JAVAMS12 Deploying to GKE

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 the 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 Google Kubernetes Engine (GKE).

GKE 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. GKE services, support, and tools are widely available.

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

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

Objectives

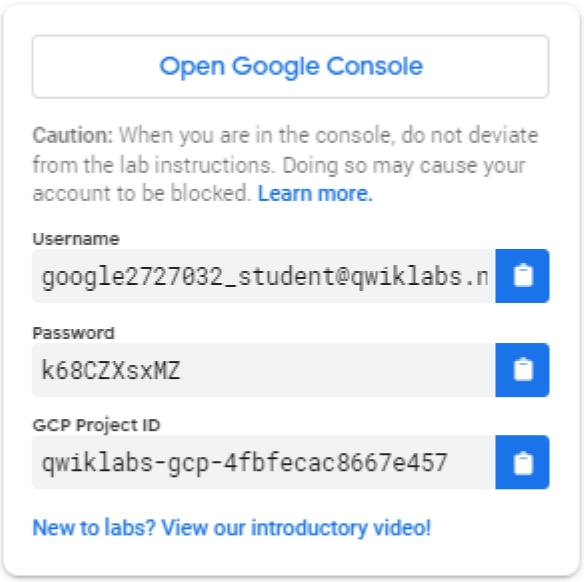
In this lab, you learn how to perform the following tasks:

- Create a GKE cluster
- Create a containerized version of a Java application
- Create a GKE deployment for a containerized application

Setup and requirements

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.



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

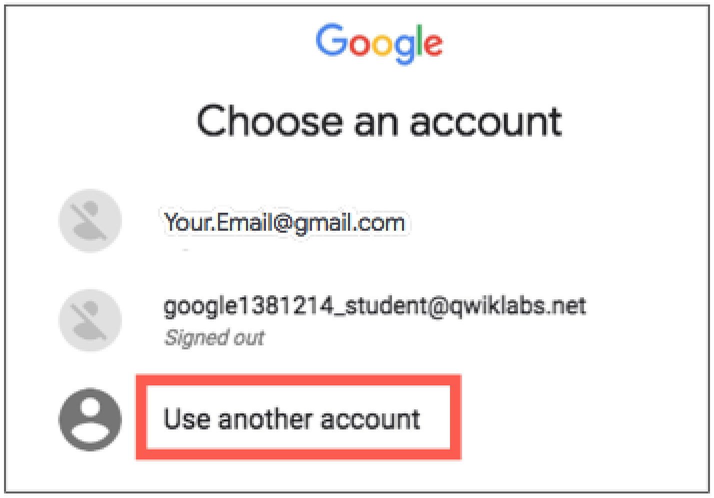
GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

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.

   Note: Open the tabs in separate windows, side-by-side.

3. On the Choose an account page, click **Use Another Account**. The Sign in page opens.



4. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

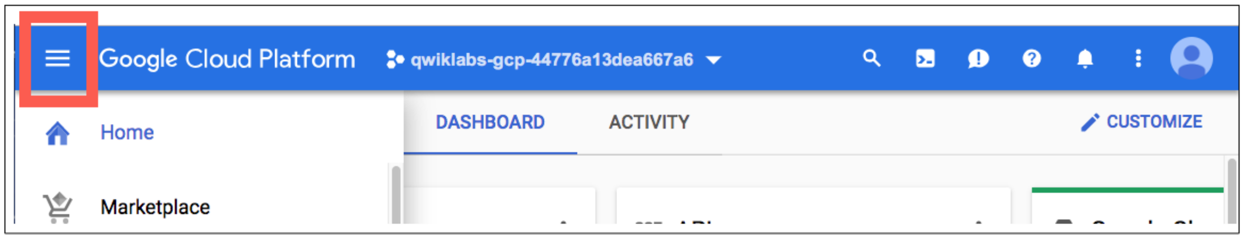
Note: You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

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 Cloud console opens in this tab.

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



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

Task 1. Fetch the application source files

In this task you clone the source repository files that are used throughout this lab.

1. To begin the lab, click the **Activate Cloud Shell** button at the top of the Google Cloud Console.
2. To activate the code editor, click the `Open Editor` button on the toolbar of the Cloud Shell window. This sets up the editor in a new tab with continued access to Cloud Shell.

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.

3. In Cloud Shell, 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)')
```
Copied!
content_copy

4. Verify that the demo application files were created:

```
gsutil ls gs://$PROJECT_ID
```
Copied!
content_copy
Note: Re-run the above **demo application** command until the files appear in the output. It will take around 5- 10 minutes.

5. Copy the application folders to Cloud Shell:

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

6. Make the Maven wrapper scripts executable:

```
chmod +x ~/guestbook-frontend/mvnw
chmod +x ~/guestbook-service/mvnw
```
Copied!
content_copy

Task 2. Create a GKE cluster

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

1. In Cloud Shell, enable the Kubernetes Engine API:

