

App Dev: Developing a Backend Service - Java

GSP170



Google Cloud Self-Paced Labs

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.

[Open Google Console](#)

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


Username
google2727032_student@qwiklabs.n 

Password
k68CZxsxMZ 

GCP Project ID
qwiklabs-gcp-4fbfecac8667e457 

[New to labs? View our introductory video!](#)

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




Sign in

Use your Google Account


[Forgot email?](#)


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


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



Choose an account

 Your.Email@gmail.com

 google1381214_student@qwiklabs.net
Signed out

 **Use another account**

Account.

3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

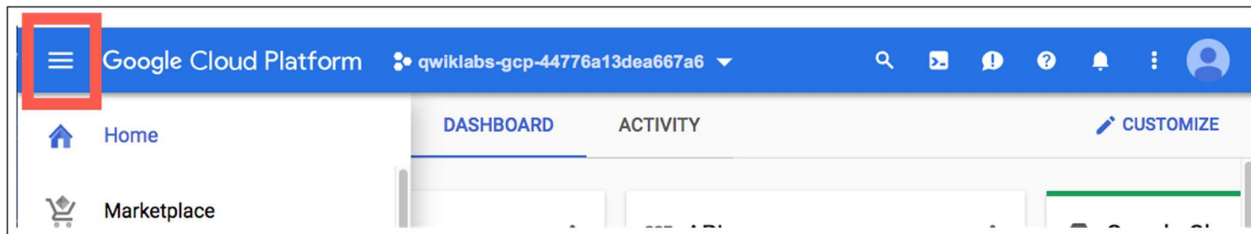
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. 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.

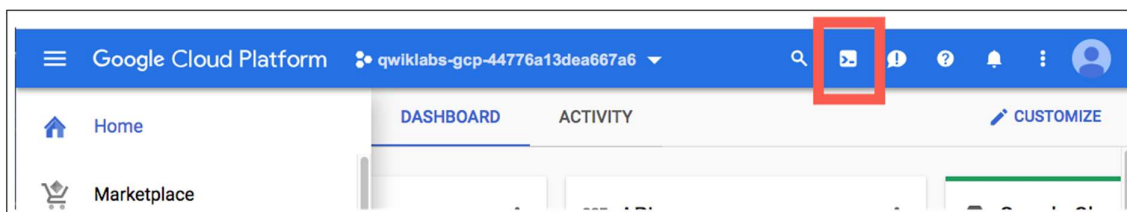


Cloud Shell

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.



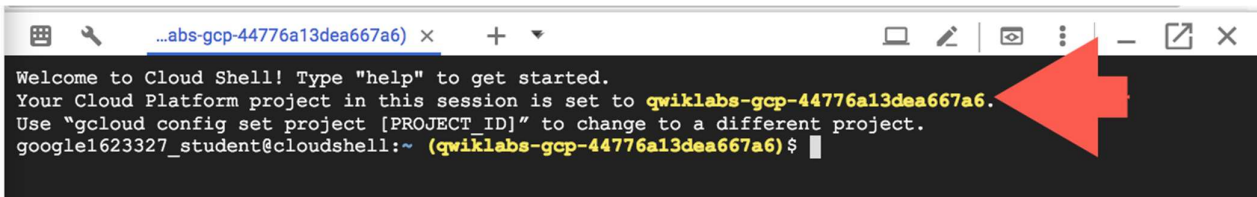
Click **Continue**.

Cloud Shell

Google Cloud Shell provides you with command-line access to your cloud resources directly from your browser. You can easily manage your projects and resources without having to install the Google Cloud SDK or other tools on your system. [Learn more.](#)

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:



```
...abs-gcp-44776a13dea667a6) x + v
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-44776a13dea667a6.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
google1623327_student@cloudshell:~ (qwiklabs-gcp-44776a13dea667a6) $
```

`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
```

(Output)

```
Credentialed accounts:
- <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
- google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
[core]
project = <project_ID>
```

(Example output)

```
[core]
project = quiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the [gcloud command-line tool overview](#).

