# Chapter 4 - Cloud Development

Lab 1:

# App Dev: Setting up a Development Environment - Java

50 minutesFree

Rate Lab

## GSP166



## Objectives

In this lab, you set up a Java development environment on Google Cloud. You use Compute Engine to create a virtual machine (VM) and installing software libraries for software development.

You perform the following tasks:

* Provision a Compute Engine instance.
* Connect to the instance using SSH.
* Install a Java library on the instance.
* Verify the software installation.

## Overview

Compute Engine is just one resource provided on Google Cloud.

### **Google Cloud**

Google Cloud consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as virtual machines (VMs), that are contained in Google's data centers around the globe. Each data center location is in a global region. Regions include Central US, Western Europe, and East Asia. Each region is a collection of zones, which are isolated from each other within the region. Each zone is identified by a name that combines a letter identifier with the name of the region. For example, zone a in the East Asia region is named asia-east1-a.

This distribution of resources provides several benefits, including redundancy in case of failure and reduced latency by locating resources closer to clients. This distribution also introduces some rules about how resources can be used together.

### **Projects**

Any Google Cloud resources that you allocate and use must belong to a project. You can think of a project as the organizing entity for what you're building. A project is made up of the settings, permissions, and other metadata that describe your applications. Resources within a single project can work together easily, for example by communicating through an internal network, subject to the regions-and-zones rules. The resources that each project contains remain separate across project boundaries; you can only interconnect them through an external network connection.

Each Google Cloud project has:

A project name, which you provide. A project ID, which you can provide or Google Cloud can provide for you. A project number, which Google Cloud provides. As you work with Google Cloud, you'll use these identifiers in certain command lines and API calls. The following screenshot shows a project name, its ID, and number:

The Cloud Console displays project ID and name

In this example:

Example Project is the project name. example-id is the project ID. 123456789012 is the project number. Each project ID is unique across Google Cloud. Once you have created a project, you can delete the project but its ID can never be used again.

When billing is enabled, each project is associated with one billing account. Multiple projects can have their resource usage billed to the same account.

A project serves as a namespace. This means every resource within each project must have a unique name, but you can usually reuse resource names if they are in separate projects. Some resource names must be globally unique. Refer to the documentation for the resource for details.

In this lab, you provision a Compute Engine virtual machine (VM) and install software libraries for Java software development on Google Cloud.

### **Ways to interact with the services**

Google Cloud gives you three basic ways to interact with the services and resources.

* Cloud Console: a web-based, graphical user interface that you can use to manage your Google Cloud projects and resources.
* Command-line interface
  + Cloud SDK: provides the gcloud command-line tool, which gives you access to the commands you need.
  + Cloud Shell: a browser-based, interactive shell environment for Google Cloud. You can access Cloud Shell from the Google Cloud console. If you prefer to work in a terminal window, the Cloud SDK provides the gcloud command-line tool, which gives you access to the commands you need. The gcloud tool can be used to manage both your development workflow and your Google Cloud resources. See the gcloud reference for the complete list of available commands.
* Client libraries: The Cloud SDK includes client libraries that enable you to easily create and manage resources. Google Cloud client libraries expose APIs to provide access to services and resource management functions. You also can use the Google API client libraries to access APIs for products such as Google Maps, Google Drive, and YouTube.

## Setup and Requirements

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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.

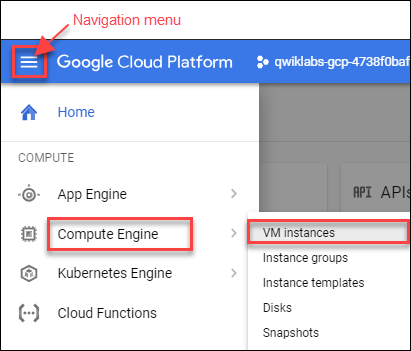
**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

## Create a Compute Engine Virtual Machine Instance

In this section, you use the Google Cloud Console to provision a new Compute Engine virtual machine (VM) instance.

### **Create and connect to a VM**

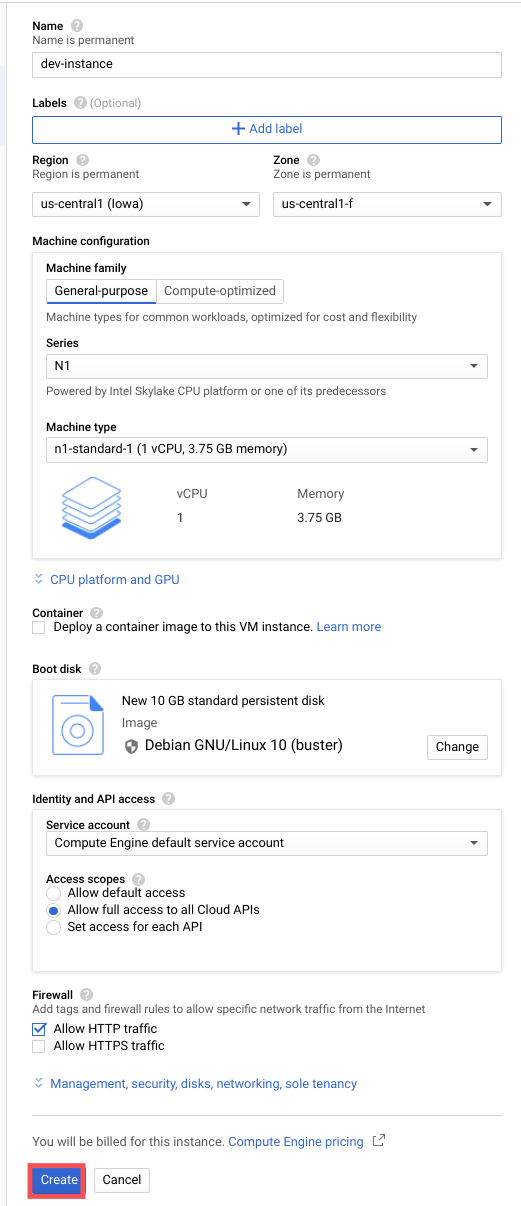
1. In the Cloud Console, Click **Navigation menu** > **Compute Engine** > **VM Instances**.



1. On the **VM Instances** page, click **Create.**
2. On the **Create an instance** page, for **Name** type **dev-instance**, and select the **us-central1-f** zone.

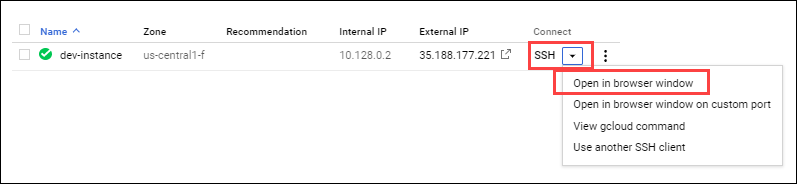
**Regions and zones:** Google Cloud offers products and services in multiple distinct geographic locations, called regions. Each region has multiple distinct zones. Each zone is isolated from other zones in terms of power and internet connectivity.

1. In the **Boot disk**, click on the **Change** button. Set the **Version** to **Debian GNU/Linux 10 (buster)**, and then click on **Select**.
2. In the **Identity and API access**, **Access Scopes** section, select **Allow full access to all Cloud APIs**.
3. In the **Firewall** section, enable **Allow HTTP traffic**.
4. Leave the remaining settings as their defaults, and click **Create**.



It takes about 20 seconds for the virtual machine to be provisioned and started.

1. On the **VM instances** page, in the row for the **dev-instance**, click **SSH** (in the **Connect** column).



This launches a browser-hosted SSH session. If you have a popup blocker, you may need to click twice. There's no need to configure or manage SSH keys.

#### Test Completed Task

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Create a Compute Engine Virtual Machine Instance (zone: us-central1-f)

Check my progress

### **Install software and configure the VM instance**

1. In the SSH session, to update the Debian package list, enter the following command:

sudo apt-get updatecontent\_copy

1. Install Git:

sudo apt-get install git -ycontent\_copy

1. Install Maven:

sudo apt-get install -yq mavencontent\_copy

1. Configure IP tables:

sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080content\_copy

This command (above) to configure the IP tables redirects requests on Port 80 to Port 8080 - the Java Web application listens on Port 8080.

1. Export the Project ID as an environment variable. Replace [PROJECT\_ID] with the Project ID located under the credentials provided in your lab:

export GCLOUD\_PROJECT=[PROJECT\_ID]content\_copy

#### Test Completed Task

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Install software and configure the VM instance

Check my progress

## Configure the VM to Run Application Software

In this section, you verify the software installation and run some sample code.

### **Verify Java installation**

1. To check the version of Java, enter the following command:

java -versioncontent\_copy

You should see the Java version number for version 11.

1. Clone the class repository:

git clone https://github.com/GoogleCloudPlatform/training-data-analystcontent\_copy

1. Change the working directory:

cd ~/training-data-analyst/courses/developingapps/java/devenv/content\_copy

1. Run a simple web application:

mvn clean installcontent\_copy

Wait for the project to build. When the project successfully finishes you will see output similar to this:

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 25.913 s

[INFO] Finished at: 2018-05-28T00:26:23+00:00

[INFO] Final Memory: 34M/82M

[INFO] ------------------------------------------------------------------------

content\_copy

1. Run the application.

mvn spring-boot:runcontent\_copy

The project is running when you see output similar to the following:

01:11:05.274 [restartedMain] INFO c.g.training.appdev.DemoApplication - Started DemoApplication in 1 (JVM running for 9.863)

content\_copy

1. Return to the Cloud Console VM instances list, and click on the **External IP** address for the **dev-instance**.



A browser opens to display a Hello GCP dev! message from Java.

#### Test Completed Task

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Run application software to get success response

Check my progress

1. Return to the SSH window, and stop the application by pressing **Ctrl+C**.
2. To run a simple Java application that lists Compute Engine instances, execute the following command:

mvn exec:java@list-gcecontent\_copy

Many details about your VM should appear in the terminal window.

## Test your Understanding

Below are multiple-choice questions to reinforce your understanding of this lab's concepts. Answer them to the best of your abilities.

Which one of the following protocol-port combination is responsible for allowing HTTP traffic?



http:80



icmp



tcp:443



tcp:80

Submit

Lab 2:

# App Dev - Storing Application Data in Cloud Datastore - Java

1 hourFree

Rate Lab

## GSP167



## Overview

[Google Cloud Datastore](https://cloud.google.com/datastore/docs/concepts/overview) is a NoSQL document database built for automatic scaling, high performance, and ease of application development. In this lab, you use Datastore to store application data for an online Quiz application. You also configure the application to retrieve from Datastore and display the data in the quiz.

The Quiz application skeleton has already been written. You clone the repository that contains the skeleton using Google Cloud Shell, review the code using the Cloud Shell editor, and view it using the Cloud Shell web preview feature. You then modify the code that stores data to use Cloud Datastore.

## Objectives

In this lab, you perform the following tasks:

* Harness Cloud Shell as your development environment
* Preview the application
* Update the application code to integrate Cloud Datastore

## Setup and Requirements

### **Qwiklabs setup**

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Prepare the Quiz Application

In this section, you access Cloud Shell, clone the git repository containing the Quiz application, and run the application.

### **Clone source code in Cloud Shell**

In Cloud Shell command line, enter the following command to clone the repository for the class:

git clone https://github.com/GoogleCloudPlatform/training-data-analyst

content\_copy

### **Configure and run the Quiz application**

1. To navigate to the working directory, enter the following command:
2. cd ~/training-data-analyst/courses/developingapps/java/datastore/start

content\_copy

1. Create the environment variable GCLOUD\_PROJECT that references the Project ID:
2. export GCLOUD\_PROJECT=$DEVSHELL\_PROJECT\_ID

content\_copy

1. Install the application dependencies:
2. mvn clean install

content\_copy

The installation may take a couple of minutes. It's complete when you see output similar to the following:

Example output:

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 38.805 s

[INFO] Finished at: 2018-05-27T22:40:49-04:00

[INFO] Final Memory: 35M/84M

[INFO] ------------------------------------------------------------------------

content\_copy

1. Run the application:

mvn spring-boot:runcontent\_copy

The application is running When you see output similar to the following.

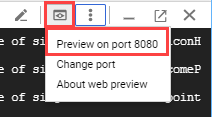
01:39:26.404 [restartedMain] INFO c.g.training.appdev.QuizApplication - Started QuizApplication in 12.628 seconds (JVM running for 14.1

57)

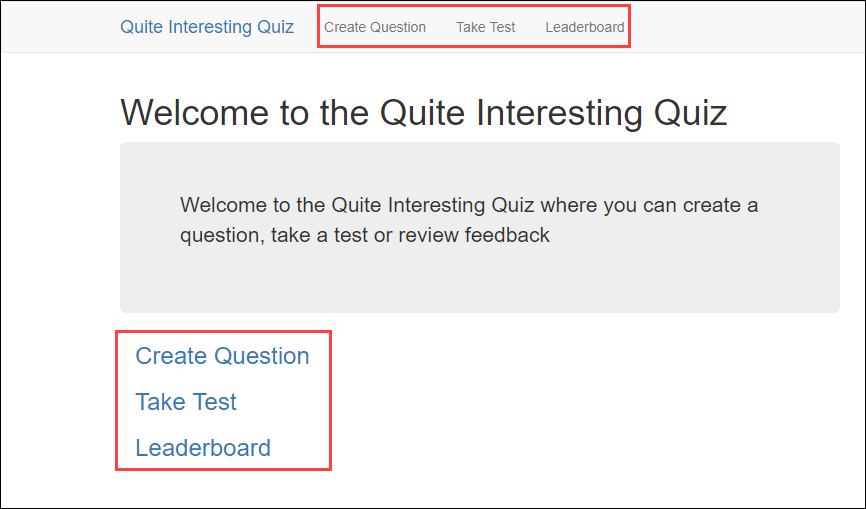
content\_copy

### **Review the Quiz application**

1. In Cloud Shell, click **Web preview > Preview on port 8080** to preview the quiz application.



You should see the user interface for the web application. The three main parts to the application are **Create Question**, **Take Test**, and **Leaderboard**. Links to each are shown in the top navigation bar and on the page.



1. Click **Create Question**.

You should see a simple form that contains textboxes for the question and answers with radio buttons to select the correct answer. Quiz authors can add questions in this part of the application.

This part of the application is written as a server-side web application using the popular Java web application framework Spring Boot.

1. Click **Take Test**.

You are taken to a client application. Since you haven't made any questions, the Quite Interesting Quiz is blank.

1. In the top navigation bar, click **GCP**.

You should see a Dummy Title and Dummy Answers. The Dummy Title represents a sample question.

Quiz takers will answer questions in this part of the application.

This part of the application is written as a client-side web application using the popular JavaScript framework AngularJS. It receives JSON data from the server and uses JavaScript in the browser to display questions and collect answers.

To return to the server-side application, click on the **Quite Interesting Quiz** link in the navigation bar.

## Examine the Quiz Application Code

In this section and throughout the lab you'll review the Quiz application code in a code editor. You can use the shell editors that are installed on Cloud Shell, such as nano or vim, or use the Cloud Shell code editor. This lab uses the Cloud Shell code editor.

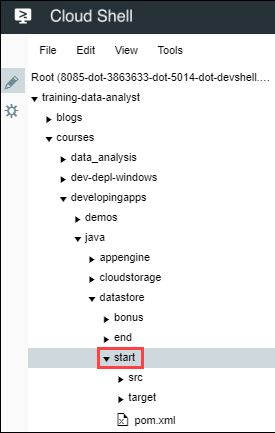
### **Launch the Cloud Shell text editor**

From Cloud Shell, click on the **Open Editor** icon.

### **Review the code**

The application is a standard Java application written using the popular Spring Boot application framework.

1. Navigate to the /training-data-analyst/courses/developingapps/java/datastore/start folder using the file browser panel on the left side of the editor.



