JAVAMS12 Deploying to Kubernetes Engine

2 hoursFree

Rate Lab

Overview

In this series of labs, you take a demo microservices Java application built with the Spring framework and modify it to use an external database server. You adopt some of the best practices for tracing, configuration management, and integration with other services using integration patterns.

In an earlier lab, you repacked the application and deployed it to App Engine. You can easily modify Spring applications so that they can be built into container packages. The packages can then be quickly and efficiently deployed into a container environment such as Kubernetes Engine.

Kubernetes is a portable, extensible open source platform for managing containerized workloads and services. It facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes Engine is Google's managed, production-ready environment for deploying containerized applications. Kubernetes Engine enables rapid application development and iteration by making it easy to deploy, update, and manage your applications and services. Kubernetes Engine enables you to quickly get up and running with Kubernetes by eliminating the need to install, manage, and operate your own Kubernetes clusters.

In this lab, you build the application into a container and then deploy the containerized application to Kubernetes Engine.

Objectives

In this lab, you learn how to perform the following tasks:

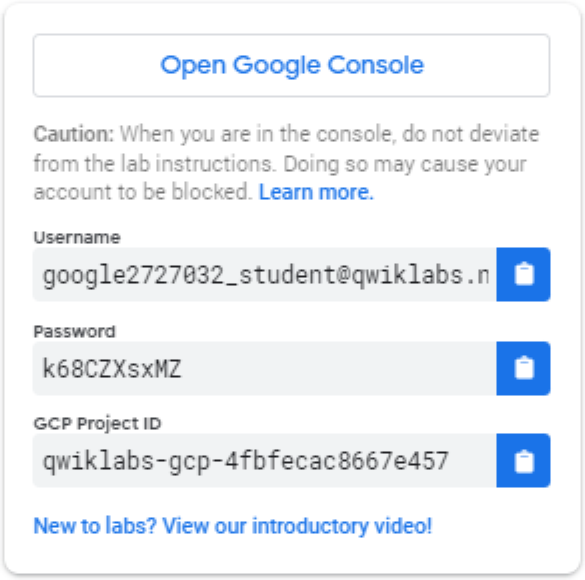
- Create a Kubernetes Engine cluster
- Create a containerized version of a Java application
- Create a Kubernetes deployment for a containerized application

Task 0. Lab Preparation

Access Qwiklabs

How to start your lab and sign in to the Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



Open Google Console

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username

google2727032_student@qwiklabs.n

Password

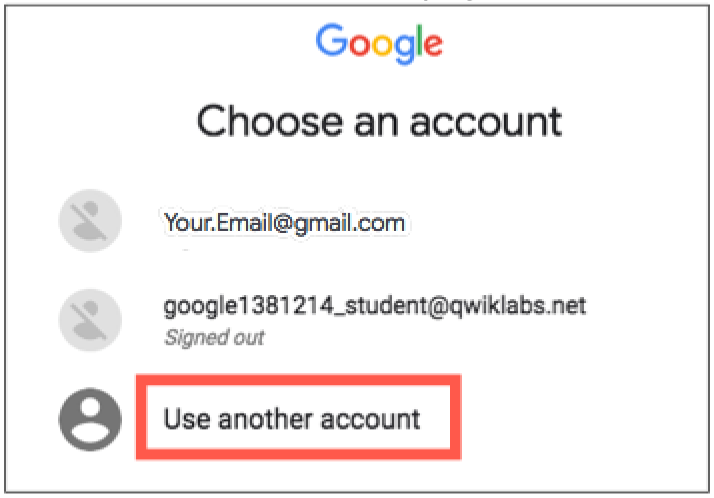
k68CZXsxMZ

GCP Project ID

qwiklabs-gcp-4fbfecac8667e457

New to labs? View our introductory video!

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.
   Tip: Open the tabs in separate windows, side-by-side.
3. On the Choose an account page, click **Use Another Account**.