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
```

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
```

3. Configure the web application:

```
4. . prepare_web_environment.sh
```

This script file:

- Creates an App Engine application.
- Exports environment variables: `GCP_PROJECT` and `GCP_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
```

5. Run the web application:

```
6. mvn spring-boot:run
```

The application is running when 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)
```

Set up the worker application

1. Click **Open a new tab (+)** 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
```

3. Prepare the environment in the second Cloud Shell window:

```
4. . prepare_worker_environment.sh
```

This script file:

- Exports environment variables `GCPLOUD_PROJECT` and `GCPLOUD_BUCKET`.
- Creates and configures a Google Cloud Service Account.
- Prints out the Project ID.

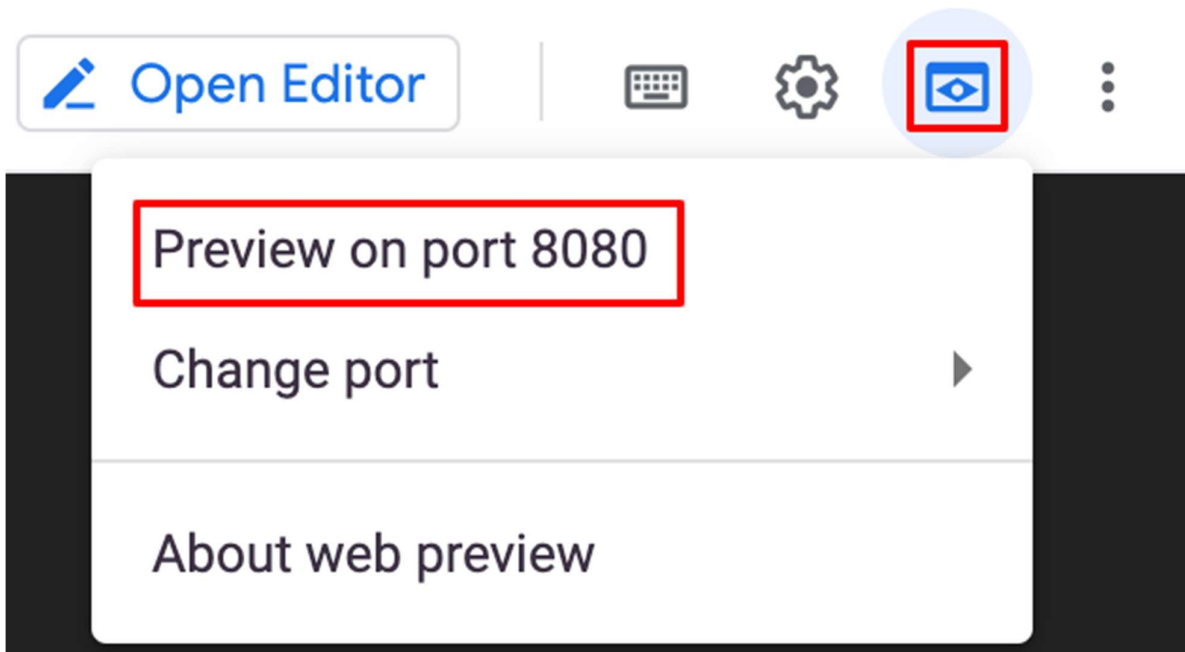
Click *Check my progress* to verify the objective.

5. Now start the worker application:

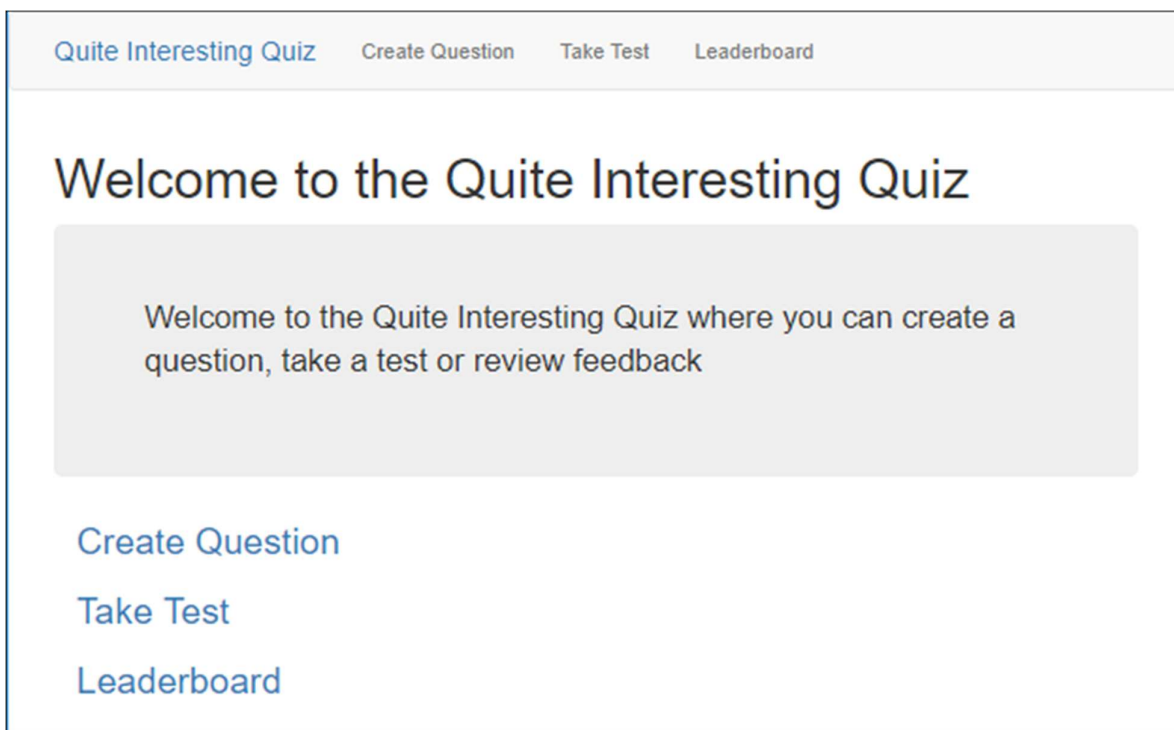
```
6. mvn exec:java@worker
```

Review the Quiz application

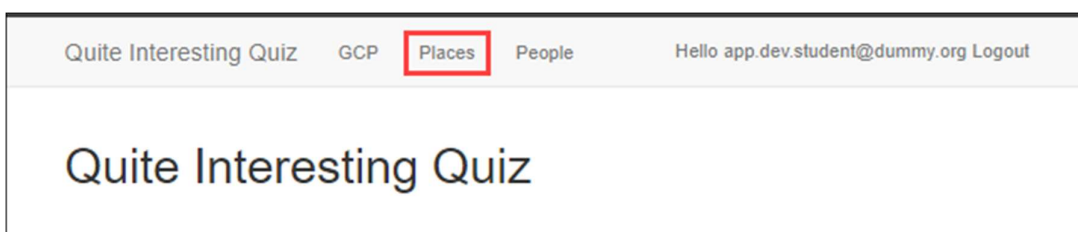