This is the root folder for the application.

In the datastore folder, notice the end folder. The end folder contains the same files as the start folder, but each file in the end folder contains the complete code required to perform this lab.

1. From the start folder, navigate to the src/main/java/com/google/training/appdev folder using the file browser panel on the left side of the editor.

The paths for Java source code files are relative to the appdev folder.

### **Review the Spring Boot Web application**

1. Select the .../QuizApplication.java file.

In this file, the class contains the entrypoint for the Spring Boot application.

1. Select the .../services/gcp/domain/Question.java file.

In this file, the domain class represents question data submitted in the question form and questions displayed when taking a quiz.

1. Select the .../web/QuestionsController.java file.

This file contains the handlers that display the form and collect form data posted by quiz authors in the web application.

In the QuestionsController.java file, find the handler that responds to HTTP POST requests for the /questions/add route.

Notice that the controller delegates the implementation of the handler to a service, questionService.

1. Navigate to the /training-data-analyst/courses/developingapps/java/datastore/start/src/main/resources folder using the file browser panel on the left side of the editor.

This folder contains templates for the web application user interface and static content displayed in the client-side web application.

1. Select the templates folder.

This folder contains the template for the web application user interface using the Thyme templating engine.

1. Select the **.../templates/new\_question.html** file.

This file contains the template for the Create Question form. Notice how there is a select list to pick a quiz, textboxes where an author can enter the question and answers, and radio buttons to select the correct answer.

Return to the folder containing Java source code using the file browser panel on the left side of the editor. (Do you remember? It's start/src/main/java/com/google/training/appdev.)

1. Select the .../api/QuizEndpoint.java file.

This file contains the handler that sends JSON data to students taking a test. Notice that the handlers also make use of the questionService object.

1. Select the .../services/gcp/datastore/QuestionService.java \_\_file.

This is the file where you write Datastore code to save and load quiz questions to and from Cloud Datastore. The web application and API use this class.

## Add Entities to Cloud Datastore

In this section, you write code to save form data in Cloud Datastore.

Important: Update code within the sections marked as follows: `// TODO` `// END TODO` To maximize your learning, review the code, inline comments, and related API documentation.For more information see [Google Cloud Datastore Documentation](https://cloud.google.com/datastore/docs/)

### **Create an App Engine application to provision Cloud Datastore**

From Cloud Shell, click on the **Open Terminal** icon.

Stop the application by pressing **Ctrl**+**C**.

To create an App Engine application in your project, execute the following command in Cloud Shell:

gcloud app create --region "us-central"content\_copy

Note: Although You aren't yet using App Engine for your web application, Cloud Datastore requires you to create an App Engine application in your project.

Click Check my progress to verify the objective.

Create an App Engine application

Check my progress

### **Import and use the Java Datastore package**

From Cloud Shell, click on the **Open Editor** icon.

1. Open the .../services/gcp/datastore/QuestionService.java file in the Cloud Shell editor.
2. Write a star import for the com.google.cloud.datastore.\* package.
3. // TODO: Import the com.google.cloud.datastore.\* package
4. import com.google.cloud.datastore.\*;
5. // END TODO

content\_copy

1. Declare a Datastore client object named datastore and initialize it.
2. // TODO: Create a Datastore client object, datastore
3. // The DatastoreOptions class has a getDefaultInstance()
4. // static method.
5. // Use the getService() method of the DatastoreOptions
6. // object to get the Datastore client
7. private Datastore datastore =
8. DatastoreOptions.getDefaultInstance().getService();
9. // END TODO

content\_copy

After the updates, the first part of QuestionService.java is as follows:

package com.google.training.appdev.services.gcp.datastore;

// TODO: Import the com.google.cloud.datastore.\* package

import com.google.cloud.datastore.\*;

// END TODO

import com.google.training.appdev.services.gcp.domain.Question;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.springframework.stereotype.Service;

@Service

public class QuestionService {

// TODO: Create a Datastore client object, datastore

// The DatastoreOptions class has a getDefaultInstance()

// static method.

// Use the getService() method of the DatastoreOptions

// object to get the Datastore client

private Datastore datastore =

DatastoreOptions.getDefaultInstance().getService();

// END TODO

content\_copy

### **Write code to create a Cloud Datastore entity**

1. Declare a static final string named ENTITY\_KIND, initialized with the value "Question".
2. // TODO: Declare a static final String named kind
3. //The Datastore key is the equivalent of a primary key in a // relational database.
4. // There are two main ways of writing a key:
5. // 1. Specify the kind, and let Datastore generate a unique // numeric id
6. // 2. Specify the kind and a unique string id
7. private static final String ENTITY\_KIND = "Question";
8. // END TODO

content\_copy

1. Create a KeyFactory for Question entities.

For more information on entities, see [Entities, Properties, and Keys](https://cloud.google.com/datastore/docs/concepts/entities).

// TODO: Create a KeyFactory for Question entities

private final KeyFactory keyFactory =

datastore.newKeyFactory().setKind(ENTITY\_KIND);

// END TODO

content\_copy

1. In the createQuestion(Question question) method, modify the method's return type to Key.
2. // TODO: Modify return type to Key
3. public Key createQuestion(Question question) {
4. // END TODO

content\_copy

1. Declare a key with an allocated ID for the Question entity using the datastore client and Key Factory.
2. // TODO: Declare the entity key,
3. // with a Datastore allocated id
4. Key key = datastore.allocateId(keyFactory.newKey());
5. // END TODO

content\_copy

For more information see [Class KeyFactory](https://cloud.google.com/appengine/docs/standard/java/javadoc/com/google/appengine/api/datastore/KeyFactory).

1. Declare an entity named questionEntity, and initialize it using an entity builder.
2. // TODO: Declare the entity object, with the key and data
3. // The entity's members are set using the Entity.Builder.
4. // This has a set method for property names and values
5. // Values are retrieved from the Domain object
6. Entity questionEntity = Entity.newBuilder(key)
7. .set(Question.QUIZ, question.getQuiz())
8. .set(Question.AUTHOR, question.getAuthor())
9. .set(Question.TITLE, question.getTitle())
10. .set(Question.ANSWER\_ONE,question.getAnswerOne())
11. .set(Question.ANSWER\_TWO, question.getAnswerTwo())
12. .set(Question.ANSWER\_THREE,question.getAnswerThree())
13. .set(Question.ANSWER\_FOUR, question.getAnswerFour())
14. .set(Question.CORRECT\_ANSWER,
15. question.getCorrectAnswer())
16. .build();
17. // END TODO

content\_copy

1. Use the Datastore client object (datastore) to save the entity by calling its put(questionEntity) method.
2. // TODO: Save the entity
3. datastore.put(questionEntity);
4. // END TODO

content\_copy

1. Modify the return statement to return the key for the entity.
2. // TODO: Return the key
3. return key;
4. // END TODO

content\_copy

1. Save the file.

The following is the QuestionService.java content with all updates to this point.

// The createQuestion(Question question) method

// is passed a Question object using data from the form

// Extract the form data and add it to Datastore

// TODO: Modify return type to Key

public Key createQuestion(Question question) {

// END TODO

// TODO: Declare the entity key,

// with a Datastore allocated id

Key key = datastore.allocateId(keyFactory.newKey());

// END TODO

// TODO: Declare the entity object, with the key and data

// The entity's members are set using the Entity.Builder.

// This has a set method for property names and values

// Values are retrieved from the Domain object

Entity questionEntity = Entity.newBuilder(key)

.set(Question.QUIZ, question.getQuiz())

.set(Question.AUTHOR, question.getAuthor())

.set(Question.TITLE, question.getTitle())

.set(Question.ANSWER\_ONE,question.getAnswerOne())

.set(Question.ANSWER\_TWO, question.getAnswerTwo())

.set(Question.ANSWER\_THREE,question.getAnswerThree())

.set(Question.ANSWER\_FOUR, question.getAnswerFour())

.set(Question.CORRECT\_ANSWER,

question.getCorrectAnswer())

.build();

// END TODO

// TODO: Save the entity

datastore.put(questionEntity);

// END TODO

// TODO: Return the key

return key;

// END TODO

content\_copy

From Cloud Shell, click on the **Open Terminal** icon.

### **Run the application and create a Cloud Datastore entity**

1. In Cloud Shell, run the application:
2. mvn spring-boot:run

content\_copy

When the application is running you'll see output similar to the following:

08:11:19.014 [restartedMain] INFO c.g.training.appdev.QuizApplication - Started QuizApplication in 10.401 seconds (JVM running for 11.28)

content\_copy

1. In Cloud Shell, click **Web preview** > **Preview on port 8080** to preview the quiz application.
2. Click **Create Question**, complete the form with the following values, and then click **Save**.

|  |  |
| --- | --- |
| **Form Field** | **Value** |
| Author | Your Name |
| Quiz | Google Cloud |
| Title | Which company owns Google Cloud? |
| Answer 1 | Amazon |
| Answer 2 | **Google** (select the Answer 2 radio button!) |
| Answer 3 | IBM |
| Answer 4 | Microsoft |

You are returned to the application home page.

The question you just made is now in DataReturn. In the Console, click **Navigation menu** > **Datastore** > **Entities** to see your new question!

Click Check my progress to verify the objective.

Run the application and create a Cloud Datastore entity

Check my progress

## Query Cloud Datastore

In this section, you write code to retrieve entity data from Cloud Datastore.

### **Write code to retrieve Cloud Datastore entities**

From Cloud Shell, click on the **Open Editor** icon.

1. Move to the getAllQuestions(String quiz) method in the .../services/gcp/datastore/QuestionService.java file, and remove the code for the existing Dummy questions.
2. // TODO: Remove this code
3. List<Question> questions = new ArrayList<>();
4. Question dummy = new Question.Builder()
5. .withQuiz("gcp")
6. .withAuthor("Dummy Author")
7. .withTitle("Dummy Title")
8. .withAnswerOne("Dummy Answer One")
9. .withAnswerTwo("Dummy Answer Two")
10. .withAnswerThree("Dummy Answer Three")
11. .withAnswerFour("Dummy Answer Four")
12. .withCorrectAnswer(1)
13. .withId(-1)
14. .build();
15. questions.add(dummy);
16. return questions;
17. // END TODO

content\_copy

1. Return the transformed results, using buildQuestions(entities) method to convert Datastore entities to domain objects.
2. // TODO: Return the transformed results
3. // Use the buildQuestions(entities) method to convert
4. // from Datastore entities to domain objects
5. return null;
6. // END TODO

content\_copy

1. In the getAllQuestions(String quiz)method, create a query object and initialize it with a query that retrieves Question entities for a specific quiz from Cloud Datastore.
2. public List<Question> getAllQuestions(String quiz){
3. // TODO: Create the query
4. // The Query class has a static newEntityQueryBuilder()
5. // method that allows you to specify the kind(s) of
6. // entities to be retrieved.
7. // The query can be customized to filter the Question
8. // entities for one quiz.
9. Query<Entity> query = Query.newEntityQueryBuilder()
10. .setKind(ENTITY\_KIND)
11. .setFilter(StructuredQuery.PropertyFilter.eq(
12. Question.QUIZ, quiz))
13. .build();
14. // END TODO

content\_copy

For more information, see: [Datastore Queries](https://cloud.google.com/datastore/docs/concepts/queries).

1. Call the Datastore client object's datastore.run(query) method, and assign the result to entity iterator named entities.
2. // TODO: Execute the query
3. // The datastore.run(query) method returns an iterator
4. // for entities
5. Iterator<Entity> entities = datastore.run(query);
6. // END TODO

content\_copy

1. Uncomment the buildQuestions(...) and entityToQuestion(...) helper methods provided in the QuestionService class and use them to map the iterator to a list of question domain objects.
2. // TODO: Return the transformed results
3. // Use the buildQuestions(entities) method to convert
4. // from Datastore entities to domain objects
5. return buildQuestions(entities);
6. // END TODO

content\_copy

The following is the updated QuestionService.java:

public List<Question> getAllQuestions(String quiz){

// TODO: Create the query

// The Query class has a static newEntityQueryBuilder()

// method that allows you to specify the kind(s) of

// entities to be retrieved.

// The query can be customized to filter the Question

// entities for one quiz.

Query<Entity> query = Query.newEntityQueryBuilder()

.setKind(ENTITY\_KIND)

.setFilter(StructuredQuery.PropertyFilter.eq(

Question.QUIZ, quiz))

.build();

// END TODO

// TODO: Execute the query

// The datastore.run(query) method returns an iterator

// for entities

Iterator<Entity> entities = datastore.run(query);

// END TODO

// TODO: Return the transformed results

// Use the buildQuestions(entities) method to convert

// from Datastore entities to domain objects

return buildQuestions(entities);

// END TODO

}

private List<Question> buildQuestions(Iterator<Entity> entities){

List<Question> questions = new ArrayList<>();

entities.forEachRemaining(entity-> questions.add(entityToQuestion(entity)));

return questions;

}

private Question entityToQuestion(Entity entity){

return new Question.Builder()

.withQuiz(entity.getString(Question.QUIZ))

.withAuthor(entity.getString(Question.AUTHOR))

.withTitle(entity.getString(Question.TITLE))

.withAnswerOne(entity.getString(Question.ANSWER\_ONE))

.withAnswerTwo(entity.getString(Question.ANSWER\_TWO))

.withAnswerThree(entity.getString(Question.ANSWER\_THREE))

.withAnswerFour(entity.getString(Question.ANSWER\_FOUR))

.withCorrectAnswer(entity.getLong(Question.CORRECT\_ANSWER))

.withId(entity.getKey().getId())

.build();

}

}

content\_copy

### **Run the application and test the Cloud Datastore query**

1. Save the .../services/gcp/datastore/QuestionService.java file, and then return to the Cloud Shell command.
2. From Cloud Shell, click on the **Open Terminal** icon. Stop the application by pressing **Ctrl**+**C**.
3. Start the application.
4. In Cloud Shell, click **Web preview > Preview on port 8080** to preview the quiz application.
5. Replace the query string at the end of the application's URL with /api/quizzes/gcp.

The URL is in the form: https://8080-dot-####-dot-devshell.appspot.com/api/quizzes/gcp You should see that JSON data has been returned to the client corresponding to the question you added in the web application!

1. Return to the application home page, and click **Take Test** and then click **GCP**.

You should see that the quiz question has been formatted inside the client-side web application!

Lab 3:

# App Dev-Storing Image and Video Files in Cloud Storage-Java

1 hourFree

Rate Lab

## GSP168



## Overview

Cloud Storage allows world-wide storage and retrieval of any amount of data at any time. You can use Cloud Storage for a range of scenarios including serving website content, storing data for archival and disaster recovery, or distributing large data objects to users via direct download.

In this lab you'll configure an application to use Cloud Storage to store and retrieve application data. The application is an online Quiz, the data is the form data, including an image you upload from your local machine.

## Objectives

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

* Set up Cloud Shell as your development environment
* Update the application code to integrate Cloud Datastore
* Use the Quiz application to upload an image file into Cloud Storage and view the image in the Quiz

## Setup

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Review the Quiz application

In this section, you access Cloud Shell, clone the git repository containing the Quiz application, and run the application.

### **Clone source code in Cloud Shell**

Clone the repository for the class.

git clone https://github.com/GoogleCloudPlatform/training-data-analystcontent\_copy

### **Configure and run the Quiz application**

1. Change the working directory:

cd ~/training-data-analyst/courses/developingapps/java/cloudstorage/startcontent\_copy

1. Configure the application:

. prepare\_environment.shcontent\_copy

This script file:

* + Creates an App Engine application
  + Exports an environment variable GCLOUD\_PROJECT
  + Runs mvn clean install
  + Prints out the Project ID.