Google

Choose an account

Your.Email@gmail.com

google1381214_student@qwiklabs.net
Signed out

Use another account

4. The Sign in page opens. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

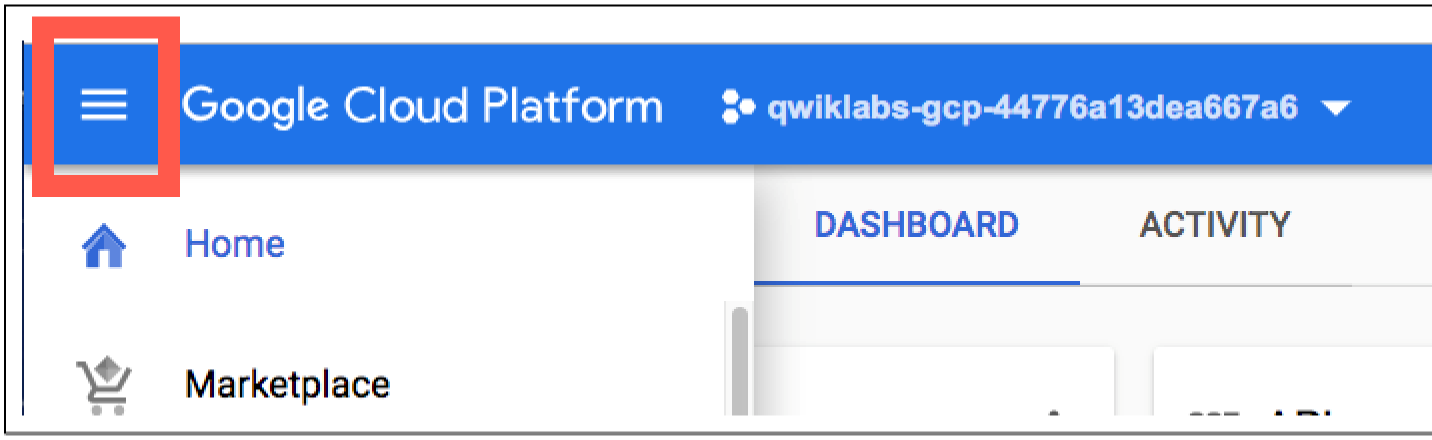
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own GCP account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:

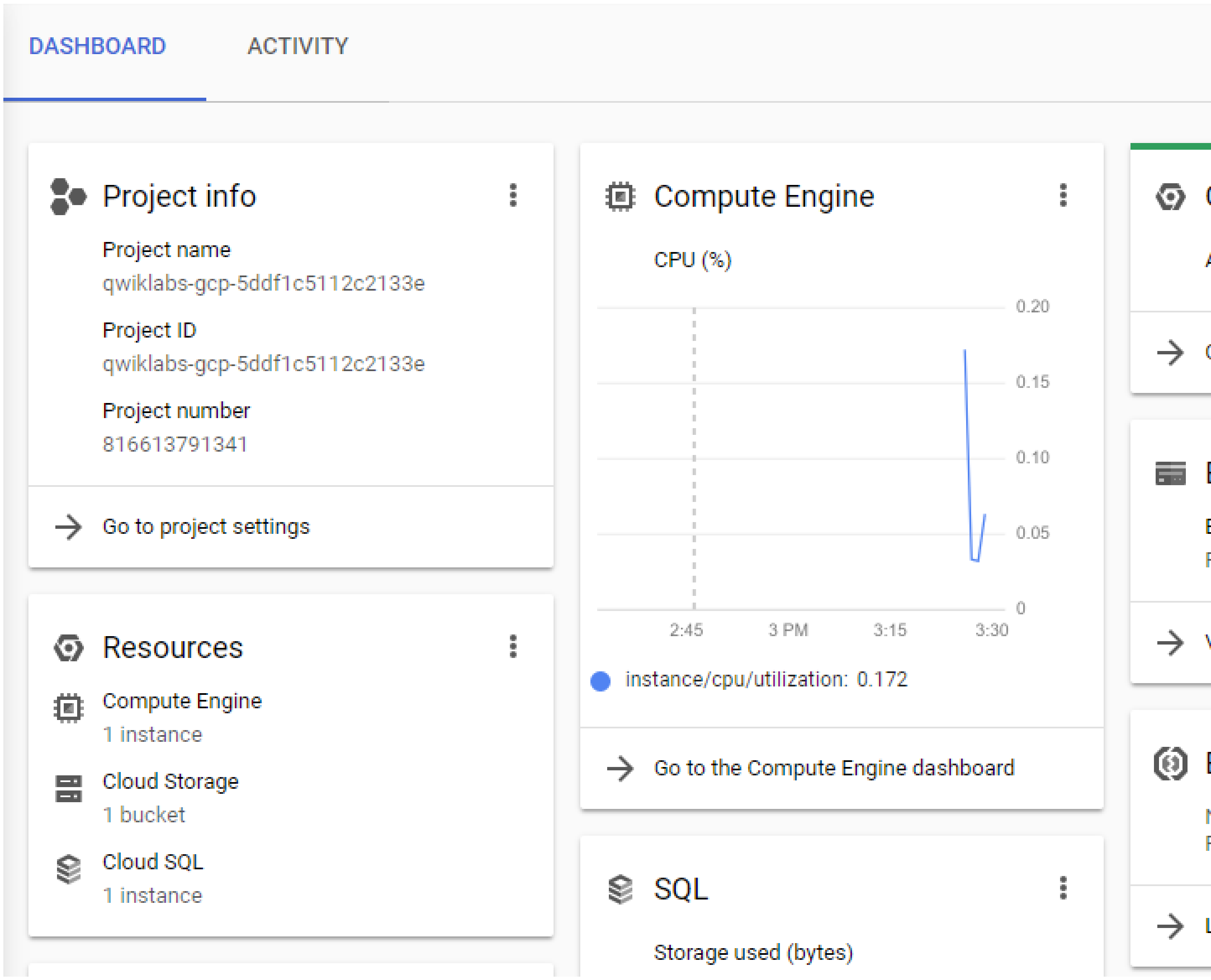
   - Accept the terms and conditions.
   - Do not add recovery options or two-factor authentication (because this is a temporary account).
   - Do not sign up for free trials.