1. Still in the second Cloud Shell window, click **Web preview > Preview on port 8080** to preview the Quiz application.



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



3. Click **Places**.



4. Answer the question.

Quite Interesting Quiz

What is the capital of France?

☐ Berlin

☐ London

☐ Paris

☐ Stockholm

Submit Answer

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

Quite Interesting Quiz

Game Over

You scored 1 out of 1

What did you think?

Please let us know how you found the quiz!

Send Feedback

Quiz takers can select a rating and enter feedback.

5. Return to the 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. Then click **Open in a new window** to open the Cloud Shell Code Editor.

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 `.../api` folder.

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.

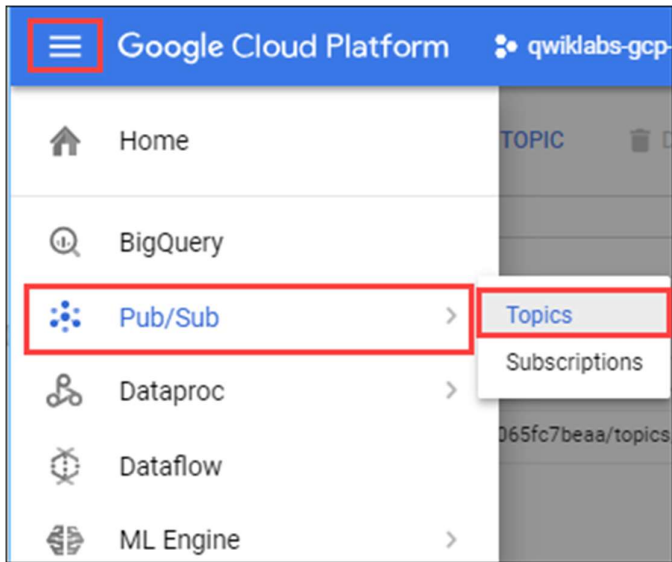
Working with Cloud Pub/Sub

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

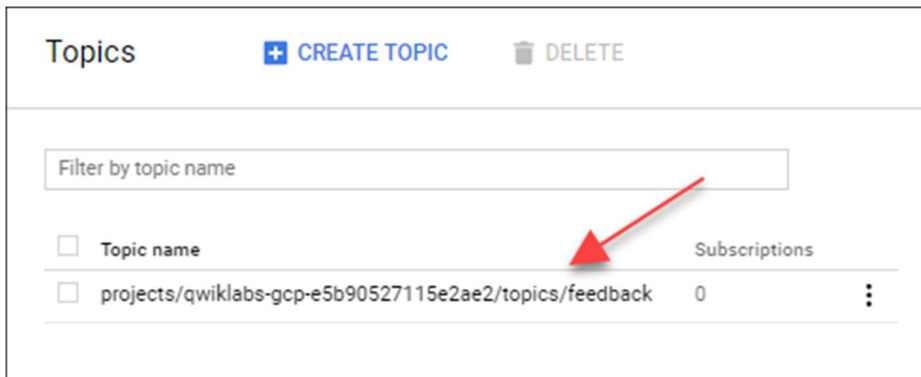
In this section, you create a Cloud Pub/Sub topic and subscription in your Google Cloud project, then publish and retrieve a message.

Create a Cloud Pub/Sub topic

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



2. 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
```