When the application has been configured, you'll see output similar to the following:

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 35.702 s

[INFO] Finished at: 2018-05-20T21:22:33-04:00

[INFO] Final Memory: 35M/86M

[INFO] ------------------------------------------------------------------------

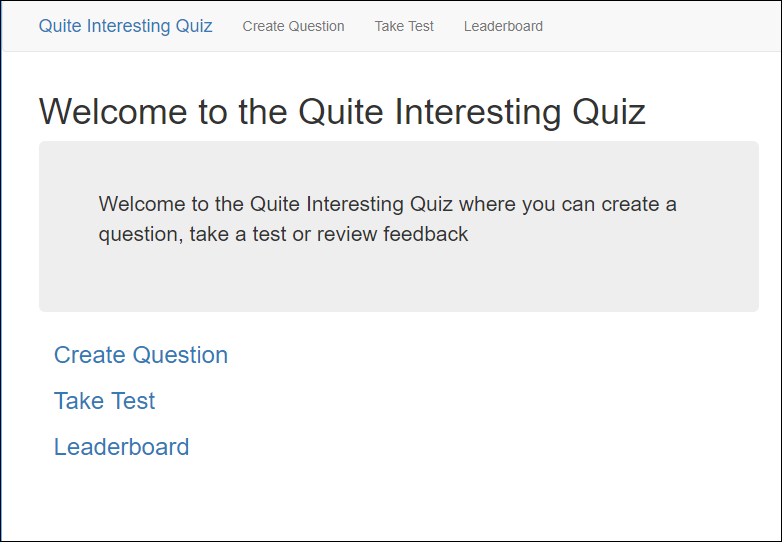
Project ID: qwiklabs-gcp-a3b32951ac6cfb59content\_copy

1. To run the application, execute the following command:

mvn spring-boot:runcontent\_copy

### **Review the Quiz application**

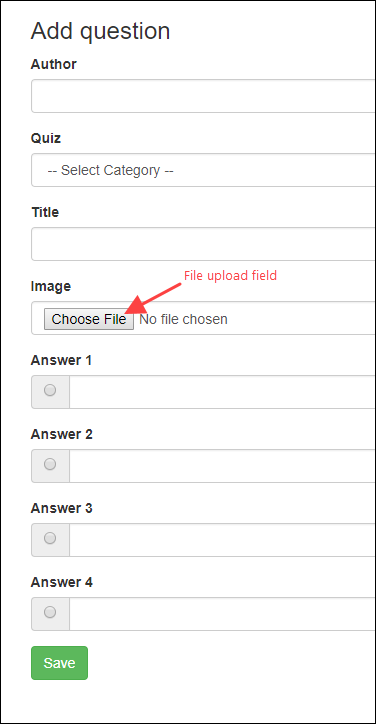
1. To view the application, click **Web preview** > **Preview on port 8080**.



1. Click **Create Question** in the toolbar.

You should see a simple form that contains textboxes for the question and answers and radio buttons to select the correct answer.

The form has a new file upload field that is used to upload an image or video file. In this lab you upload an image file; you would use the same process to upload a video file.



## Examining the Quiz application code

In this section, you use the Cloud Shell text editor to review the Quiz application code.

### **Launch the Cloud Shell text editor**

1. From Cloud Shell, click **Launch the code editor** icon (looks like a pencil) to launch the code editor.



The code editor launches in a separate tab of your browser, along with Cloud Shell.

1. Navigate to /training-data-analyst/courses/developingapps/java/cloudstorage/start folder using the file browser panel on the left side of the editor.

### **Review the Spring Boot Web application**

1. Select new\_question.html file in the .../src/main/resources/templates folder.

This file contains the Thyme template for the Create Question form. Notice how the form uses multipart/form-data as the enc-type, and the two form controls for images and videos:

* + A file upload control called image
  + A hidden field called imageUrl

1. Select the .../src/main/java/com/google/training/appdev folder.

Java file paths are relative to this folder.

1. Select .../web/QuestionsController.java.

In this file the POST handler invokes a method on an image service.

1. Select the .../services/gcp/cloudstorage/ImageService.java file.

This is the file where you write code to save image file data into Cloud Storage.

## Creating a Cloud Storage Bucket

In this section, you create a Cloud Storage bucket and export an environment variable that references it.

### **Create a Cloud Storage bucket**

1. Return to Cloud Shell and stop the application by pressing **Ctrl**+**C**.
2. To create a Cloud Storage bucket named <Project ID>-media, execute the following command:

gsutil mb gs://$DEVSHELL\_PROJECT\_ID-mediacontent\_copy

You can create a bucket using the gsutil mb command, passing through the name of the bucket as gs://BUCKET\_NAME You can use $DEVSHELL\_PROJECT\_ID as the bucket name prefix followed by -media

Click **Check my progress** to verify the objective.

Create a storage bucket

Check my progress

1. To export the Cloud Storage bucket name as an environment variable named GCLOUD\_BUCKET, execute the following command:

export GCLOUD\_BUCKET=$DEVSHELL\_PROJECT\_ID-mediacontent\_copy

Recall that the application makes use of environment variables for configuration. This allows the development team to deploy the application into development, test, staging, and production just by changing these variables.

## Adding Objects to Cloud Storage

In this section, you write code to save uploaded files into Cloud Storage.

**Important:** Update code within the sections marked as follows: // TODO // END TODO To maximize your learning, review the code, inline comments, and related [Cloud Storage APIs & Reference documentation](https://cloud.google.com/storage/docs/apis).

### **Import and use the Java Cloud Storage Package**

1. In the code editor, move to the top of the .../services/gcp/cloudstorage/ImageService.java file.
2. Write a star import for the com.google.cloud.storage.\* package.
3. // TODO: Write a start import for Cloud Storage
4. import com.google.cloud.storage.\*;
5. // END TODO

content\_copy

1. Construct a Cloud Storage client named storage using the StorageOptions class.
2. // TODO: Create the storage client
3. // The StorageOptions class has a getDefaultInstance()
4. // static method.
5. // Use the getService() method to get the storage client
6. private static Storage storage = StorageOptions
7. .getDefaultInstance()
8. .getService();
9. // END TODO

content\_copy

1. Declare a String named bucketName, and annotate it with a Spring value annotation to retrieve the value from a property named google.storage.bucket.
2. Declare a constant named bucket to reference the Cloud Storage bucket.

// TODO: Get the name of the Cloud Storage bucket

// Use a Spring @Value annotation to get the value

// Get the value using ${google.storage.bucket}

// This references the GCLOUD\_BUCKET environment variable

@Value("${google.storage.bucket}")

private String bucketName;

// END TODO

content\_copy

#### ImageService.java

package com.google.training.appdev.services.gcp.cloudstorage;

// TODO: Write a start import for Cloud Storage

import com.google.cloud.storage.\*;

// END TODO

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

@Service

public class ImageService {

// TODO: Create the storage client

// The StorageOptions class has a getDefaultInstance()

// static method.

// Use the getService() method to get the storage client

private static Storage storage = StorageOptions

.getDefaultInstance()

.getService();

// END TODO

// TODO: Get the name of the Cloud Storage bucket

// Use a Spring @Value annotation to get the value

// Get the value using ${google.storage.bucket}

// This references the GCLOUD\_BUCKET environment variable

@Value("${google.storage.bucket}")

private String bucketName;

// END TODO

content\_copy

### **Write code to send a file to Cloud Storage**

Still updating ImageService.java, move to the saveImage(MultipartFile file) handler and then use the Cloud Storage client to upload a file to your Cloud Storage bucket and make it publicly available.

1. Declare a BlobInfo object and initialize it using the storage client object. Customize the BlobInfo object using its Builder. Use the option to set the Content Type and to set the ACL to allow unauthenticated Read access.
2. // TODO: Create a new Cloud Storage object
3. // Use the BlobInfo class to represent this object
4. // Use the BlobInfo.Builder to customize the Blob
5. // Set the content type from the file
6. // Set the object ACL to Public Read
7. BlobInfo blobInfo = storage.create(
8. BlobInfo.newBuilder(bucketName, fileName)
9. .setContentType(file.getContentType())
10. .setAcl(new ArrayList<>(
11. Arrays.asList(Acl.of(Acl.User.ofAllUsers(),
12. Acl.Role.READER))))
13. .build(),
14. file.getInputStream());
15. // END TODO

content\_copy

1. Return the public URL for the new Cloud Storage object as a string.
2. // TODO: Cloud Storage public URLs are in the form:
3. // https://storage.googleapis.com/[BUCKET]/[OBJECT]
4. // Use String concatentation to create return the URL
5. return "https://storage-download.googleapis.com/" + bucketName+ "/" +fileName;
6. // END TODO

content\_copy

#### ImageService.java

public String saveImage(MultipartFile file)

throws IOException {

// The existing code in the method creates a unique name

// based on the file's original name. It has a

// prefix generated using the current date and time.

// This should ensure that a new file upload won't

// overwrite an existing object in the bucket

String fileName = System.nanoTime() +

file.getOriginalFilename();

// TODO: Create a new Cloud Storage object

// Use the BlobInfo class to represent this object

// Use the BlobInfo.Builder to customize the Blob

// Set the content type from the file

// Set the object ACL to Public Read

BlobInfo blobInfo = storage.create(

BlobInfo.newBuilder(bucketName, fileName)

.setContentType(file.getContentType())

.setAcl(new ArrayList<>(

Arrays.asList(Acl.of(Acl.User.ofAllUsers(),

Acl.Role.READER))))

.build(),

file.getInputStream());

// END TODO

// TODO: Cloud Storage public URLs are in the form:

// https://storage.googleapis.com/[BUCKET]/[OBJECT]

// Use String concatenation to return the URL

return "https://storage-download.googleapis.com/" +

bucketName + "/" + fileName;

// END TODO

}

content\_copy

### **Run the application and create a Cloud Storage object**

1. Save .../services/gcp/cloudstorage/ImageService.java and then return to the Cloud Shell.
2. Start the application by typing
3. mvn spring-boot:run

content\_copy

1. Download an image file to your local machine from [here](https://storage.googleapis.com/cloud-training/quests/Google_Cloud_Storage_logo.png).
2. In Cloud Shell, click **Web preview** > **Preview on port 8080** to preview the Quiz application.
3. Click the **Create Question** link.
4. Complete the form with the following values, and then click **Save**.

|  |  |
| --- | --- |
| **Field** | **Value** |
| Author | Your name |
| Quiz | Google Cloud |
| Title | Which product does this logo relate to? |
| Image | Upload the Google\_Cloud\_Storage\_logo.png file you previously downloaded |
| Answer 1 | App Engine |
| Answer 2 | Cloud Storage (and select the Answer 2 radio button to indicate this as the correct answer) |
| Answer 3 | Compute Engine |
| Answer 4 | Container Engine |

1. Click **Check my progress** to verify the objective.
2. Add Objects to Cloud Storage
3. Check my progress
4. Return to the Cloud Console and navigate to **Navigation menu** > **Storage**.
5. On the **Storage** > **Browser** page, click the correct bucket (named <Project ID>-media).

You should see your new object named #UniqueNumber#Google\_Cloud\_Storage\_logo.png.

Click **Check my progress** to verify the objective.

Create a Datastore entity

Check my progress

### **Run the client application and test the Cloud Storage public URL**

1. Add /api/quizzes/gcp to the end of the Quiz application's URL.

You should see that the application returns JSON data to the client corresponding to the Question you added in the web application.

{"questions":[{"quiz":"gcp","author":"Mary Smith","title":"Which product does this logo relate to?","correctAnswer":2,"imageUrl":"https://storage-download.googleapis.com/qwiklabs-gcp-33f2cf3e36ee1794-media/2293278773582Websites\_and\_Web\_Apps\_GCP.png","image":null,"id":5629499534213120,"answer1":"App Engine","answer2":"Cloud Storage","answer3":"Compute Engine","answer4":"Container Engine"}]}

content\_copy

The imageUrl property should have a value corresponding to the new object you created in Cloud Storage.

1. Return to the application home page and click the **Take Test** link.
2. Click **GCP**, and answer each question.

When you get to the question you just added, you should see the image has been formatted inside the client-side web application!

Lab 4

# App Dev-Adding User Authentication to your Application-Java

50 minutesFree

## GSP169



## Overview

This lab shows you how to add authentication to your application using Firebase. This authorization identifies who you are, and determines what you can do. For more information, see [Authentication Overview](https://cloud.google.com/docs/authentication/).

Firebase is a comprehensive framework that allows you to create web and mobile applications. It integrates with Google Cloud so that you can import a Google Cloud project into a Firebase project. For more information about Firebase, see [Cloud Functions and Firebase](https://firebase.google.com/docs/functions/functions-and-firebase).

The application used in this lab is an online Quiz application. You add Firebase authentication and then configure authentication to use a simple email address and password credential, and ensure that users must register and log in before taking a quiz.

## Objectives

In this lab, you will perform the following tasks:

* Register a Google Cloud project with Firebase.
* Add Firebase configuration to a client-side web application.
* Write Java code to integrate Firebase Authentication into a client-side web application.

## Setup and requirements

### **Qwiklabs setup**

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **The Google Cloud Shell**

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Prepare the Quiz application

In this section, you access Cloud Shell, clone the git repository containing the Quiz application, configure environment variables, and run the application.

### **Clone source code in Cloud Shell**

To clone the repository for this lab enter the following command:

git clone https://github.com/GoogleCloudPlatform/training-data-analystcontent\_copy

### **Configure and run the case study application**

1. Change to the working directory::

cd ~/training-data-analyst/courses/developingapps/java/firebase/startcontent\_copy

1. Configure the application dependencies:
2. . prepare\_environment.sh

content\_copy

This script file:

* + Creates an App Engine application.
  + Creates a Cloud Storage bucket named gs://[Project-ID]-media.
  + Exports two environment variables: GCLOUD\_PROJECT and GCLOUD\_BUCKET.
  + Runs mvn clean install.
  + Creates entities in Cloud Datastore.
  + Prints out the Project ID.

When the application dependencies are configured, you will see the following output:

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 8.225 s

[INFO] Finished at: 2018-05-16T13:01:20-04:00

[INFO] Final Memory: 21M/51M

[INFO] ------------------------------------------------------------------------content\_copy

Click Check my progress to verify the objective.

Configure the case study application

Check my progress

1. To run the application, enter the following command:
2. mvn spring-boot:run

content\_copy

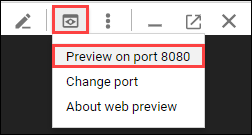
The application has started when you see the last line of the output, which looks similar to the following example output:

08:01:40.245 [restartedMain] INFO c.g.training.appdev.QuizApplication - Started QuizApplication in 12.428 seconds (JVM running for 13.485)

content\_copy

### **Start the Quiz application**

In Cloud Shell, click **Web preview** > **Preview on port 8080** to preview the Quiz application.



Leave this window open for now, because you will need the Cloud Shell Web preview domain (which is in the form: 8080-dot-2958229-dot-devshell.appspot.com) later in this lab.

## Examine the Quiz application code

In this section and throughout the lab you'll review the Quiz application code in a code editor. You can use the shell editors that are installed on Cloud Shell, such as nano or vim, or use the Cloud Shell code editor. This lab uses the Cloud Shell code editor.

### **Launch the Cloud Shell text editor**

From Cloud Shell, click on the **Open Editor** icon in the top ribbon.

Navigate to the /training-data-analyst/courses/developingapps/java/firebase/start folder using the file browser panel on the left side of the editor.

In the firebase folder, notice the end folder. The end folder contains the same files as the start folder, but each file in the end folder contains the complete code required to perform this lab.

### **Review the client application**

1. Select the index.html in the .../src/main/resources/static/client folder. Do not confuse with index.html in the static folder.

This file is the single page in the AngularJS SPA. It contains <script></script> tags for the application libraries and code and markup where the SPA will render dynamic output.