After a few moments, the GCP console opens in this tab.

Note: You can view the menu with a list of GCP Products and Services by clicking the **Navigation menu** at the top-left, next to "Google Cloud Platform".



After you complete the initial sign-in steps, the project dashboard appears.



Project info

Project name
qwiklabs-gcp-5ddf1c5112c2133e

Project ID
qwiklabs-gcp-5ddf1c5112c2133e

Project number
816613791341

Go to project settings

Resources

Compute Engine
1 instance

Cloud Storage
1 bucket

Cloud SQL
1 instance

Compute Engine

CPU (%)

● instance/cpu/utilization: 0.172

Go to the Compute Engine dashboard

SQL

Storage used (bytes)

Fetch the application source files

The lab setup includes automated deployment of the services that you configured yourself in previous labs. When the setup is complete, copies of the demo application (configured so that they are ready for this lab session) are put into a Cloud Storage bucket named using the project ID for this lab.

Before you proceed with the tasks for this lab, you must first copy the demo application into Cloud Shell so you can continue to work on it.

1. In the upper-right corner of the screen, click **Activate Cloud Shell** ( ) to open Cloud Shell.
2. Click **Start Cloud Shell**.

Boost mode is not needed for this lab.

3. In the Cloud Shell command line, enter the following command to create an environment variable that contains the project ID for this lab:

```
export PROJECT_ID=$(gcloud config list --format 'value(core.project)')
```

4. Verify that the demo application files were created.

```
gsutil ls gs://$PROJECT_ID
```

Repeat the last step if the command reports an error or if it does not list the two folders for the `guestbook-frontend` application and the `guestbook-service` backend application and a folder that contains sample Kubernetes deployment files.

Note:

A Cloud Storage bucket that is named using the project ID for this lab is automatically created for you by the lab setup. The source code for your applications is copied into this bucket when the Cloud Spanner instance that is used for the service application in this lab is ready. Compared to the preceding labs in this course the startup process for this is relatively quick so you might not have to wait here for it to complete.

