. "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):
      1. "heroku-postbuild": "react-scripts build"
   3. Add a new file named Procfile to the repository with the following entry:
      1. 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:
      1. Set the following code to initialize the Express application:
         1. app.use(express.static(\_\_dirname);
         2. app.use(express.static(path.join(\_\_dirname, 'build');
      2. The /route should contain the following code:
         1. res.sendFile(path.join(\_\_dirname, 'build', 'index.html');

![A screenshot of a social media post

Description automatically generated]()

1. Deploy from a Build:
   1. Run a build using the npm run build command.
   2. Zip up all the code and files within the build directory, but do not include the node\_modules and src directories.
   3. Click on the Upload and deploy button.
   4. Click the Choose file button and select your zip file.
   5. Click the Deploy button.
2. Deploy from a GIT CI/CD Build Pipeline:
   1. Configure code and set up build pipeline (if not already completed):
   2. Add a buildspec.yml to the root of your application code for Node 10 application.

![A screenshot of a social media post

Description automatically generated]()

* 1. Log into AWS and select Services from the main menu.
  2. Select the CodePipeline service.
  3. Click the Create Pipeline button.
  4. Give your pipeline a name (i.e., TestAppPipeline). Click the Next step button.
  5. 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.
  6. 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.
  7. Click the Create Project button.
  8. Click the Next step button.
  9. Select AWS Elastic Beanstalk from the Deployment provider dropdown. Choose your Application and Environment from the dropdowns. Click the Next step button.
  10. Click the Create pipeline button.
  11. To build and deploy your application:
      1. Select the CodePipeline service from the Services dashboard. Open the Pipeline.
      2. Either make a change to code in GitHub, or click the Release change button to start a build and deployment.

# Part 5: Cloud Platform Deployment Notes for Google Cloud

**Deploy a Spring Framework App to Google Cloud**

**NOTE:** You will need to create a Google Cloud Platform account (this requires a credit card; however, it is free for 12 months and your card will NEVER be charged when using this activity as written).

Create Java (Jetty) Container and deploy your application in the Google App Engine (GAE):

1. Create an App Engine application of type Java using the following steps:
   1. Select App Engine from the Main Menu.
   2. Click the ‘Select a Project’ dropdown list and then click the New Project icon.
   3. Give your Project a Name and click the Create button.
   4. From the Welcome to App Engine screen, click the Create Application button.
   5. Select a Region from the US and click the Create App button.
   6. Select Java from Language list and a Standard Environment. Click the Next button.
2. Clone your Application Code from GIT (from Google Cloud Shell) using the following steps:
3. Open up a Cloud Shell from the Activate Cloud Shell icon in the top menu. From the Cloud Shell 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.
4. Run the following command from the Cloud Shell:
   * git clone [URL to your Test App Repo]
5. Configure your application using the following steps:
6. Update POM file:

<plugin>

<groupId>com.google.cloud.tools</groupId>

<artifactId>appengine-maven-plugin</artifactId>

<version>1.3.1</version>

</plugin>

<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>

1. Add an app.yaml configuration for a Java app into the root directory of the application. In order to get MySQL database connectivity, you must add the following entry and replace the cloud\_sql\_instances setting with the Instance Connection Name for your MySQL database instance. There are sample files available in the Google Cloud documentation, or one can be provided by your instructor.



1. Add an appengine-web.xml configuration file to the WEB-INF directory of your project. There are sample files available in the Google Cloud documentation, or one can be provided by your instructor.
2. If you are using logging you should configure the path for your log file to /tmp/[APP\_NAME]/logs/[LOG\_FILENAME].log in appengine-web.xml.
3. Update your database configuration for your application (i.e., config.properties to setup db.connection property for Google MySQL database).
   * **NOTE:** the JDBC Connection String for MySQL requires the following format:

jdbc:mysql://google/[SCHEMA]?socketFactory=com.google.cloud.sql.mysql.SocketFactory&amp;cloudSqlInstance=[PROJECT\_NAME\_ID]:[DB\_REGION]:[DB\_INSTANCE\_NAME]

1. Create the MySQL Database Container and initialize the schema in the Google Cloud Platform using the following steps:
2. Select SQL menu item from the Main Menu.
3. Select MySQL Database Engine and click the Next button.
4. Select the MySQL Second Generation type.
5. Fill out the Instance ID, root password, and region.
6. Expand the Show Configuration Options. Select the desired MySQL version under the ‘Choose data version” dropdown. Expand the ‘Configure machine type and storage’ dropdown. Click the Change button. Select the db-f1-micro under the Shared-core machines options. Click the Select button. **NOTE:** it is extremely important that these options are set to avoid being charged by Google for your database usage.
7. Click the Create button.
8. Open the instance of the new Database and note your Public IP Address.
9. Select the Users menu and then create a new user [DB\_USERNAME]/[DB\_PASSWORD] that is available for all hosts. Click the Create button.
10. Select the Database menu and then create a new Database (your schema).
11. Get your public IP Address by going to your browser and, in the search bar, entering ‘My IP’. Note your IP Address for the next step.
12. 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.
13. Setup a MySQL Workbench connection using the databases IP address (listed in the Overview menu) and your database credentials (setup from the prior step).
14. Connect to the database in MySQL Workbench and run your DDL script.
15. In the main Google menu, go to APIs & Services, click on the Library menu, search for Google Cloud SQL, and make sure both Cloud SQL and Cloud SQL Admin API are enabled.
16. Update your database configuration for your application (i.e., config.properties to set up db.connection property for Google MySQL database).
    * **NOTE:** the JDBC Connection String for MySQL requires the following format:

jdbc:mysql://google/[SCHEMA]?socketFactory=com.google.cloud.sql.mysql.SocketFactory&amp;cloudSqlInstance=[PROJECT\_NAME\_ID]:[DB\_REGION]:[DB\_INSTANCE\_NAME]

1. Build and Deploy your application using the following steps:
2. Open your Cloud Shell.
3. cd to your cloned project root directory
4. Optionally Test locally in Shell:
   * mvn -P[PROFILE] clean appengine:run
     + **NOTE:** use of Maven Profiles is optional, and if not used, leave the P command line option off when running mvn from the command line.
   * **NOTE:** comment out the google-api-client and google-api-client-appengine as dependencies in your maven file
   * TEST: click on the Web Preview icon in the Shell and go to https://[project name].appspot.com/
5. Deploy to App Engine: mvn -P[PROFILE] clean appengine:deploy
   * **NOTE:** use of Maven Profiles is optional, and if not used, leave the P command line option off when running mvn from the command line.
6. Test at https://[PROJECT\_NAME].appspot.com/
   * To view Exception Stack Track and Error logs, you can go to the Home menu item and go under the Error Reporting section to view your most recent errors.
   * To view Application logs (to the console), go to App Engine->Versions and select Logs from the Tools dropdown.

**Deploy a PHP App to Google Cloud**

**NOTE:** You will need to create a Google Cloud Platform account (this requires a credit card; it is free for 12 months, and your card will NEVER be charged when using this activity as written).

Create PHP Container and deploy your application in the Google App Engine (GAE):

1. Create an App Engine application of type PHP using the following steps:
   1. Select App Engine from the Main Menu.
   2. Click the ‘Select a Project’ dropdown list and then click the New Project icon.
   3. Give your Project a Name and click the Create button.
   4. From the Welcome to App Engine screen, click the Create Application button.
   5. Select a Region from the US and click the Create App button.
   6. Select PHP from Language list and a Flexible Environment. Click the Next button.
2. Clone your Application Code from GIT (from Google Cloud Shell) using the following steps:
3. Open up a Cloud Shell from the Activate Cloud Shell icon in the top menu. From the Cloud Shell perform the following operations.
   * **NOTE:** if you click on the Pencil icon, this will open a tree view of your code, which allows you to edit some of your configuration files.
4. Run the following command from the Cloud Shell:
   * git clone [URL to your Test App Repo]
5. Configure your application using the following steps:
6. Add an app.yaml configuration for a PHP app into the root directory of the application. In order to get MySQL database connectivity, you must add the following entry and replace the cloud\_sql\_instances setting with the Instance Connection Name for your MySQL database instance. There are sample files available in the Google Cloud documentation or one can be provided by your instructor.



1. To Update your APP\_KEY in the app.yaml run: php artisan key:generate --show
2. **NOTE:** Apache Web Server is not used in Google App Engine, so the public rewrite rule is invalid. You must set your document\_root to public and copy all JS, CSS, and IMG from /resources/assets to /public/resources/assets.
3. If using PHP Laravel, you can update environment specific configuration in the .env file. Refer to [Run Laravel on Google App Engine Flexible Environment](https://cloud.google.com/community/tutorials/run-laravel-on-appengine-flexible)
4. Update composer.son require section (PHP v7.2 does not work at this point), ensure your Laravel version is correct, and run the following post install commands (for non-Laravel PHP applications see the notes below):

"require": {

"php": "7.1.\*",

"laravel/framework": "5.4.\*",

"laravel/tinker": "~1.0"

},

"post-install-cmd": [

"Illuminate\\Foundation\\ComposerScripts::postInstall",

"mkdir -p bootstrap/cache",

"chmod -R 755 app bootstrap storage",

"mkdir -p storage”,

"mkdir -p storage/app",

"mkdir -p storage/framework/cache",

"mkdir -p storage/framework/sessions",

"mkdir -p storage/framework/views",

"mkdir -p storage/logs

],

NOTE: If you are deploying a non-Laravel application, then you must configure the runtime\_config section in your yaml.xml as shown below, replacing the cloud\_sql\_instances setting with the Instance Connection Name for your MySQL database instance, and make sure you copy all your PHP code files into a public direction in your project.

![A picture containing bird

Description automatically generated]()

NOTE: if you are deploying a non-Laravel application, then you must configure the composer.json file as shown below and set the version of PHP accordingly.