1. Select qiq-login-template.html in the .../src/main/resources/static/client/app/auth/ folder.

This file contains the AngularJS template for the Login component.

Notice how it contains a couple of textboxes and a button. The button has an event handler that runs code when it is clicked.

1. Select qiq-login.js.

This file contains an AngularJS component. It allows the user to log in to the application or to navigate to a registration page.

## Work with Firebase

In this section, you will create a Firebase project for your Google Cloud project and enable the Authentication feature.

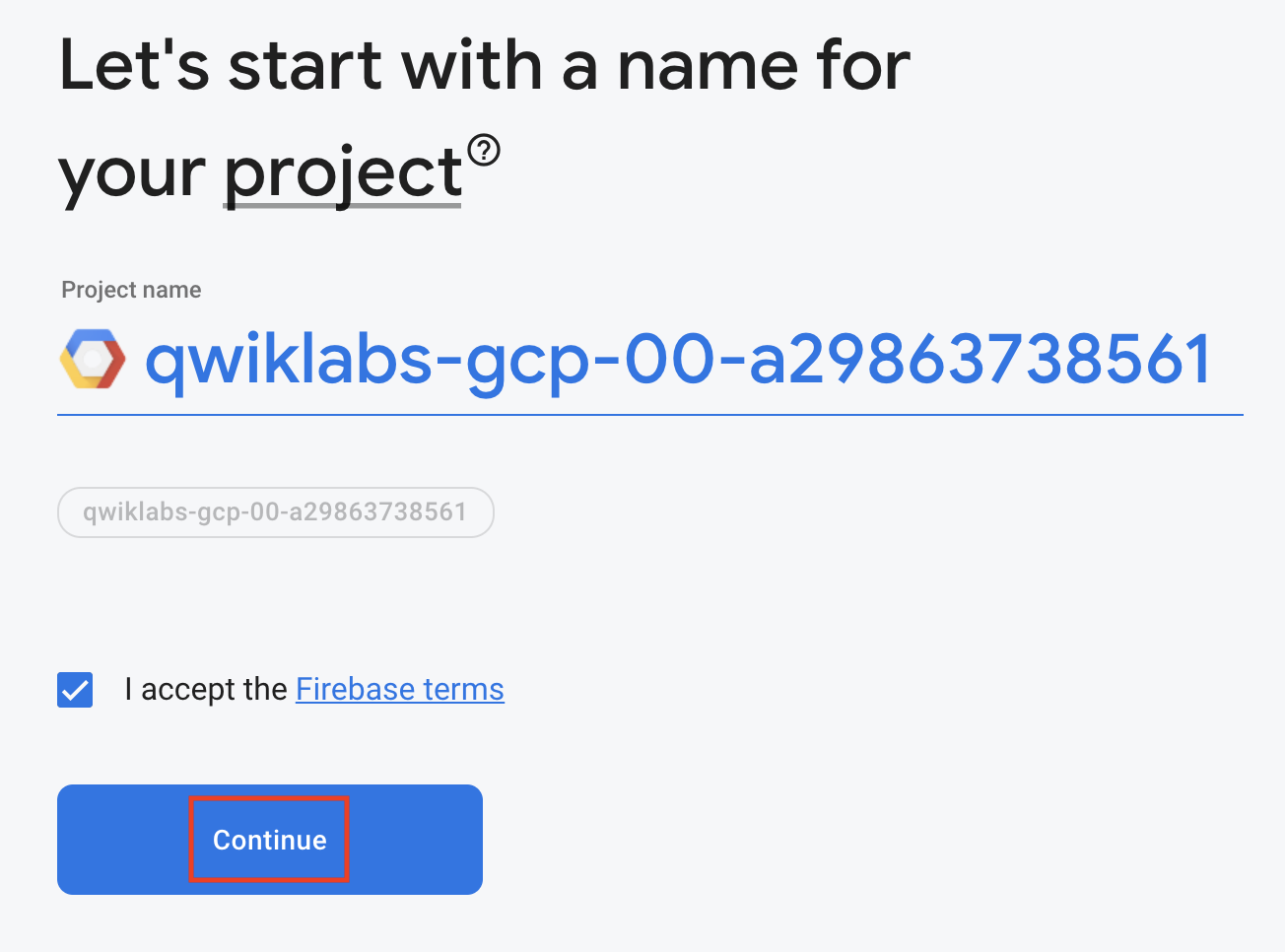
### **Create a Firebase project**

1. From the Google Cloud console, open a new browser tab and navigate to <https://console.firebase.google.com/>.

Be sure to open the new tab from the Google Cloud console window to stay in the lab environment. You may need to sign in again using your Qwiklabs username and password.

### **Add your project to Firebase:**

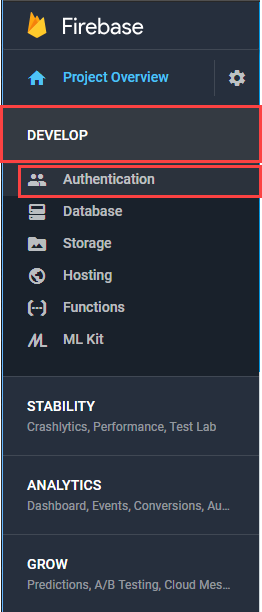
1. On the **Welcome to Firebase** page, click **Add project**.
2. In the **Add a project** dialog, select your Qwiklabs Google Cloud Project from the **Project name** dropdown.
3. Accept the Firebase terms and conditions and click **Continue**.



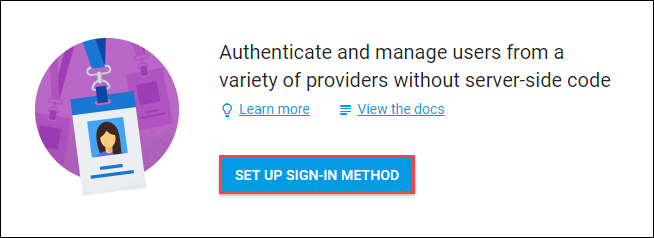
1. In the **Confirm Firebase billing plan** dialog, click **CONFIRM PLAN**.
2. Click **Continue**.
3. Click **Continue**.
4. Accept the final two terms of conditions and then click **Add Firebase**.
5. Then click **Continue**.

### **Configure Firebase Authentication**

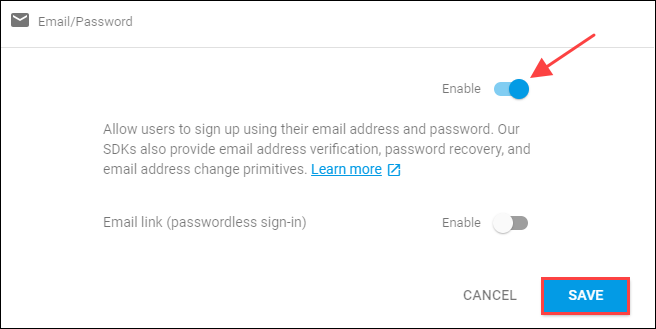
1. In the left pane of the Firebase page, click **Develop > Authentication**.



1. On the **Authentication** page, click **Set up sign-in method**.



1. On the **Sign-in providers** page, hover over the Email/Password provider, and then click the edit icon (a pencil).
2. Click **Enable** to allow users to sign up using their email address and password. Click **Save**.



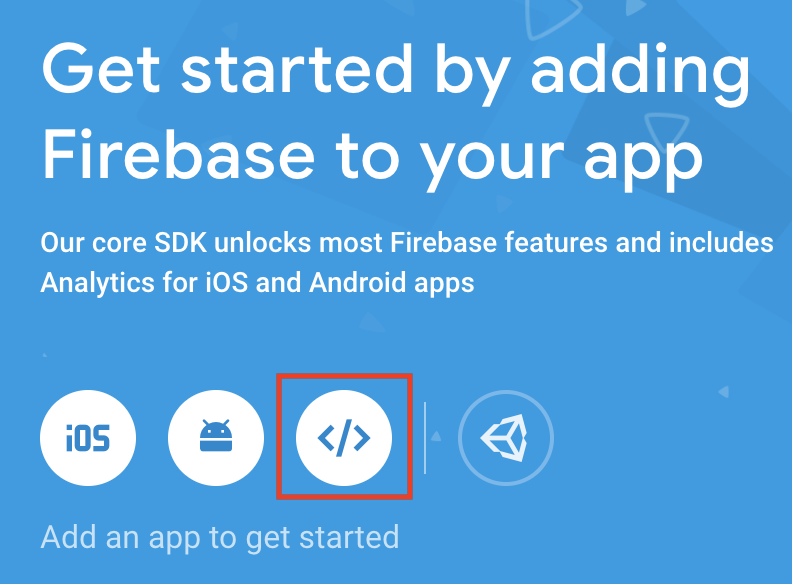
1. Under **Authorized Domains**, click **ADD DOMAIN**.
2. Return to your running Quiz application and copy the domain (which has the format 8080-dot-2999129-dot-devshell.appspot.com).
3. Return to the Firebase console, paste the domain into the **Domain** text box, and then click **Add**.

## Integrate a Client-Side Web Application with Firebase

In this section, you will apply your Firebase configuration to your client-side web application.

### **Apply Firebase configuration to a web application**

1. Click **Project Overview** from the left-hand navigation menu. From the "Get started by adding Firebase to your app" console, select the web icon:



1. This will bring you to a "Add Firebase to your web app" page. For the **App nickname** field, enter in **firebase\_app**. Then click **Register app**.

Click Check my progress to verify the objective.

Work with firebase

Check my progress

1. You will now be brought to the "Add Firebase SDK" page. To copy the Firebase configuration, click the copy button in the bottom right corner of the code block. A sample configuration is shown below.

**Firebase configuration example (DO NOT COPY)**

<!-- DO NOT COPY THIS BLOCK - IT'S A SAMPLE -->

<!-- The core Firebase JS SDK is always required and must be listed first -->

<script src="https://www.gstatic.com/firebasejs/6.2.0/firebase-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use

https://firebase.google.com/docs/web/setup#config-web-app -->

<script>

// Your web app's Firebase configuration

var firebaseConfig = {

apiKey: "AIzaSyAZNRXW1DylpbJu2fmfjIcyKYtz0nOz7Pc",

authDomain: "qwiklabs-gcp-9a7cb292df849681.firebaseapp.com",

databaseURL: "https://qwiklabs-gcp-9a7cb292df849681.firebaseio.com",

projectId: "qwiklabs-gcp-9a7cb292df849681",

storageBucket: "qwiklabs-gcp-9a7cb292df849681.appspot.com",

messagingSenderId: "776822443186",

appId: "1:776822443186:web:5c5e92b0d93ae206"

};

// Initialize Firebase

firebase.initializeApp(firebaseConfig);

</script>

content\_copy

1. In the **Cloud Shell code editor**, open and update .../src/main/resources/static/client/index.html:

Paste the Firebase configuration markup just before the other <script></script> tags at the bottom of the page.

1. Now replace this this line of code that you copied over:

<script src="https://www.gstatic.com/firebasejs/7.14.2/firebase-app.js"></script>

content\_copy

with the following:

<script src="https://www.gstatic.com/firebasejs/4.2.0/firebase.js"></script>

content\_copy

1. Click the **Open Terminal** icon.
2. In Cloud Shell, press Ctrl-C to stop the application, and then re-enter the following command to restart your application.

mvn spring-boot:run

content\_copy

### **Run the application**

1. Return to the Quiz application and refresh your browser.
2. In the navigation bar, click **Take Test**.

You should not be able to take a test without being logged in.

1. Register for access. In the navigation bar, click **Register**.

Enter any email address and password into the form fields, and then click **Register**.

You will have to use a password with a certain level of complexity, like abcd1234$. You should be logged in and redirected to the Google Cloud quiz.

1. In the navigation bar, click **Logout**.

You should be logged out and redirected to the Quiz homepage.

Try logging in again, register another identity, and have fun taking the quiz.

# App Dev: Developing a Backend Service - Java

1 hour 20 minutesFree

## GSP170



## Overview

In this lab, you develop a backend service for an online Quiz application to process user feedback and save scores.

The Quiz application has two parts, the web application that will run in the first Cloud shell window and the worker application that runs in the second Cloud Shell window.

* Web application: manages the logic of sending the user's feedback to a pub/sub topic.
* Worker application: listens to the feedback provided by the user to eventually perform sentiment analysis and store them in a database (Cloud Spanner).

This process takes advantage of Google Cloud products and services:

* Cloud Pub/Sub: The Topic alerts and provides the subscribing worker application to new scores and feedback for analysis.
* Cloud Natural Language: Provides sentiment analysis on the feedback.
* Cloud Spanner: Database for the Quiz application.

## Objectives

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

* Create and publish messages to a Cloud Pub/Sub topic.
* Subscribe to the topic to receive messages in a separate worker application.
* Perform sentiment analysis on feedback.
* Create a Cloud Spanner database instance and schema, then insert data into the database.

## Setup

### **Qwiklabs setup**

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

### **Cloud Console**

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **Cloud Shell**

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Prepare the Quiz application

In this section, you access Cloud Shell, clone the git repository containing the Quiz application, configure environment variables, and run the application.

### **Clone source code in Cloud Shell**

Enter the following command to clone the repository for the class:

git clone https://github.com/GoogleCloudPlatform/training-data-analyst

content\_copy

### **Set up the web application**

In this section you'll configure dependencies and run the web application.

1. Enter the following command to change the working directory:
2. cd ~/training-data-analyst/courses/developingapps/java/pubsub-languageapi-spanner/start

content\_copy

1. Configure the web application:
2. . prepare\_web\_environment.sh

content\_copy

This script file:

* + Creates an App Engine application.
  + Exports environment variables: GCLOUD\_PROJECT and GCLOUD\_BUCKET.
  + Runs mvn clean install.
  + Creates entities in Cloud Datastore.
  + Prints out the Project ID.

You can run this application when you see output similar to the following:

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 8.979 s

[INFO] Finished at: 2018-05-28T22:02:26-04:00

[INFO] Final Memory: 21M/51M

[INFO] ------------------------------------------------------------------------

Project ID: qwiklabs-gcp-198243c75d5ac1b7

content\_copy

1. Run the web application:
2. mvn spring-boot:run

content\_copy

The application is running when you see you see output similar to the following.

00:13:09.089 [restartedMain] INFO c.g.training.appdev.QuizApplication - Started QuizApplication in 10.434 seconds (JVM running for 11.351)

content\_copy

### **Set up the worker application**

1. Click **Add Cloud Shell session** (**+**) on the right of the Cloud Shell tab to open a second Cloud Shell window. Enter the following command to change the working directory:
2. cd ~/training-data-analyst/courses/developingapps/java/pubsub-languageapi-spanner/start

content\_copy

1. Prepare the environment in the second Cloud Shell window:
2. . prepare\_worker\_environment.sh

content\_copy

This script file:

* + Exports environment variables GCLOUD\_PROJECT and GCLOUD\_BUCKET.
  + Creates and configures a Google Cloud Service Account.
  + Prints out the Project ID.

Click Check my progress to verify the objective.

Configure the Quiz application

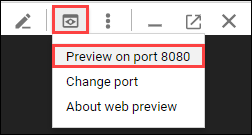
Check my progress

1. Now start the worker application:
2. mvn exec:java@worker

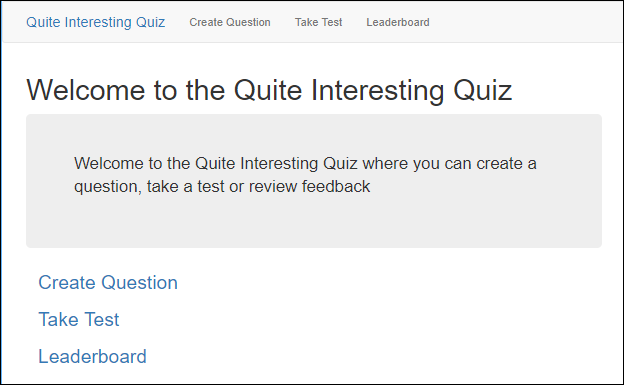
content\_copy

### **Review the Quiz application**

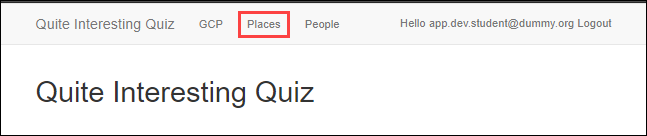
1. Still in the second Cloud Shell window, click **Web preview** > **Preview on port 8080** to preview the Quiz application.