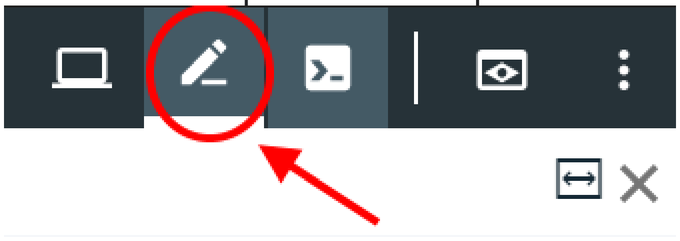
5. Copy the application folders to Cloud Shell.

```
gsutil -m cp -r gs://$PROJECT_ID/* ~/
```

6. Make the Maven wrapper scripts executable.

```
chmod +x ~/guestbook-frontend/mvnw
chmod +x ~/guestbook-service/mvnw
```

7. Click the pencil icon to open the Cloud Shell code editor.



Task 1. Create a Kubernetes Engine cluster

In this task, you create a Kubernetes Engine cluster. You use the Kubernetes Engine cluster to run your containerized application later in the lab.

1. In the Cloud Shell enable Kubernetes Engine API.

```
gcloud services enable container.googleapis.com
```

2. Create a Kubernetes Engine cluster that has Stackdriver Logging and Monitoring enabled.

```
gcloud container clusters create guestbook-cluster \
    --zone=us-central1-a \
    --num-nodes=2 \
    --machine-type=n1-standard-2 \
    --enable-autorepair \
    --enable-cloud-monitoring \
    --enable-cloud-logging
```

Because this operation takes a few minutes, you can go to the next task while the cluster is created in the background.

Task 2. Containerize the applications

In this task, you add the Jib plugin to the Maven `pom.xml` file for each of the applications and configure them to use the Google Container Registry (`gcr.io`) as your container registry. Jib is a Maven plugin that enables you to containerize your application by building Docker and OCI images. You use Maven to build each application as a container.

1. In a new Cloud Shell tab enable Container Registry API.

```
gcloud services enable containerregistry.googleapis.com
```

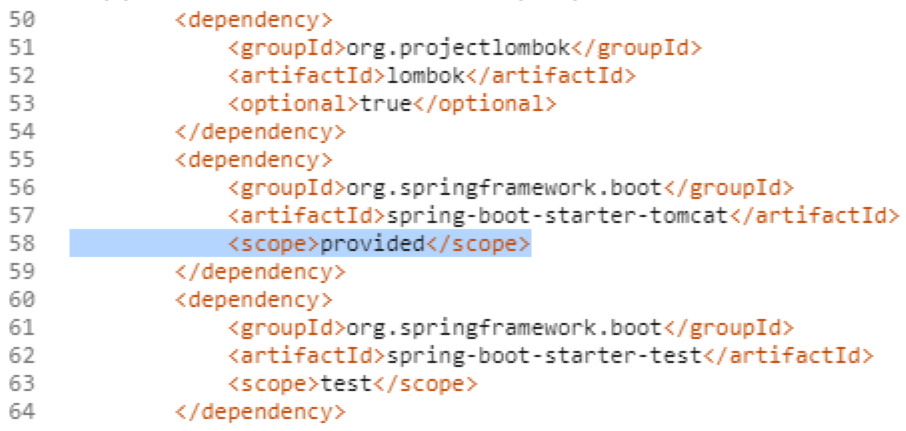
2. Run the following command to display the project ID for the current project:

```
gcloud config list --format 'value(core.project)'
```

3. Make a note of the project ID.

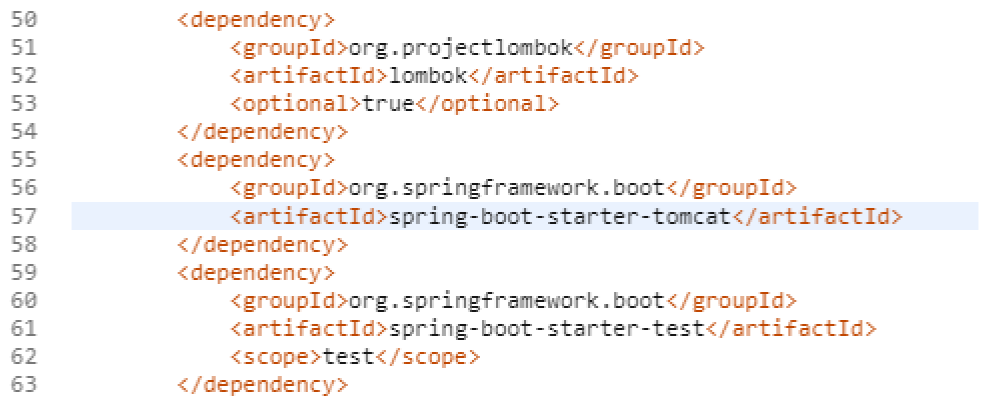
In a number of later steps, you replace `[PROJECT_ID]` placeholders with this project ID.