A picture containing knife

Description automatically generated

1. Create the MySQL Database Container and initialize the schema in the Google Cloud Platform using the following steps:
2. Select SQL menu item from the Main Menu.
3. Select MySQL Database Engine and click the Next button.
4. Select the MySQL Second Generation type.
5. Fill out the Instance ID, root password, region, and click the Create button.
6. Expand the Show Configuration Options. Select the desired MySQL version under the ‘Choose data version” dropdown. Expand the ‘Configure machine type and storage’ dropdown. Click the Change button. Select the db-f1-micro under the Shared-core machines options. Click the Select button. NOTE: it is extremely important that these options are set to avoid being charged by Google for your database usage.
7. Open the instance of the new Database and note your Public IP Address.
8. Select the Users menu and then create a new user [DB\_USERNAME]/[DB\_PASSWORD] that is available for all hosts. Click the Create button.
9. Select the Database menu and then create a new Database (your schema).
10. Get your public IP Address by going to your browser and, in the search bar, entering ‘My IP’. Note your IP Address for the next step.
11. 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.
12. Setup a MySQL Workbench connection using the databases IP address (listed in the Overview menu) and your database credentials (setup from the prior step).
13. Connect to the database in MySQL Workbench and run your DDL script.
14. In the main Google menu, go to APIs & Services, click on the Library menu, search for Google Cloud SQL, and make sure both Cloud SQL and Cloud SQL Admin API are enabled.
15. Update your database configuration for your application.
    * NOTE: see the following mysqli and PDO Database Connection examples:

mysqli:



PDO:



1. Build and deploy your application using the following steps:
2. Open your Cloud Shell
3. cd to your cloned project root directory
4. Optionally Test locally in Shell: php -S 0.0.0.0:8080 -t ./
5. Deploy to App Engine: gcloud app deploy
6. Test at <https://[PROJECT_NAME].appspot.com/>
   1. To view application logs, you can go to the Home menu item and go under the Error Reporting section to view your most recent errors.
   2. To view application logs, go to App Engine Versions and select Logs from the Tools dropdown.

**Deploy an Angular App to Google Cloud**

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

![A screenshot of a cell phone

Description automatically generated]()

1. Deploy from a Build:
   1. Activate the Cloud Shell for your project:
   2. Create a working directory or navigate to an existing working directory.
      1. 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).
   3. Clone the GIT Repository using the 'git clone [GIT REPO URL]' command. Navigate to the root of your project code.
   4. Run the 'npm install' command.
   5. 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).
   6. 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:
   7. Update: "start": "serve -s dist/[APP NAME]",
   8. Add: "prestart": "npm install -g serve",
   9. Run the 'gcloud app deploy --project [PROJECT ID]' command.
   10. 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.
2. Deploy from a GIT CI/CD Build Pipeline:
   1. Select Cloud Build from the Google Cloud main menu.
   2. 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.
   3. Click the Settings menu. Enable App Engine Service account permissions.
   4. 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.
   5. Create a Google Cloud Build file named cloudbuild.yaml and add this to the root of the project in GitHub.

![A screenshot of a cell phone

Description automatically generated]()

* 1. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts:
  2. Update: "start": "serve -s build",
  3. Add: "prestart": "npm install -g serve",
  4. To build and deploy your application:
  5. Select the Cloud Build->Triggers menu open options.
     1. 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.
   1. Give your project a name, click the Create button, then select a US region and NodeJS for framework.
   2. 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:
   4. Add a Google Cloud app.yaml file to the root of your repository.
   5. Set runtime to nodejs and the environment to flex and the desired minimum manual scaling settings (see the Google Cloud web site for examples).

![A screenshot of a cell phone

Description automatically generated]()

1. Deploy from a Build:
   1. Activate the Cloud Shell for your project:
   2. Create a working directory, or navigate to an existing working directory.
      1. 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).
   3. Clone the GIT Repository using the 'git clone [GIT REPO URL]' command. Navigate to the root of your project code.
   4. Run the 'npm install' command.
   5. 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).
   6. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts:
   7. Update: "start": "serve -s build",
   8. Add: "prestart": "npm install -g serve",
   9. Run the 'gcloud app deploy --project [PROJECT ID]' command.
   10. 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.
2. Deploy from a GIT CI/CD Build Pipeline:
   1. Select Cloud Build from the Google Cloud main menu.
   2. 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.
   3. Click the Settings menu. Enable App Engine Service account permissions.
   4. 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.
   5. Create a Google Cloud Build file named cloudbuild.yaml and add this to the root of the project in GitHub.

![A screenshot of a cell phone

Description automatically generated]()

* 1. Update the package.json file (you can use the build in Code Editor in the Cloud Shell) with the following changes to the scripts:
     1. Update: "start": "serve -s build",
     2. Add: "prestart": "npm install -g serve",
  2. To build and deploy your application:
     1. Select the Cloud Build->Triggers menu open options.
     2. 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.