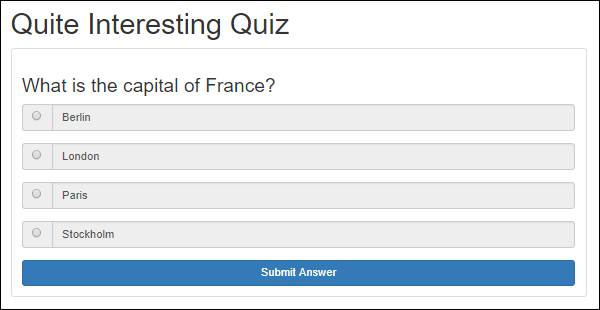
1. In the navigation bar, click **Take Test**.



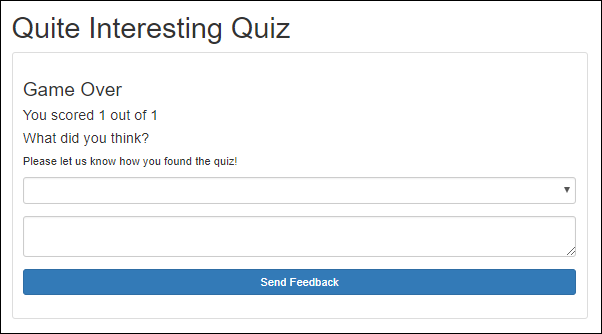
1. Click **Places**.



1. Answer the question.



After you answer the question, you should see a final screen inviting you to submit feedback.



Quiz takers can select a rating and enter feedback.

1. Return Cloud Shell. Press **Ctrl**+**c** in the first and second windows to stop the web and worker applications.

## Examine the Quiz Application Code

In this section you examine the file structure and the files that impact the Quiz application.

In this lab you'll view and edit files. You can use the shell editors that are installed on Cloud Shell, such as nano or vim, or use the Cloud Shell code editor. This lab uses the Cloud Shell code editor.

### **Launch the Cloud Shell Editor**

From the Cloud Shell ribbon, click on the **Open Editor** icon.

### **Review the Google Cloud application code structure**

Navigate to the /training-data-analyst/courses/developingapps/java/pubsub-languageapi-spanner/start folder using the file browser panel on the left side of the editor.

Now expand the /src/main/java/com/google/training/appdev folder. All Java code paths are relative to this folder.

Select the Feedback.java file in the .../services/gcp/domain folder.

This file contains a model class that represents the feedback submitted by quiz takers.

Select the PublishService.java file in the .../services/gcp/pubsub folder.

This file contains a service class that allows applications to publish feedback messages to a Cloud Pub/Sub topic.

Select the LanguageService.java file in the .../services/gcp/languageapi folder. This file contains a service class that allows users to send text to the Cloud Natural Language ML API and to receive the sentiment score from the API.

Select the SpannerService.java file in the .../services/gcp/spanner folder.

This file contains a service class that allows users to save the feedback and Natural Language API response data in a Cloud Spanner database instance.

### **Review the web and backend application code**

Select the QuizEndpoint.java file in the .../apifolder.

The handler for POST messages sent to the /api/quizzes/feedback/:quiz route publishes the feedback data received from the client to Pub/Sub.

Select the ConsoleApp.java file in the .../backend folder.

This file runs as a separate console application to consume the messages delivered to a Pub/Sub subscription.

From the Cloud Shell ribbon, click on the **Open Terminal** icon.

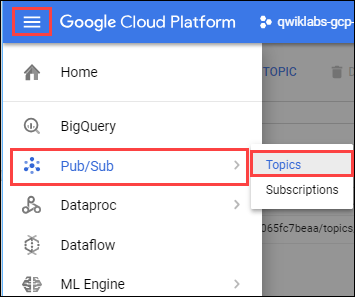
## Working with Cloud Pub/Sub

The Quiz application uses a Pub/Sub function to retrieve answers and feedback that a user inputs through the quiz interface.

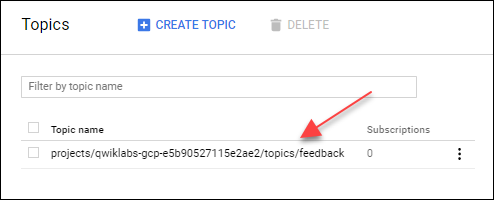
In this section, you create a Cloud Pub/Sub topic and subscription in your Google Cloud project, then publish and retrieve a message.

### **Create a Cloud Pub/Sub topic**

1. In the Console, click **Navigation menu** > **Pub/Sub** > **Topics** and then click **Create topic**.



1. Name the Topic feedback, and then click **CREATE TOPIC**. The following shows the topic you just created:



### **Create a Cloud Pub/Sub subscription**

Return to the second Cloud Shell window.

Enter the following command to create a Cloud Pub/Sub subscription named cloud-shell-subscription against the feedback topic:

gcloud beta pubsub subscriptions create cloud-shell-subscription --topic feedback

content\_copy

If you receive an error about the active account not having valid credentials, wait for a minute and try the command again.

Click Check my progress to verify the objective.

Work with Cloud Pub/Sub

Check my progress

### **Publish a message to a Cloud Pub/Sub topic**

Publish a "Hello World" message into the feedback topic in the second Cloud Shell window:

gcloud beta pubsub topics publish feedback --message="Hello World"

content\_copy

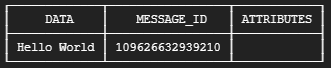
### **Retrieve a message from a Cloud Pub/Sub subscription**

To pull the message from the feedback topic with automatic acknowledgement of the message in the second Cloud Shell window:

gcloud beta pubsub subscriptions pull cloud-shell-subscription --auto-ack

content\_copy

Output:



## Publish Messages to Cloud Pub/Sub Programmatically

In this section, you write code to publish messages to Cloud Pub/Sub.

**Important**: Update code within the sections marked as follows:

// TODO

// END TODO

To maximize your learning, review the code, inline comments, and related API documentation.

### **Publish a Pub/Sub message**

Open the .../services/gcp/pubsub/PublishService.java file in the code editor. Update the file by adding code as directed.

1. Declare two static final strings for the PROJECT\_ID and TOPIC\_NAME.
2. // TODO: Declare and initialize two Strings,
3. // PROJECT\_ID and TOPIC\_NAME
4. private static final String PROJECT\_ID =
5. ServiceOptions.getDefaultProjectId();
6. private static final String TOPIC\_NAME = "feedback";
7. // END TODO

content\_copy

1. Move to the publishFeedback(...) method. Create a TopicName object using the PROJECT\_ID and TOPIC\_NAME strings.

The topic name references the Cloud Pub/Sub topic you just created.

// TODO: Create a TopicName object

// for the feedback topic in the project

TopicName topicName =

TopicName.create(PROJECT\_ID, TOPIC\_NAME);

// END TODO

content\_copy

1. Declare a Publisher object and set it to null. It will be initialized in the try block that follows.
2. // TODO: Declare a publisher for the topic
3. Publisher publisher = null;
4. // END TODO

content\_copy

1. Move to the try block, and initialize the publisher object using its builder.
2. // TODO: Initialize the publisher
3. // using a builder and the topicName
4. publisher = Publisher.defaultBuilder(topicName).build();
5. // END TODO

content\_copy

1. Copy the JSON serialized feedbackMessage string to a ByteString.
2. // TODO: Copy the serialized message
3. // to a byte string
4. ByteString data = ByteString.copyFromUtf8(
5. feedbackMessage);
6. // END TODO

content\_copy

1. Declare a PubsubMessage object; initialize the message using its builder.
2. // TODO: Create a Pub/Sub message using a
3. // builder; set the message data
4. PubsubMessage pubsubMessage = PubsubMessage
5. .newBuilder().setData(data).build();
6. // END TODO

content\_copy

1. Use the publisher to publish the message, assign the return value to the message ID future object.
2. // TODO: Publish the message,
3. // assign to the messageIdFuture
4. messageIdFuture = publisher.publish(
5. pubsubMessage);
6. // END TODO

content\_copy

1. Move to the finally block and retrieve the Pub/Sub messageId from the message ID future object.
2. // TODO: Get the messageId from
3. // the messageIdFuture
4. String messageId = messageIdFuture.get();
5. // END TODO

content\_copy

1. Complete the publishing code by shutting down the publisher.
2. // TODO: Shutdown the publisher
3. // to free up resources
4. if (publisher != null) {
5. publisher.shutdown();
6. }
7. // END TODO

content\_copy

1. Save services/gcp/pubsub/PublishService.java .

### **Write code to use the Pub/Sub publish functionality**

1. In the .../api/QuizEndpoint.java file, declare a new PublishService field named publishService. Apply the Spring @Autowired annotation.
2. // TODO: Declare the publishService
3. @Autowired
4. private PublishService publishService;
5. // END TODO

content\_copy

1. In the processFeedback(...) method that handles POST requests to the '/feedback/:quiz' route, invoke the publishService.publishFeedback(feedback) method.
2. // TODO: Publish the feedback to Pub/Sub
3. publishService.publishFeedback(feedback);
4. // END TODO

content\_copy

1. Save /api/QuizEndpoint.java.

### **Run the application and create a Pub/Sub message**

1. In the first Cloud Shell window, restart the web application (if it is running, stop and start it).
2. mvn spring-boot:run

content\_copy

1. Preview the web application, click **Take Test** > **Places**.
2. Answer the question, select the rating, enter some feedback text, and click **Send Feedback**.
3. In the second Cloud Shell window, to pull a message from the cloud-shell-subscription:
4. gcloud beta pubsub subscriptions pull cloud-shell-subscription --auto-ack

content\_copy

## Subscribing to Cloud Pub/Sub Topics Programmatically

In this section you write the code to create a subscription and receive message notifications from a Cloud Pub/Sub topic to the worker application.

### **Write code to create a Cloud Pub/Sub subscription and receive messages**

In the code editor open the ...backend/ConsoleApp.java file. Update the file by adding code as directed.

1. In the main()method, create a SubscriptionName object representing a new subscription named "worker1-subscription".
2. // TODO: Create the Pub/Sub subscription name
3. SubscriptionName subscription =
4. SubscriptionName.create(projectId,
5. "worker1-subscription");
6. // END TODO

content\_copy

1. Create a SubscriptionAdminClient object using a try block.

Also in the try block, use the subscription admin client to create a new subscription against the feedback topic.

// TODO: Create the subscriptionAdminClient

try (SubscriptionAdminClient subscriptionAdminClient =

SubscriptionAdminClient.create()) {

// TODO: create the Pub/Sub subscription

// using the subscription anema and topic

subscriptionAdminClient.createSubscription(

subscription, topic,

PushConfig.getDefaultInstance(), 0);

// END TODO

}

// END TODO

content\_copy

1. Move to the code that creates a MessageReceiver, and in the receiveMessage(...) override, extract the message data into a String.
2. // TODO: Extract the message data as a JSON String
3. String fb = message.getData().toStringUtf8();
4. // END TODO

content\_copy

1. Use the consumer to acknowledge the message.
2. // TODO: Ack the message
3. consumer.ack();
4. // END TODO

content\_copy

1. After the code that initializes an ObjectMapper, deserialize the JSON String message data into a feedback object.
2. // TODO: Deserialize the JSON String
3. // representing the feedback
4. // Print out the feedback
5. Feedback feedback = mapper.readValue(
6. fb, Feedback.class);
7. System.out.println("Feedback received: "
8. + feedback);
9. // END TODO

content\_copy

1. After the block that creates the MessageReceiver, declare a Subscriber and initialize it to null.
2. // TODO: Declare a subscriber
3. Subscriber subscriber = null;
4. // END TODO

content\_copy

1. Move to the try block, and initialize the Subscriber using its default builder. This requires the subscription and receiver.
2. // TODO: Initialize the subscriber using
3. // its default builder
4. // with a subscription and receiver
5. subscriber = Subscriber.defaultBuilder(
6. subscription, receiver).build();
7. // END TODO

content\_copy

1. Add a listener to the subscriber to display errors.
2. // TODO: Add a listener to the subscriber
3. subscriber.addListener(
4. new Subscriber.Listener() {
5. @Override
6. public void failed(
7. Subscriber.State from,
8. Throwable failure) {
9. System.err.println(failure);
10. }
11. },
12. MoreExecutors.directExecutor());
13. // END TODO

content\_copy

1. Start the subscriber.
2. // TODO: Start subscribing
3. subscriber.startAsync().awaitRunning();
4. // END TODO

content\_copy

1. Move to the finally block. Write the code to stop the subscriber, and delete the subscription.
2. // TODO: Stop subscribing
3. if (subscriber != null) {
4. subscriber.stopAsync().awaitTerminated();
5. }
6. // END TODO
7. // TODO: Delete the subscription
8. try (SubscriptionAdminClient
9. subscriptionAdminClient =
10. SubscriptionAdminClient.create()) {
11. subscriptionAdminClient.deleteSubscription(
12. subscription);
13. }
14. // END TODO

content\_copy

1. Save backend/ConsoleApp.java.

### **Run the web and worker application and create a Pub/Sub message**

1. In the first Cloud Shell window, stop and start the web application.
2. mvn spring-boot:run

content\_copy

1. In the second Cloud Shell window, start the worker application.
2. mvn compile exec:java@worker

content\_copy

1. In Cloud Shell, click **Web preview** > **Preview on port 8080** to preview the quiz application.
2. Click **Take Test**.
3. Click **Places**.
4. Answer the question, select the rating, enter some feedback text, and then click **Send Feedback**.
5. Return to the second Cloud Shell window. You should see that the worker application has received the feedback message via its handler and displayed it in the second Cloud Shell window. An example of a feedback message is as follows
6. Feedback received: Feedback{email='app.dev.student@dummy.org', quiz='places', feedback='love the test', rating=5, timestamp=1527564677609, sentimentScore=0.0}

content\_copy

1. Stop the web and console applications.

## Use the Cloud Natural Language API

In this section you write the code to perform sentiment analysis on the feedback text submitted by the user. For more information see [Cloud Natural Language API](https://cloud.google.com/natural-language/docs/reference/rest/).