4. In the Cloud Shell code editor, open `~/guestbook-frontend/pom.xml`.
5. Remove the `provided` scope from the Tomcat dependency the frontend application's `pom.xml` file, highlighted here.



```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

When complete this section looks like the following:



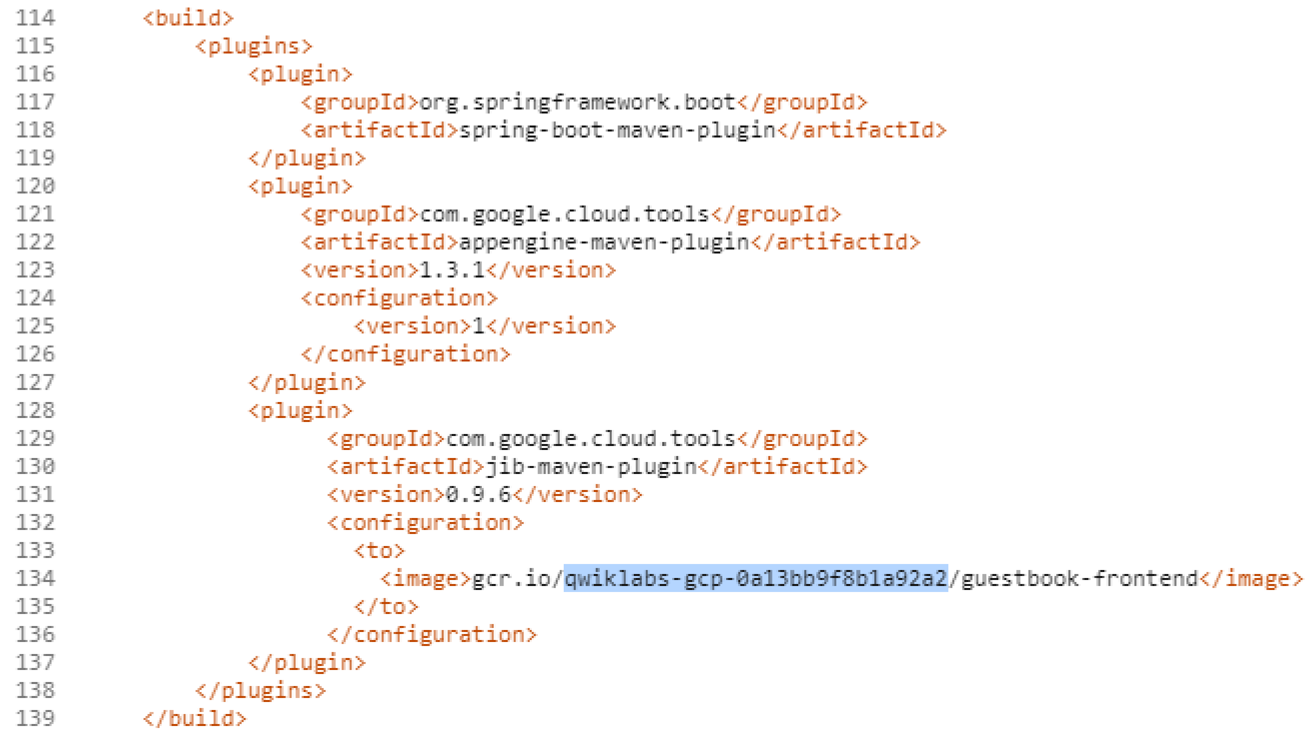
```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

6. Insert a new plugin definition for the Jib Maven plugin into the `<plugins>` section inside the `<build>` section near the end of the file, immediately before the closing `</plugins>` tag.

```
            <plugin>
                  <groupId>com.google.cloud.tools</groupId>
                  <artifactId>jib-maven-plugin</artifactId>
                  <version>0.9.6</version>
                  <configuration>
                          <to>
                      <image>gcr.io/[PROJECT_ID]/guestbook-frontend</image>
                    </to>
                  </configuration>
            </plugin>
```

Warning
You must replace the placeholder for [PROJECT_ID] here with your project ID so that the build section looks similar to the screenshot below. The actual project ID for your lab will be slightly different.