```
gcloud services enable container.googleapis.com
```
Copied!
content_copy

2. Create a GKE cluster that has Cloud Logging and Monitoring enabled. Because this operation takes a few minutes, you can go to the next task while the cluster is created in the background:

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

Copied!
content_copy

3. Check the GKE server version to verify that the GKE cluster you deployed has been created:

```
kubectl version
```

Copied!
content_copy
The output should contain version information similar to the following:

```
Client Version: version.Info{Major:"1", Minor:"18"...
Server Version: version.Info{Major:"1", Minor:"14+"...
```

Task 3. 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 the Container Registry API:

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

Copied!
content_copy

2. Run the following command to set and display the `PROJECT_ID` environment variable:

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

Copied!
content_copy

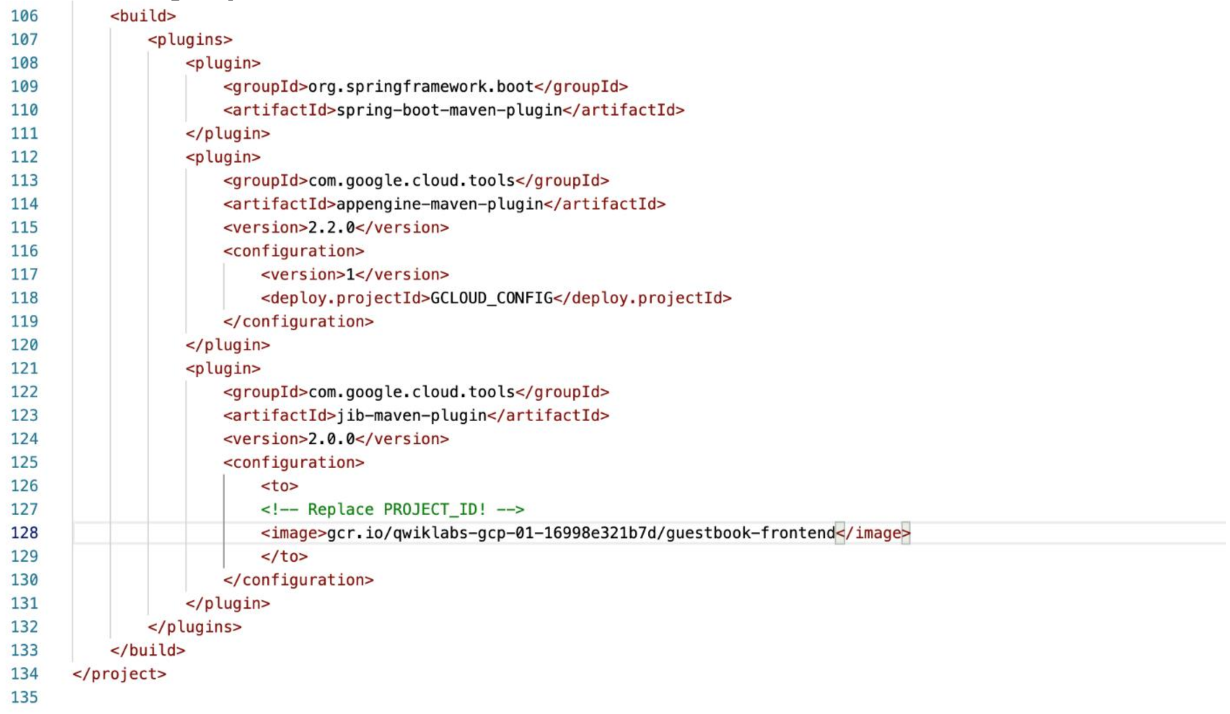
3. Make a note of this project ID. In a number of later steps, you replace `[PROJECT_ID]` placeholders with this project ID.

4. In the Cloud Shell code editor, open `~/guestbook-frontend/pom.xml`.

5. 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>2.4.0</version>
    <configuration>
        <to>
        <!-- Replace [PROJECT_ID]! -->
        <image>gcr.io/[PROJECT_ID]/guestbook-frontend</image>
        </to>
    </configuration>
</plugin>
```

Copied!
content_copy
Note: 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>2.2.0</version>
                <configuration>
                    <version>1</version>
                    <deploy.projectId>GCLOUD_CONFIG</deploy.projectId>
                </configuration>
            </plugin>
            <plugin>
                <groupId>com.google.cloud.tools</groupId>
                <artifactId>jib-maven-plugin</artifactId>
                <version>2.0.0</version>
                <configuration>
                    <to>
                    <!-- Replace PROJECT_ID! -->
                    <image>gcr.io/qwiklabs-gcp-01-16998e321b7d/guestbook-frontend</image>
                    </to>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