### **Write code to invoke the Cloud Natural Language API**

1. Return to the editor and open the LanguageService.java file in the services/gcp/languageapi folder.
2. Move to the analyzeSentiment(...) method, and create a LanguageServiceClient object in a try block. In this step note that there is not a // END TODO in the content that you copy into the file.
3. // TODO: Create the LanguageServiceClient object
4. try (LanguageServiceClient language =
5. LanguageServiceClient.create()) {

content\_copy

1. Create a new Document object using its builder. Configure this object with the document content and type.
2. // TODO: Create a new Document object
3. // using the builder
4. // Set the content and type
5. Document doc = Document.newBuilder()
6. .setContent(feedback)
7. .setType(Document.Type.PLAIN\_TEXT)
8. .build();
9. // END TODO

content\_copy

1. Use the Natural Language client object to [analyze the sentiment](http://googlecloudplatform.github.io/google-cloud-java/latest/apidocs/com/google/cloud/language/v1/LanguageServiceClient.html#analyzeSentiment(com.google.cloud.language.v1.Document)) of the document, assigning the result to a Sentiment object.
2. // TODO: Use the client to analyze
3. // the sentiment of the feedback
4. Sentiment sentiment = language
5. .analyzeSentiment(doc)
6. .getDocumentSentiment();
7. // END TODO

content\_copy

1. Then, return the sentiment score from the sentiment object.
2. // TODO: Return the sentiment score
3. return sentiment.getScore();
4. // END TODO
5. }

content\_copy

1. Save the file.

### **Write code to use the Natural Language API functionality**

1. Return to the backend/ConsoleApp.java file.
2. Move to the main(...) method.
3. In the main()method, create a SubscriptionName object representing a new subscription named "worker2-subscription".
4. // TODO: Create the Pub/Sub subscription name
5. SubscriptionName subscription =
6. SubscriptionName.create(projectId,
7. "worker2-subscription");
8. // END TODO
9. ```

content\_copy

1. At the point indicated by the comments, create the LanguageService instance using its static create() method.
2. // TODO: Create the languageService
3. LanguageService languageService = LanguageService.create();
4. // END TODO

content\_copy

1. At the point indicated by the comments, use the languageService object to perform sentiment detection on the feedback.
2. // TODO: Use the Natural Language API to analyze sentiment
3. float sentimentScore = languageService.analyzeSentiment(
4. feedback.getFeedback());
5. // END TODO

content\_copy

1. Then, log the score to the console and assign a new score property to the feedback object.
2. // TODO: Set the feedback object sentiment score
3. feedback.setSentimentScore(sentimentScore);
4. System.out.println("Score is: " + sentimentScore);
5. // END TODO

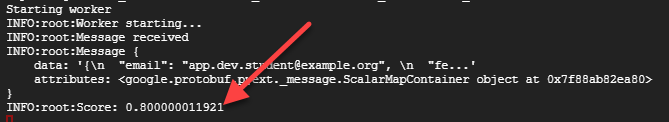
content\_copy

1. Save the file.

### **Run the web and worker application and test the Natural Language API**

1. Return to the first Cloud Shell window and restart the web application.
2. Switch to the second Cloud Shell window and restart the worker application.
3. Preview the web application, then click **Take Test** > **Places**.
4. Answer the questions, select the rating, enter some feedback text, and then click **Send Feedback**.
5. Return to the second Cloud Shell window.

You should see that the worker application has invoked the Cloud Natural Language API and displayed the sentiment score in the console.



1. Stop the web and worker applications.

## Persist Data to Cloud Spanner

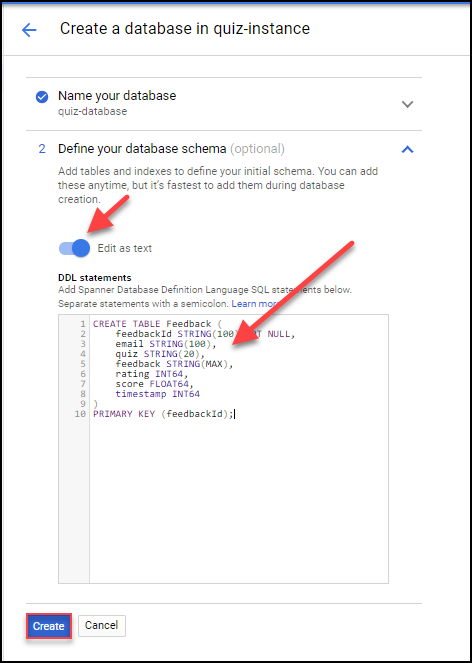
In this section you create a Cloud Spanner instance, database, and table. You then write the code to persist the feedback data into the database.

### **Create a Cloud Spanner instance**

1. Return to the **Cloud Console**.
2. Click **Navigation menu** > **Spanner**.
3. Click **Create instance**.
4. For **Name**, type **quiz-instance**
5. In the **Configuration** section, select **us-central1** as the region.
6. Click **Create**.

### **Create a Cloud Spanner database and table**

1. On the **Instance Details** page for **quiz-instance**, click **Create database**.
2. For **Name**, type **quiz-database**, and then click **Continue**.
3. Under **Define your database schema**, click **Edit as text**.



1. For **DDL statements**, type the following SQL statement:
2. CREATE TABLE Feedback (
3. feedbackId STRING(100) NOT NULL,
4. email STRING(100),
5. quiz STRING(20),
6. feedback STRING(MAX),
7. rating INT64,
8. score FLOAT64,
9. timestamp INT64 )
10. PRIMARY KEY (feedbackId);

content\_copy

1. Click **Create**.

Create an cloud spanner instance and database

Check my progress

### **Write code to persist data into Cloud Spanner**

You can view the API documentation for Cloud Spanner at:

[http://googlecloudplatform.GitHub.io/google-cloud-java/latest/apidocs/com/google/cloud/spanner/package-summary.html](http://googlecloudplatform.github.io/google-cloud-java/latest/apidocs/com/google/cloud/spanner/package-summary.html)

1. Return to the code editor, and move to the insertFeedback(...) method in the **...services/gcp/spanner/SpannerService.java** file.
2. Get a reference to [Cloud Spanner](http://googlecloudplatform.github.io/google-cloud-java/latest/apidocs/com/google/cloud/spanner/Spanner.html).
3. // TODO: Get a reference to the Spanner service
4. SpannerOptions options =
5. SpannerOptions.newBuilder().build();
6. Spanner spanner = options.getService();
7. // END TODO

content\_copy

1. Get a reference to the [Spanner database](http://googlecloudplatform.github.io/google-cloud-java/latest/apidocs/com/google/cloud/spanner/DatabaseId.html) via the Database Id.
2. // TODO: Get a reference to the quiz-instance
3. // and its quiz-database
4. DatabaseId db = DatabaseId.of(
5. options.getProjectId(),
6. "quiz-instance",
7. "quiz-database");
8. // END TODO

content\_copy

1. Get a reference to the [Cloud Spanner Database client](http://googlecloudplatform.github.io/google-cloud-java/latest/apidocs/com/google/cloud/spanner/DatabaseClient.html).
2. // TODO: Get a client for the quiz-database
3. DatabaseClient dbClient =
4. spanner.getDatabaseClient(db);
5. // END TODO

content\_copy

1. Create a new List<Mutation> to reference all the changes that will be made to the database.
2. // TODO: Create a list to hold mutations
3. // against the database
4. List<Mutation> mutations = new ArrayList<>();
5. // END TODO
6. // END TODO

content\_copy

1. Add the Mutation that represents an [insert](http://googlecloudplatform.github.io/google-cloud-java/latest/apidocs/com/google/cloud/spanner/Mutation.html#newInsertBuilder(java.lang.String)) against the feedback table, using data from the feedback object.
2. // TODO: Add an insert mutation
3. mutations.add(
4. // TODO: Build a new insert mutation
5. Mutation.newInsertBuilder("Feedback")
6. .set("feedbackId")
7. .to(feedback.getEmail() + '\_' +
8. feedback.getQuiz() + "\_" +
9. feedback.getTimestamp())
10. .set("email")
11. .to(feedback.getEmail())
12. .set("quiz")
13. .to(feedback.getQuiz())
14. .set("feedback")
15. .to(feedback.getFeedback())
16. .set("rating")
17. .to(feedback.getRating())
18. .set("score")
19. .to(
20. feedback.getSentimentScore())
21. .set("timestamp")
22. .to(feedback.getTimestamp())
23. .build());
24. // END TODO

content\_copy

* 1. Use the database client to write the mutations.

// TODO: Write the change to Spanner

dbClient.write(mutations);

// END TODO

content\_copy

1. Save the file.

### **Write code to use the Cloud Spanner functionality**

1. Move to the main(...) method in the backend/ConsoleApp.java file.
2. In the main()method, create a SubscriptionName object representing a new subscription named "worker3-subscription".
3. // TODO: Create the Pub/Sub subscription name
4. SubscriptionName subscription =
5. SubscriptionName.create(projectId,
6. "worker3-subscription");
7. // END TODO

content\_copy

1. At the point indicated by the comments, create the SpannerService instance.
2. // TODO: Create the spannerService
3. SpannerService spannerService = SpannerService.create();
4. // END TODO

content\_copy

1. At the point indicated by the comments, use the spannerService object to insert the feedback into the database and print out a message to the console.
2. // TODO: Insert the feedback into Cloud Spanner
3. spannerService.insertFeedback(feedback);
4. System.out.println("Feedback saved");
5. // END TODO

content\_copy

1. Save the file.

### **Run the web and worker application and test Cloud Spanner**

1. Return to the first Cloud Shell window, start the web application.
2. Switch to the second Cloud Shell window, restart the worker application.
3. Preview the Quiz application, click **Take Test** > **Places**.
4. Answer the questions, select the rating, enter some feedback text, and then click **Send Feedback**.
5. Return to the second Cloud Shell window.

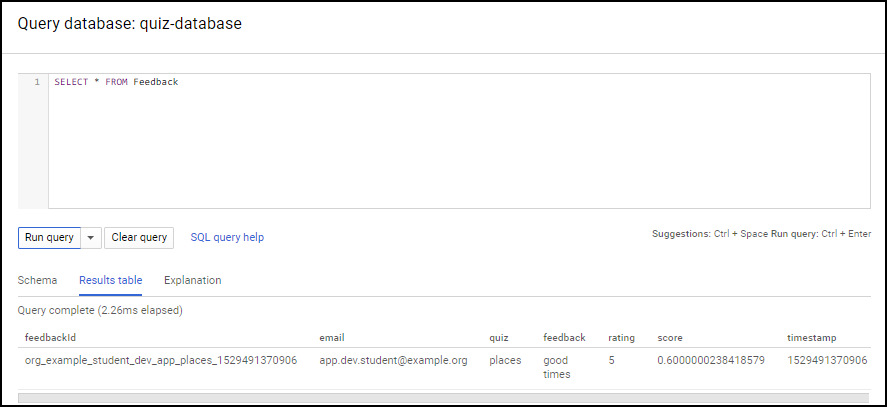
You should see that the worker application has invoked the Cloud Spanner API and displayed the message in the console window.

1. Return to the Console. Click **Navigation menu** > **Spanner**.
2. Select **quiz-instance > quiz-database > Query**.
3. To execute a query, in the **Query** dialog, type SELECT \* FROM Feedback, and then click **Run query**.

SELECT \* FROM Feedback

content\_copy

You should see the new feedback record in the Cloud Spanner database, including the message data from Cloud Pub/Sub and the Quiz score from the Cloud Natural Language API.



# App Dev - Deploying the Application into Kubernetes Engine - Java

50 minutesFree

## GSP171



## Overview

Google Kubernetes Engine provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The environment that Kubernetes Engine provides consists of multiple machines (specifically, Compute Engine instances) grouped together to form a cluster.

Kubernetes provides the mechanisms through which you interact with your cluster. You use Kubernetes commands and resources to deploy and manage your applications, perform administration tasks and set policies, and monitor the health of your deployed workloads.

In this lab, you deploy the Quiz application into Kubernetes Engine, leveraging Google Cloud resources, including Container Builder and Container Registry, and Kubernetes resources, such as Deployments, Pods, and Services.

## Objectives

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

* Create Dockerfiles to package up the Quiz application frontend and backend code for deployment.
* Harness Container Builder to produce Docker images.
* Provision a Kubernetes Engine cluster to host the Quiz application.
* Employ Kubernetes deployments to provision replicated Pods into Kubernetes Engine.
* Leverage a Kubernetes service to provision a load balancer for the quiz frontend.

## Setup and Requirements

### **Qwiklabs setup**

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Prepare the Quiz Application

In this section, you clone the git repository containing the Quiz application, configure environment variables, and run the application.

### **Clone source code in Cloud Shell**

Enter the following command in Cloud Shell to clone the repository for this lab.

git clone https://github.com/GoogleCloudPlatform/training-data-analystcontent\_copy

### **Configure the Quiz application**

Change the working directory:

cd ~/training-data-analyst/courses/developingapps/java/kubernetesengine/start

content\_copy

Configure the Quiz frontend application:

. prepare\_environment.sh

content\_copy

This script file:

* Creates a Google App Engine application.
* Exports environment variables GCLOUD\_PROJECT and GCLOUD\_BUCKET.
* Runs mvn clean install.
* Creates entities in Google Cloud Datastore.
* Creates a Google Cloud Pub/Sub topic.
* Creates a Cloud Spanner Instance, Database, and Table.
* Prints out the Project ID.

Click Check my progress to verify the objective.

Configure the Quiz application

Check my progress

### **Review the code**

Next review and update the Quiz application code in a code editor. You can use the Cloud Shell code editor or the shell editors that are installed on Cloud Shell, such as nano or vim. This lab uses the Cloud Shell code editor.

#### Launch the Cloud Shell text editor

Use the Google Cloud Shell Code Editor to easily create and edit directories and files in the Cloud Shell instance.

Once you activate the Google Cloud Shell, click the **Open editor** > **Open In New Window** to open the Cloud Shell Code Editor.



The code editor launches in a separate tab of your browser.

#### Examine the code and folder structure

In Cloud Shell code editor, navigate to training-data-analyst/courses/developingapps/java/kubernetesengine/start.

In the kubernetesengine folder, notice the end folder. The end folder contains the same files as the start folder, but each file in the end folder contains the complete code required to perform this lab.

The folder structure for the Quiz application now reflects how it's deployed in Kubernetes Engine.

* There is a new folder called frontend. This contains the packaged output for the web application.
* There is a folder called backend. This contains the packaged output for the console application.
* There are configuration files for Docker (a Dockerfile in the frontend and backend folder) and Kubernetes Engine (\*.yaml).

1. Go back to cloud shell UI and click on **Open Terminal**.
2. In Cloud Shell, use the following command to copy the output jar for the frontend application to the frontend folder:
3. cp ./target/quiz-frontend-0.0.1.jar ./frontend/

content\_copy

1. Configure the Quiz backend application:
2. mvn package -f pom-backend.xml

content\_copy

1. Copy the output jar for the backend application to the backend folder:
2. cp ./target/quiz-backend-0.0.1.jar ./backend/

content\_copy

## Creating a Kubernetes Engine Cluster

In this section you create a Google Kubernetes Engine cluster to host the Quiz application.

1. In the Console, click **Navigation menu** > **Kubernetes Engine** > **Clusters**, and then click **Create cluster**.
2. To configure the cluster, use the specified values for the properties listed in the following table; leave the properties not listed at the default values:

|  |  |
| --- | --- |
| **Property** | **Value** |
| Name | quiz-cluster |
| Zone | us-central1-b |
| In the left hand side expand **default-pool** option under **Node pools** and select **Security** | For **Access scopes**, Select **Allow full access to all Cloud APIs** |

1. Click **Create**. The cluster will take a couple of minutes to provision.

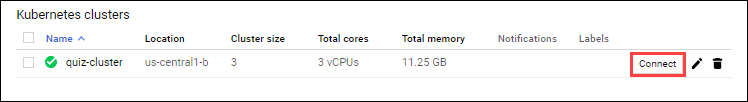
Click Check my progress to verify the objective.