This configures the image name for the guestbook frontend application on Google Container Registry.



```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>com.google.cloud.tools</groupId>
                <artifactId>appengine-maven-plugin</artifactId>
                <version>1.3.1</version>
                <configuration>
                    <version>1</version>
                </configuration>
            </plugin>
            <plugin>
                  <groupId>com.google.cloud.tools</groupId>
                  <artifactId>jib-maven-plugin</artifactId>
                  <version>0.9.6</version>
                  <configuration>
                    <to>
                      <image>gcr.io/qwiklabs-gcp-0a13bb9f8b1a92a2/guestbook-frontend</image>
                    </to>
                  </configuration>
            </plugin>
        </plugins>
    </build>
```

7. In the Cloud Shell change to the frontend application directory.

```
cd ~/guestbook-frontend
```

8. Use Maven to build the frontend application container using the Jib plugin.

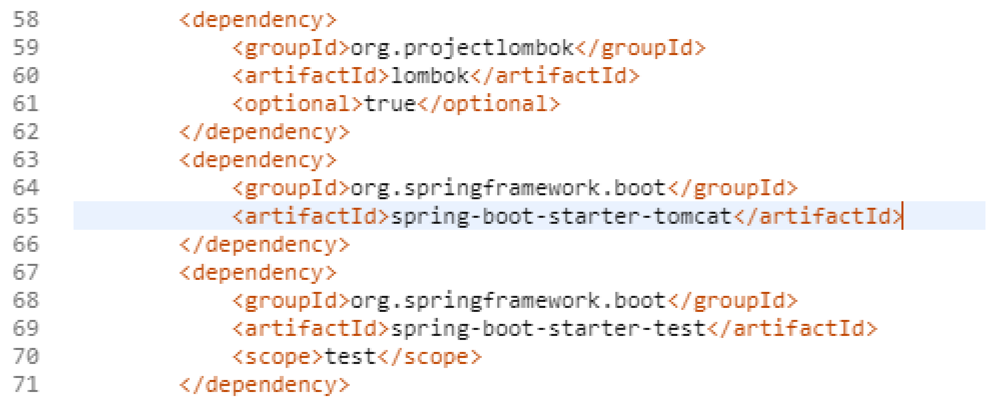
```
./mvnw clean compile jib:build
```

When the build completes, it reports success and the location of the container image in the Google `gcr.io` container registry.

```
...
[INFO] Built and pushed image as gcr.io/next18-bootcamp-test/spring-cloud-gcp-guestbook-frontend
[INFO]
[INFO] ------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------
[INFO] Total time: 43.730 s
[INFO] Finished at: 2018-07-16T16:07:34-04:00
[INFO] ------------------------------------------------
```

9. In the Cloud Shell code editor, open `~/guestbook-service/pom.xml`.
10. Remove the `provided` scope from the Tomcat dependency of the guestbook backend service application's `pom.xml` file.