6. In Cloud Shell, change to the frontend application directory:

```
cd ~/guestbook-frontend
```
Copied!
content_copy

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

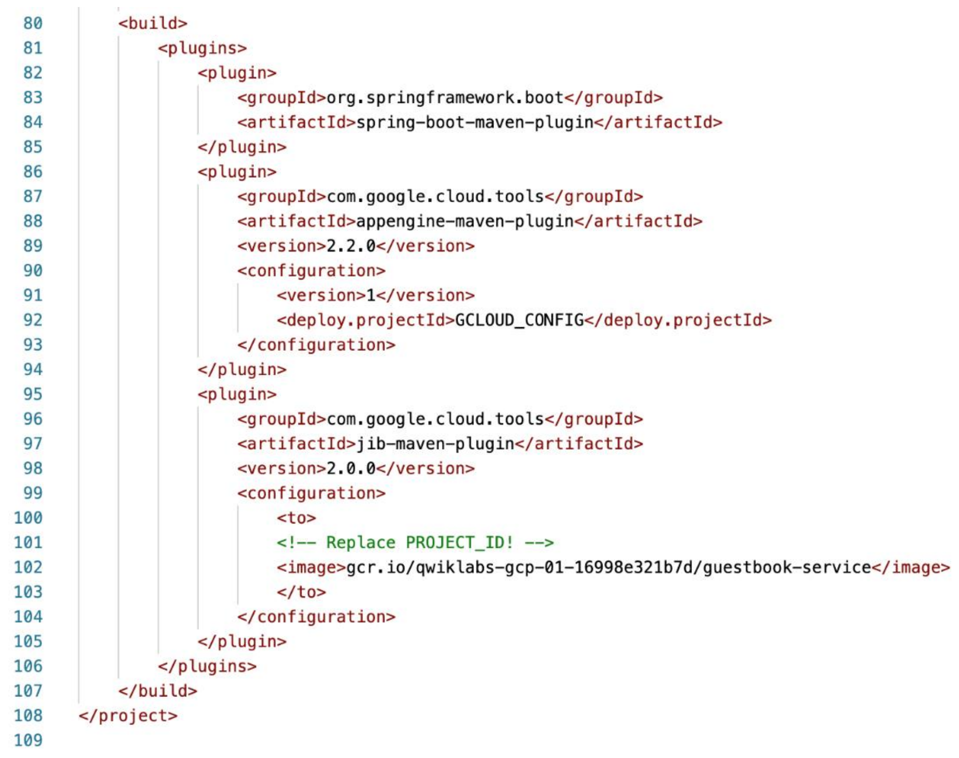
```
./mvnw clean compile jib:build
```
Copied!
content_copy
When the build completes, it reports success and the location of the container image in the Google `gcr.io` container registry.

```
...
[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] ------------------------------------------------
```

8. In the Cloud Shell code editor, open `~/guestbook-service/pom.xml`.

9. 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>2.4.0</version>
                <configuration>
                    <to>
                    <!-- Replace [PROJECT_ID]! -->
                    <image>gcr.io/[PROJECT_ID]/guestbook-service</image>
                    </to>
                </configuration>
            </plugin>
```
Copied!
content_copy
Note: 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>2.2.0</version>
                <configuration>
                    <version>1</version>
                    <deploy.projectId>GCLOUD_CONFIG</deploy.projectId>
                </configuration>
            </plugin>
            <plugin>
                <groupId>com.google.cloud.tools</groupId>
                <artifactId>jib-maven-plugin</artifactId>
                <version>2.0.0</version>
                <configuration>
                    <to>
                    <!-- Replace PROJECT_ID! -->
                    <image>gcr.io/qwiklabs-gcp-01-16998e321b7d/guestbook-service</image>
                    </to>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

10. In Cloud Shell, change to the guestbook backend service application directory:

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

Copied!
content_copy

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

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

Copied!
content_copy
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 4. Set up a service account

In this task, you create a service account with permissions to access your Google Cloud services. You then store the service account that you generated earlier in GKE 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
```

Copied!
content_copy

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

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member serviceAccount:guestbook@${PROJECT_ID}.iam.gserviceaccount.com
\
  --role roles/editor
```

Copied!
content_copy
Note: 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
```

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

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

4. Create the secret using the service account credential file:

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

Copied!
content_copy

5. Verify that the service account is stored:

```
kubectl describe secret guestbook-service-account
```
Copied!
content_copy
Note: In production systems on GKE, you should review the Use Workload Identity guide to securely provision and provide credentials to the GKE cluster.
The output should be similar to the following:

```
Name:         guestbook-service-account
Namespace:    default
Labels:
Annotations:
Type:  Opaque
Data
====
service-account.json:  ... bytes
```

Task 5. Deploy the containers

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

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

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

4. Replace the line `image: saturnism/spring-gcp-guestbook-service:latest` with the line `image: gcr.io/[PROJECT_ID]/guestbook-service:latest` 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 Cloud Shell and deploy the updated GKE deployments:

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

Copied!
content_copy
The GKE 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 to see that all pods are up and running. You can use `CTRL + C` to terminate the process:

```
 watch kubectl get pods
```

Copied!
content_copy

7. Guestbook Frontend is configured to deploy an external Load Balancer. It'll generate an external IP address that does L4 Load Balancing to your backend. Check and wait until the external IP is populated:

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

Copied!
content_copy
You can 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 GKE cluster:

```
kubectl get svc
```

Copied!
content_copy
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 GKE.

Task 6. Review

In this lab you created a GKE cluster. You also created a containerized version of a Java application. Finally, you created a GKE deployment for a containerized application.