Create Kubernetes engine cluster

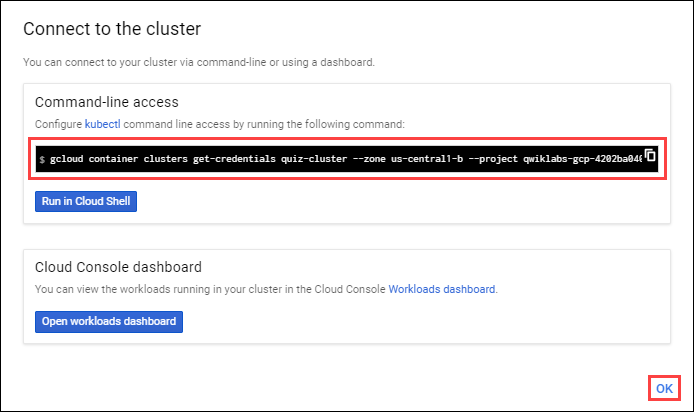
Check my progress

### **Connect to the cluster**

1. After the cluster is ready, click **Connect**:



1. In **Connect to the cluster**, copy the first command to the clipboard, and then click **OK** to close the window:



The command is in the form: gcloud container clusters get-credentials quiz-cluster --zone us-central1-b --project <Project-ID>

1. Paste the command into Cloud Shell and press **Enter**.
2. List the pods in the cluster.
3. kubectl get pods

content\_copy

The response should indicate that there are no pods in the cluster.

This confirms that you have configured security to allow the kubectl command-line tool to perform operations against the cluster.

## Build Docker Images using Container Builder

In this section, you create a Dockerfile for the application frontend and backend and employ Container Builder to build images and store them in the Container Registry.

### **Create the Dockerfile for the frontend**

In Cloud Shell code editor, navigate to training-data-analyst/courses/developingapps/java/kubernetesengine/start and open frontend/Dockerfile.

Copy and paste the following content into frontend/Dockerfile then save the file.

FROM gcr.io/google\_appengine/jetty9

VOLUME /tmp

ADD ./quiz-frontend-0.0.1.jar /app.jar

CMD java -jar /app.jar

content\_copy

##### **What this script does:**

This script is a series of Dockerfile commands.

The first command, FROM gcr.io/google\_appengine/jetty9, initializes the creation of a custom Docker image using the Google App Engine Jetty 9 image, gcr.io/google\_appengine/jetty9 as the starting point.

This second command, VOLUME /tmp, creates a volume in the container's file system with the path of /tmp.

The third command, ADD ./quiz-frontend-0.0.1.jar /app.jar, adds the Jar file, uiz-frontend-0.0.1.jar for the frontend generated by the Maven packaging process as part of the build process.

This fourth and last command, CMD java -jar /app.jar, executes when the container runs.

### **Create the Dockerfile for the backend**

You create the dockerfile for the backend the same way you created the frontend, except the jar file is added to the backend.

In Cloud Shell code editor, navigate to training-data-analyst/courses/developingapps/java/kubernetesengine/start and open backend/Dockerfile.

Copy and paste the following content into backend/Dockerfile then save the file.

FROM gcr.io/google\_appengine/jetty9

VOLUME /tmp

ADD ./quiz-backend-0.0.1.jar /app.jar

CMD java -jar /app.jar

content\_copy

## Build Docker images with Container Builder

1. In Cloud Shell, enter the following command to build the frontend Docker image:
2. gcloud builds submit -t gcr.io/$DEVSHELL\_PROJECT\_ID/quiz-frontend ./frontend/

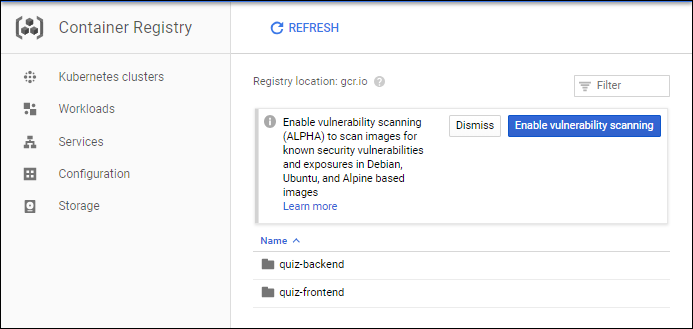
content\_copy

The files are staged into Cloud Storage, and a Docker image will be built and stored in the Container Registry. It will take a few seconds.

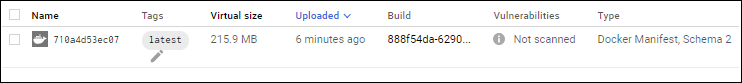
1. Build the backend Docker image:

gcloud builds submit -t gcr.io/$DEVSHELL\_PROJECT\_ID/quiz-backend ./backend/content\_copy

1. In the console, Click **Navigation menu** > **Container Registry**. You should see two folders: **quiz-frontend** and **quiz-backend**.



1. Click **quiz-frontend**. You should see the image name (a hash), tags (latest), size (about 200 MB), and other details.



Click Check my progress to verify the objective.

Build Docker Images using Container Builder

Check my progress

## Create a Kubernetes deployment and service resources

In this section you will modify template yaml files that contain the specification for Kubernetes Deployment and Service resources, and then create the resources in the Kubernetes Engine cluster.

### **Create a Kubernetes Deployment file**

In Cloud Shell code editor, navigate to training-data-analyst/courses/developingapps/java/kubernetesengine/start.

1. In the code editor, open the frontend-deployment.yaml file. The file skeleton has been created for you. Your job is to replace placeholders with values specific to your project.
2. Replace the placeholders in the frontend-deployment.yaml file using the following values:

|  |  |
| --- | --- |
| **Placeholder Name** | **Value** |
| [GCLOUD\_PROJECT] | Project ID (Display the Project ID by entering echo $GCLOUD\_PROJECT in Cloud Shell) |
| [GCLOUD\_BUCKET] | Cloud Storage bucket ID for the media bucket in your project. (Display the bucket name by entering echo $GCLOUD\_BUCKET in Cloud Shell) |
| [FRONTEND\_IMAGE\_IDENTIFIER] | The frontend image identified in the form gcr.io/[Google Cloud\_Project\_ID]/quiz-frontend |

1. The quiz-frontend deployment provisions three replicas of the frontend Docker image in Kubernetes pods, distributed across the three nodes of the Kubernetes Engine cluster.
2. **Save** the file.
3. Open the backend-deployment.yamlfile and replace the placeholders in the backend-deployment.yaml file using the following values:

|  |  |
| --- | --- |
| **Placeholder Name** | **Value** |
| [GCLOUD\_PROJECT] | Project ID |
| [GCLOUD\_BUCKET] | Cloud Storage bucket ID for the media bucket in your project. This will be the same bucket used in frontend-deployment.yaml. (Display the bucket name by entering echo $GCLOUD\_BUCKET in Cloud Shell) |
| [BACKEND\_IMAGE\_IDENTIFIER] | The backend image identified in the form gcr.io/[Google Cloud\_Project\_ID]/quiz-backend |

1. The quiz-backend deployment provisions one replica of the backend Docker image in Kubernetes pods, placed on one of the three nodes of the Kubernetes Engine cluster.
2. **Save** the file.
3. Review the contents of the frontend-service.yaml file.

The service exposes the frontend deployment using a load balancer. The load balancer will send requests from clients to all three replicas of the frontend pod.

### **Execute the Deployment and Service Files**

1. In Cloud Shell, provision the quiz frontend deployment.

kubectl create -f ./frontend-deployment.yamlcontent\_copy

1. Provision the quiz backend deployment.

kubectl create -f ./backend-deployment.yamlcontent\_copy

1. Provision the quiz frontend Service.

kubectl create -f ./frontend-service.yamlcontent\_copy

Each command provisions resources in Kubernetes Engine. It will take a few minutes to complete the process.

Click Check my progress to verify the objective.

Create Kubernetes Deployment and Service Resources

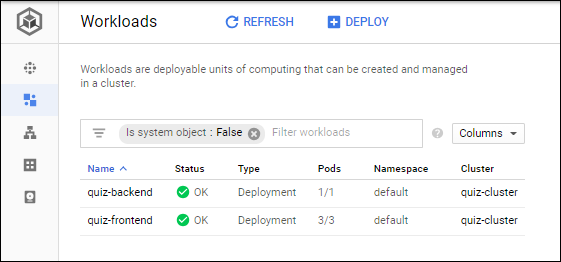
Check my progress

## Test the Quiz Application

In this section review the deployed Pods and Service and navigate to the Quiz application.

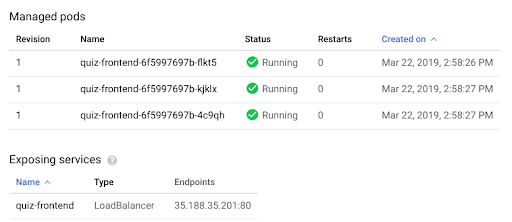
### **Review the deployed resources**

1. In the console, click **Navigation menu** > **Kubernetes Engine** > **Workloads**. You should see two items: quiz-frontend and quiz-backend.



You may see that the pod status is OK or in the process of being created.

1. Click **quiz-frontend**.
2. Scroll down to **Managed pods** to see an overview of quiz-frontend, including the three managed pods and services.



You may see that the quiz-frontend load balancer is being created or is OK. Wait until the Service is OK before continuing. You should see an IP address endpoint when the service is ready.

1. In **Exposing services**, under **Endpoints**, select the IP address and open in the new browser tab.
2. Take a test to verify the application works as expected.

# App Dev - Deploying the Application into App Engine Flexible Environment - Java

1 hourFree

## GSP172



## Overview

An App Engine app is a single application resource with one or more services. Each service can be configured to use different runtimes and to operate with different performance settings. Within each service, you can deploy versions of that service, and each then runs within one or more instances, depending on how much traffic you configured it to handle. For more information, see [an overview of App Engine](https://cloud.google.com/appengine/docs/flexible/java/an-overview-of-app-engine).

App Engine uses either a Standard or Flexible environment. A standard environment runs instances in a sandbox, limiting available CPU options and disc access.

In contrast, a flexible environment runs your application in Docker containers on Compute Engine virtual machines (VMs), which have fewer restrictions. For example, you can use the programming language or library of your choice, write to disk, and even run multiple processes. You also have the choice of Compute Engine machine types for your instances. For more information about App Engine environments, see [The App Engine Standard Environment](https://cloud.google.com/appengine/docs/standard/) and [App Engine Flexible Environment](https://cloud.google.com/appengine/docs/flexible/).

In this lab, you deploy a quiz application into App Engine flexible environment, and leverage App Engine features like versions and traffic splitting.

## Objectives

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

* Create an app.yaml file to describe the App Engine flexible environment requirements for an application.
* Deploy the quiz application into App Engine flexible environment.
* Employ versions and traffic splitting to perform A/B testing of an application feature.

## Setup and Requirements

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Preparing the Case Study Application

In this section, you access Cloud Shell, clone the git repository that contains the Quiz application, configure environment variables, and run the application.

### **Clone source code in Cloud Shell**

Enter the following command to clone the repository for the lab.

git clone https://github.com/GoogleCloudPlatform/training-data-analyst

content\_copy

### **Configure the case study application**

Now change the working directory:

cd ~/training-data-analyst/courses/developingapps/java/appengine/start

content\_copy

You will have to make one small change to your executable file. Open prepare\_environment.sh by running the following command:

nano prepare\_environment.sh

content\_copy

Now find the gcloud beta functions deploy command near the bottom of the file and remove the beta command. Ensure that line of code resembles the following:

gcloud functions deploy process-feedback --runtime nodejs8 --trigger-topic feedback --source ./function --stage-bucket $GCLOUD\_BUCKET --entry-point subscribe

content\_copy

Now exit the nano editor with **CTRL + X** > **Y** > **Enter**.

Run the executable file by by running the following command:

. prepare\_environment.sh

content\_copy

This script file

* Creates an App Engine application.
* Exports environment variables GCLOUD\_PROJECT and GCLOUD\_BUCKET.
* Runs mvn clean install.
* Creates entities in Google Cloud Datastore.
* Creates a Google Cloud Pub/Sub topic.
* Creates a Cloud Spanner Instance, Database, and Table.
* Prints out the Project ID.

When prompted with Allow unauthenticated invocations of new function [process-feedback]?, enter in **Y**.

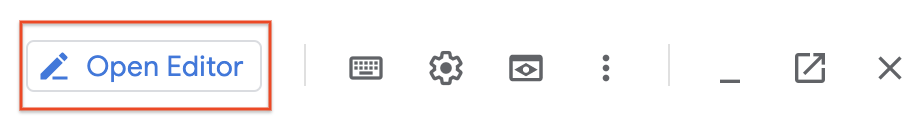
**NOTE:** Please re-run the above command, if you get an error as **(gcloud.functions.deploy) OperationError: code=3, message=Failed to retrieve function source code**

### **Review the code**

In this lab you view and edit files. You can use the shell editors that are installed on Cloud Shell, such as nano or vim, or use the Cloud Shell code editor. This lab uses the Cloud Shell code editor.

#### Launch the Cloud Shell Editor

In the Cloud Platform Console, click **Open editor**. This icon looks like a pencil.



If an error indicates that the code editor could not be loaded because third-party cookies are disabled, click **Open in New Window** and switch to the new tab.

Navigate to **training-data-analyst/courses/developingapps/java/appengine/start**.

The folder structure for the quiz application reflects how it will be deployed in App Engine.

There is a configuration file for App Engine; app.yaml in a new folder, src/main/appengine.

Click Check my progress to verify the objective.

Preparing the Case Study Application

Check my progress

## Preparing Application Code for App Engine Flexible Environment Deployment

In this section, you modify the configuration files for deployment of the quiz application frontend into App Engine flexible environment.

### **Create the app.yaml file for the frontend**

In the Cloud Shell code editor, open src/main/appengine/app.yaml.

The following content describes the App Engine configuration. Copy and paste the content into app.yaml. Be sure you replace [GCLOUD\_BUCKET] with the appropriate bucket name GCLOUD\_PROJECT-media from your project.

#### src/main/appengine/app.yaml

runtime: java

env: flex

runtime\_config:

jdk: openjdk8

handlers:

- url: /.\*

script: this field is required, but ignored

manual\_scaling:

instances: 1

resources:

cpu: 1

memory\_gb: 3.75

disk\_size\_gb: 10

env\_variables:

GCLOUD\_BUCKET: [GCLOUD\_BUCKET]

content\_copy

**Save** the file.

### **Deploy the quiz application to App Engine flexible environment**

Return to the Cloud Shell window. If the Cloud Shell is not visible, click **Open Terminal**.

In Cloud Shell, enter the following command to deploy the quiz application to App Engine flexible environment.

mvn clean compile appengine:deploy

content\_copy

It may take around 10 minutes to complete the deployment.

Maven rebuilds the project and then invokes gcloud app deploy. App Engine automatically packages, containerizes, and deploys the code.

You'll see output similar to the following when the deployment is complete:

INFO\] ----------------------------------------------------------------

\[INFO\] BUILD SUCCESS

\[INFO\] --------------------------------------------------------------

\[INFO\] Total time: 10:21 min

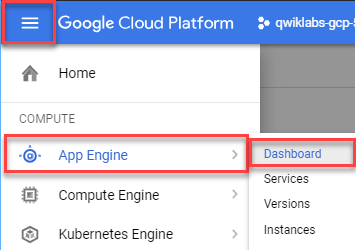
\[INFO\] Finished at: 2018-05-23T19:28:50-04:00

\[INFO\] Final Memory: 37M/90M

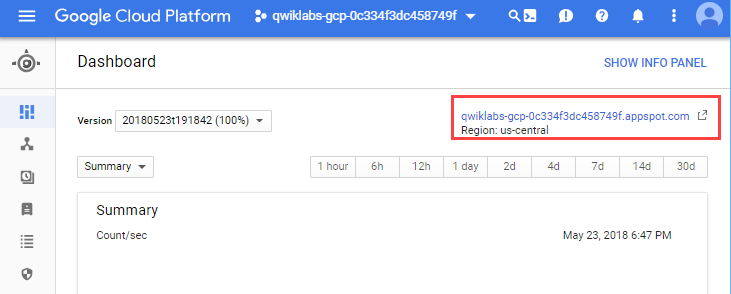
\[INFO\] --------------------------------------------------------------

content\_copy

In the Cloud Platform Console, on the **Navigation menu** > **App Engine** > **Dashboard.**



Click on the link to your application in the top-right corner of the App Engine Dashboard to see your application, the Quite Interesting Quiz.