The dependency section should now look like the following:



```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

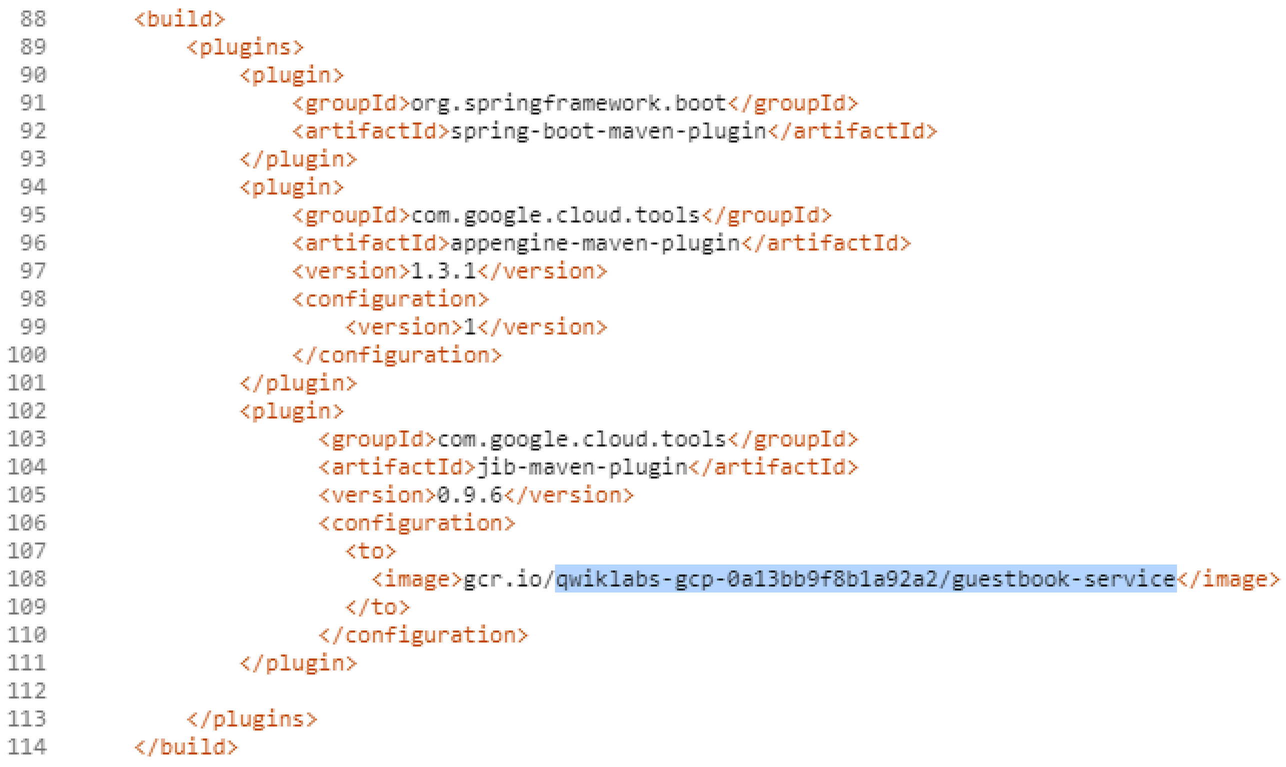
11. Insert a new plugin definition for the Jib Maven plugin into the `<plugins>` section inside the `<build>` section near the end of the file, immediately before the closing `</plugins>` tag.

```
            <plugin>
                  <groupId>com.google.cloud.tools</groupId>
                  <artifactId>jib-maven-plugin</artifactId>
                  <version>0.9.6</version>
                  <configuration>
                          <to>
                      <image>gcr.io/[PROJECT_ID]/guestbook-service</image>
                    </to>
                  </configuration>
            </plugin>
```

Warning

You must replace the placeholder for [PROJECT_ID] here with your project ID so that the build section looks similar to the screenshot below. The actual project ID for your lab will be slightly different.

This configures the image name for the guestbook backend service application on Google Container Registry and is different to the name used for the frontend application previously.



```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>com.google.cloud.tools</groupId>
                <artifactId>appengine-maven-plugin</artifactId>
                <version>1.3.1</version>
                <configuration>
                    <version>1</version>
                </configuration>
            </plugin>
            <plugin>
                  <groupId>com.google.cloud.tools</groupId>
                  <artifactId>jib-maven-plugin</artifactId>
                  <version>0.9.6</version>
                  <configuration>
                    <to>
                      <image>gcr.io/qwiklabs-gcp-0a13bb9f8b1a92a2/guestbook-service</image>
                    </to>
                  </configuration>
            </plugin>

        </plugins>
    </build>
```

12. In the Cloud Shell change to the guestbook backend service application directory.

```
cd ~/guestbook-service
```

13. Use Maven to build the build the backend service application container using the Jib plugin.

```
./mvnw clean compile jib:build
```

When the build completes, it reports success and the location of the container image for the backend service.