Click Check my progress to verify the objective.

Deploy the frontend to App Engine Flex

Check my progress

## Updating an App Engine Flexible Environment Application

In this section, you will modify the application code and then redeploy the application.

### **Update the quiz application**

In the Cloud Shell code editor, open the src/main/resources/static/index.html file. Add several exclamation points to the top-level heading.

#### src/main/resources/static/index.html

<!-- This is just a fragment, only add the exclamation points -->

<div class="container">

<h1>Welcome to the Quite Interesting Quiz!!!!!</h1>

<div class="jumbotron">

<p>Welcome to the Quite Interesting Quiz where you can create a question, take a test or review feedback</p>

</div>

content\_copy

This small change stands in for all the changes you might make when updating an application.

Click **File > Save** to save the file.

### **Deploy the updated application**

In **Cloud Shell**, redeploy the App Engine application.

mvn clean compile appengine:deploy \

-Dapp.deploy.stopPreviousVersion=False \

-Dapp.deploy.promote=Falsecontent\_copy

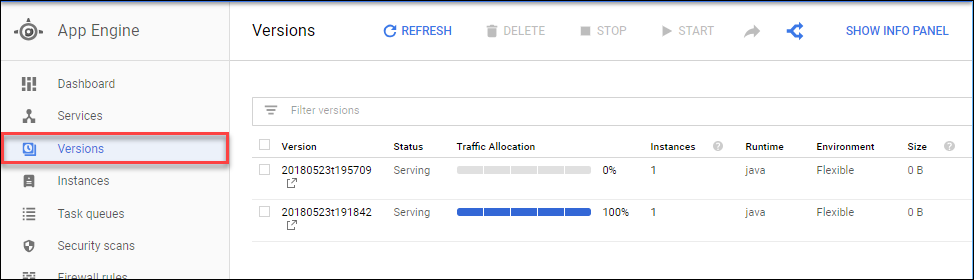
Notice the two additional flags in the command, which means that the previous version will continue to receive traffic.

It may take around 10 minutes for the deployment to update.

In the Console, click **Navigation menu** > **App Engine** > **Dashboard**.

Click on the application URL in the top-right corner of the window. You should see that your application still displays the old title.

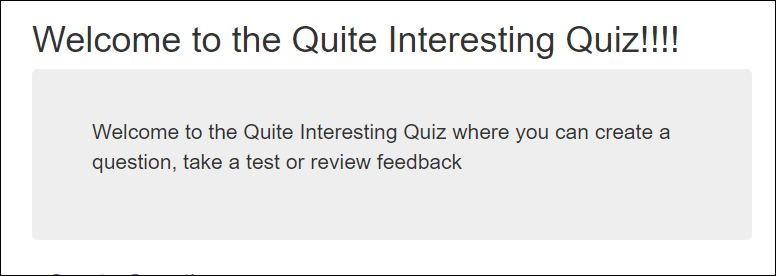
In the **App Engine** window, click **Versions**. You may need to refresh the page until you see two versions of the applications.



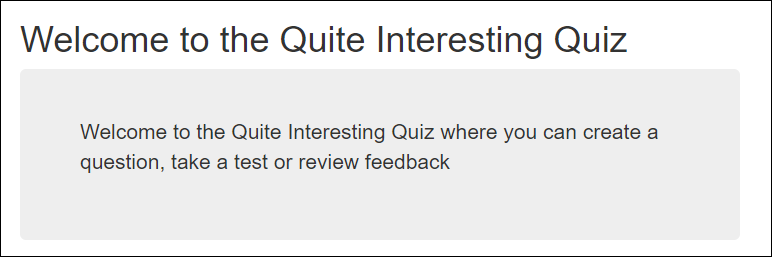
The version ID is in the form 'yyyymmddthhmmss', so it's easy to see which is the new and which is the old version.

Click on both version links to see the new and old version of the quiz application.

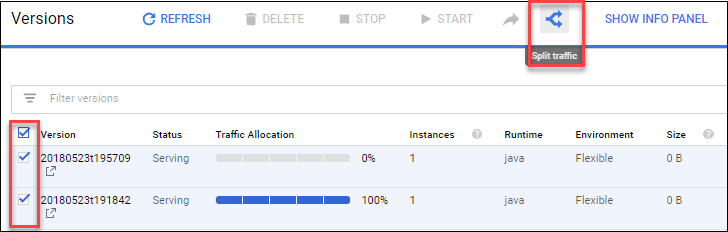
New version (notice those exclamation points!):



Old version:

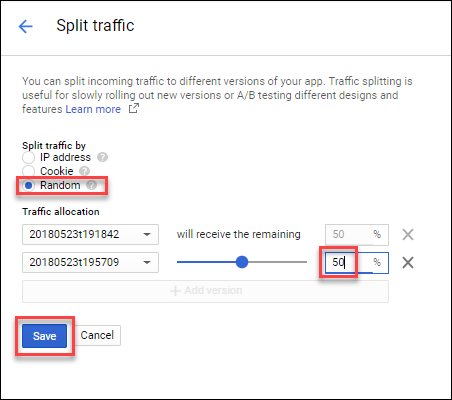


Enable the checkbox for both versions of the application, and click **Split traffic**.



Select the radio button to deliver versions randomly.

Configure the traffic split to deliver 50% of traffic to the old version, and 50% to the new version.



Click **Save**.

Return to the **Dashboard**, set **Version** to All versions and click the application link.



Open a new browser. If possible the browser should be a different type, for example Google Chrome to Internet Explorer. Copy the application link and paste it in the new browser.

Refresh the homepage a few times.

You should see that the homepage displays the old version approximately half the time, and the new version half the time.

In real-world scenarios, you might start by delivering small amounts of traffic to the new version in a canary release, and would use either a cookie or IP address to ensure that a client viewed a single consistent version of the application.

Click Check my progress to verify the objective.

Updating an App Engine Flexible Environment Applicatio

# Cloud Profiler: Qwik Start

45 minutesFree

## GSP209



## Overview

Cloud Profiler is a statistical, low-overhead profiler that continuously gathers CPU usage and memory-allocation information from your production applications. It attributes that information to the application's source code, helping you identify the parts of the application consuming the most resources, and otherwise illuminating the performance characteristics of the code.

In this lab you will learn how to set up and use Stackdriver Profiler. First you'll download a sample Go program and run it with profiling enabled. Then you'll use the Cloud Profiler interface to explore the captured data.

## Setup and Requirements

#### Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

#### What you need

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

#### How to start your lab and sign in to the Google Cloud 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.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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

If you see the **Choose an account** page, click **Use Another Account**. 

1. In the **Sign in** page, 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 Google Cloud account, do not use it for this lab (avoids incurring charges).

1. 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. 

### **Activate Cloud Shell**

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your PROJECT\_ID. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

content\_copy

(Output)

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)content\_copy

(Example output)

Credentialed accounts:

- google1623327\_student@qwiklabs.netcontent\_copy

You can list the project ID with this command:

gcloud config list project

content\_copy

(Output)

[core]

project = <project\_ID>content\_copy

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6content\_copy

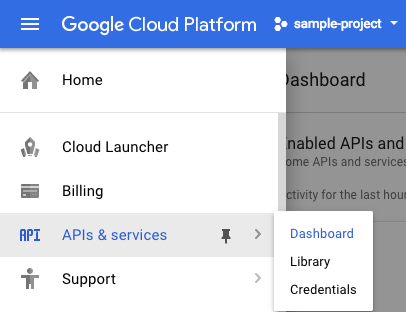
For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud).

## Enable Cloud Profiler API

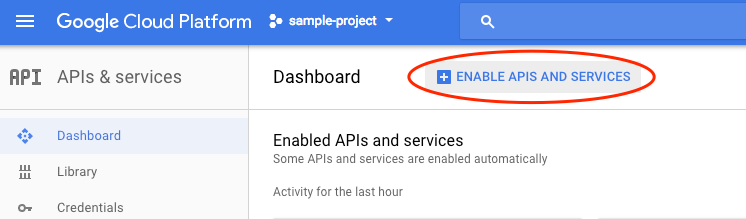
Click on the menu icon in the top left of the screen.



Select **APIs & services** from the drop down and click on **Dashboard**.



Click on **Enable APIs and services**.



Then, search for **Stackdriver Profiler API** in the search box.

Click on **Stackdriver Profiler API**, then click **Enable** if it is not **Enabled**.

## Get a program to profile

The sample program, main.go, is in the [golang-samples](https://github.com/GoogleCloudPlatform/golang-samples) repository on GitHub. To get it, retrieve the package of Go samples:

go get -u github.com/GoogleCloudPlatform/golang-samples/profiler/...

content\_copy

## Profile the code

Go to the directory of sample code for Cloud Profiler in the retrieved package:

cd ~/gopath/src/github.com/GoogleCloudPlatform/golang-samples/profiler/profiler\_quickstart

content\_copy

The main.go program creates a CPU-intensive workload to provide data to the profiler.

Run the following to start the program. You can leave it running:

go run main.go

content\_copy

This program is designed to to load the CPU as it runs, and configured to use Cloud Profiler. Cloud Profiler collects profiling data from the program as it runs and periodically saves it. Progress is indicated with a pair of messages:

2018/06/19 20:38:18 profiler has started

2018/06/19 20:39:00 successfully created profile CPU

2018/06/19 20:39:11 start uploading profile

2018/06/19 20:40:23 successfully created profile CPU

2018/06/19 20:40:33 start uploading profile

2018/06/19 20:41:15 successfully created profile CPU

2018/06/19 20:41:25 start uploading profile

2018/06/19 20:41:45 successfully created profile CPU

...

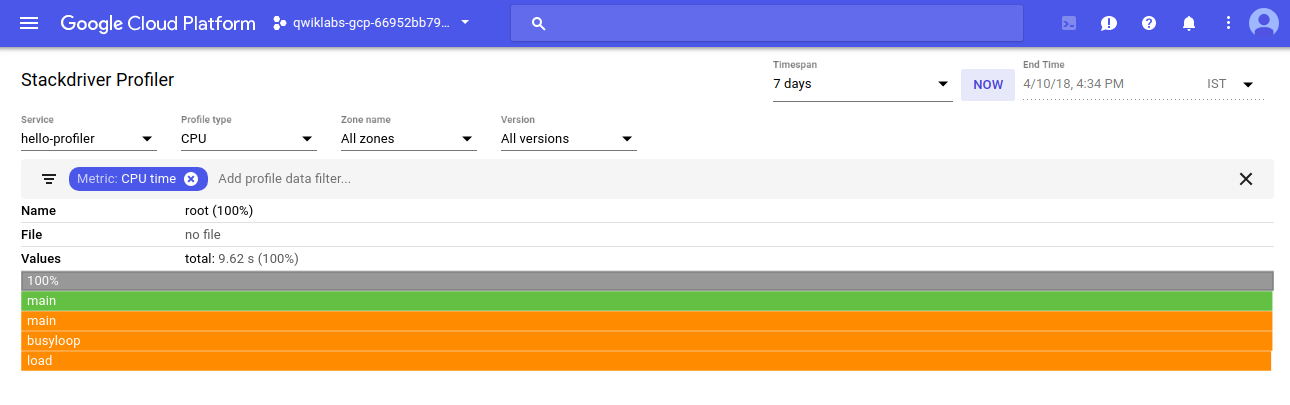
content\_copy

The program will continue to emit these messages while it runs.

Let 2-3 profiles get created, then continue with the lab.

## Start the Profiler interface

In the Console, go to Profiler: from the Navigation menu, in the Cloud section, click on **Profiler**. You will be taken to the Profiler interface:



The interface is divided into two general areas:

* A control area for selecting the data to visualize.
* A flame-graph representation of the selected data.

### **Selecting Profiles**

The interface offers an array of controls for exploring the profiling data. At the top of the interface, there are time controls, so you can examine data for the time range you choose.

Below that are options choosing the set of profile data to use:

* **Service** is for selecting the origin of the profiled data, useful if you are profiling several different applications.
* **Profile** type lets you choose the kind of profile data to display.
* **Zone name** and **Version** let you restrict display to data from Compute Engine zones or versions of the application.

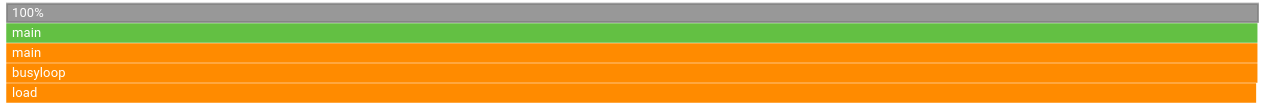
Just below the selectors for Service, Profile, etc. is the filter selector. Filters allow you to refine how the graph displays data. In the screenshot above, the **CPU time** filter is on, so all the CPU time data is displayed.

### **Exploring the data**

Below the selection controls, the selected data is displayed as a flame graph. This type of chart shows you the call stacks in the program. Each function is represented by a frame in the graph, and its relative size shows the proportion of resource consumption that function is responsible for.

The top frame represents the entire program. This frame always shows 100% of the resource consumption, and it indicates how many profiles are averaged together in this graph.

The sample program does not appear to have a complicated set of call stacks; in the preceding screenshot, you see 5 frames:



* The gray frame represents the entire executable, which accounts for 100% of the resources being consumed.
* The green main frame is the Go runtime.main.
* The orange main frame is the main routine of the sample program.
* The orange busyloop frame is a routine called from the sample's main.
* The orange main.load frame is a routine called from the sample's main.

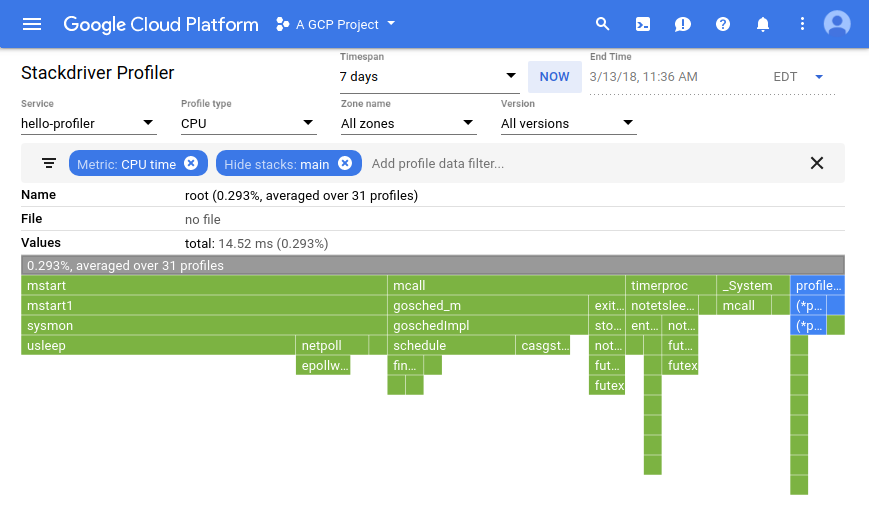
The filter selector lets you do things like filter out functions. For example, if there is a standard library of utility functions, you can remove them from the graph. You can also remove call stacks originating at a certain method, and simplify the graph in other ways.

The main.go application is very simple, so there's not much to filter out, but in a complex application, being able to remove elements from the graph is very useful.

Use a filter to hide the call stack from the main routine to let you see what's happening outside main. This extra work accounts for a tiny 0.29% of the resource consumption, but it makes a much more interesting flame graph.

Click next to the CPU time filter to see other available filters.

Select **Hide stacks** then type in "main" as the value. Your flame graph will look something like this:



The more profiles that get generated, the more interesting your flame graph gets. In a few minutes refresh the Stackdriver Profiler console to see the graph develop. Wait a few more minutes and do it again.