```
[INFO] Built and pushed image as gcr.io/qwiklabs-gcp-
0a13bb9f8b1a92a2/guestbook-service
[INFO]
[INFO] ------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------
[INFO] Total time: 35.766 s
[INFO] Finished at: 2018-12-09T13:41:02Z
[INFO] ------------------------------------------------
```

Task 3. Set up a service account

In this task, you create a service account with permissions to access your GCP services. You then store the service account that you generated earlier in Kubernetes as a secret so that it is accessible from the containers.

1. In Cloud Shell enter the following commands to create a service account specific to the guestbook application.

```
gcloud iam service-accounts create guestbook
```

2. Add the Editor role for your project to this service account.

```
export PROJECT_ID=$(gcloud config list --format 'value(core.project)')
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member serviceAccount:guestbook@${PROJECT_ID}.iam.gserviceaccount.com \
  --role roles/editor
```

Warning
This action creates a service account with the Editor role. In your production environment, you should assign only the roles and permissions that the application needs.

3. Generate the JSON key file to be used by the application to identify itself using the service account.

```
gcloud iam service-accounts keys create \
    ~/service-account.json \
    --iam-account guestbook@${PROJECT_ID}.iam.gserviceaccount.com
```

This command creates service account credentials that are stored in the `$HOME/service-account.json` file.

Warning
Treat the `service-account.json` file as your own username/password. Do not share this information.

4. Check the Kubernetes server version to verify that the Kubernetes Engine cluster you deployed in an earlier task has been created.

```
kubectl version
```

The output should contain version information similar to the following:

```
Client Version: version.Info{Major:"1", Minor:"10"...
Server Version: version.Info{Major:"1", Minor:"11+"...
```

5. Create the secret using the service account credential file.

```
kubectl create secret generic guestbook-service-account \
  --from-file=$HOME/service-account.json
```

6. Verify that the service account is stored.

```
kubectl describe secret guestbook-service-account
```

The output should be similar to the following:

```
Name:         guestbook-service-account
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
service-account.json:  ... bytes
```

Task 4. Deploy the containers

In this task, you deploy the two containers containing the guestbook frontend application and the guestbook backend service application to your Kubernetes Engine cluster.

1. In the Cloud Shell code editor, open `~/kubernetes/guestbook-frontend-deployment.yaml`.

Note

A basic Kubernetes deployment file has been created for you for each of your applications. These are a standard feature used to configure containerized application deployments for Kubernetes but the full detail is out of scope for this course. For this lab you will only update the guestbook Kubernetes deployment files to use the images that you created.

2. Replace the line `image: saturnism/spring-gcp-guestbook-frontend:latest` with the line `image: gcr.io/[PROJECT_ID]/guestbook-frontend` below the line specifying the container name.

Note

You must replace `[PROJECT_ID]` with the project ID that you recorded in an earlier task. Spaces are significant in YAML files so make sure your new line matches the indentation of the line it replaces exactly.

3. In the Cloud Shell code editor, open `~/kubernetes/guestbook-service-deployment.yaml`.

You update the guestbook frontend Kubernetes deployment files to use the image that you created.

4. Replace the line `image: saturnism/spring-gcp-guestbook-frontend:latest` with the line `image: gcr.io/[PROJECT_ID]/guestbook-service` below the line specifying the container name.

Note

You must replace `[PROJECT_ID]` with the project ID that you recorded in an earlier task.

5. Switch back to the Cloud Shell and deploy the updated Kubernetes deployments.

```
kubectl apply -f ~/kubernetes/
```

The Kubernetes configuration for your guestbook frontend application is configured to deploy an external load balancer. The configuration used in the sample deployment generates a load balanced external IP address for the frontend application

6. Check the status of the frontend application deployment.

```
kubectl get svc guestbook-frontend
```

7. Repeat the command every minute or so until the external IP address is listed.

```
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
AGE
guestbook-frontend   LoadBalancer   ...            23.251.156.216   ...
...
```

8. Check the status of all of the services running on your Kubernetes Engine cluster.

```
kubectl get svc
```

You see that only the frontend application has an external ip address.

9. Open a browser and navigate to the application at **http://[EXTERNAL_IP]:8080.**

10. Post a message to test the functionality of the application running on Kubernetes Engine.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.
You'll be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.
The number of stars indicates your rating:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, use the **Support** tab.

v