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.

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"
```

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
```

Output:

DATA	MESSAGE_ID	ATTRIBUTES
Hello World	109626632939210	

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`.

```
// TODO: Declare and initialize two Strings,
// PROJECT_ID and TOPIC_NAME

private static final String PROJECT_ID =
    ServiceOptions.getDefaultProjectId();
private static final String TOPIC_NAME = "feedback";

// END TODO
```

2. 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
```

3. Declare a `Publisher` object and set it to `null`. It will be initialized in the try block that follows.

```
// TODO: Declare a publisher for the topic

Publisher publisher = null;
```

```
// END TODO
```

4. Move to the try block, and initialize the publisher object using its builder.

```
// TODO: Initialize the publisher
// using a builder and the topicName

publisher = Publisher.newBuilder(topicName).build();

// END TODO
```

5. Copy the JSON serialized `feedbackMessage` string to a `ByteString`.

```
// TODO: Copy the serialized message
// to a byte string

ByteString data = ByteString.copyFromUtf8(
    feedbackMessage);

// END TODO
```

6. Declare a `PubsubMessage` object; initialize the message using its builder.

```
// TODO: Create a Pub/Sub message using a
// builder; set the message data

PubsubMessage pubsubMessage = PubsubMessage
    .newBuilder().setData(data).build();

// END TODO
```

7. Use the publisher to publish the message, assign the return value to the message ID future object.

```
// TODO: Publish the message,
// assign to the messageIdFuture

messageIdFuture = publisher.publish(
    pubsubMessage);

// END TODO
```

8. Move to the finally block and retrieve the Pub/Sub messageId from the message ID future object.

```
// TODO: Get the messageId from
// the messageIdFuture

String messageId = messageIdFuture.get();

// END TODO
```

9. Complete the publishing code by shutting down the publisher.

```
// TODO: Shutdown the publisher
// to free up resources
if (publisher != null) {
    publisher.shutdown();
}

// END TODO
```

10. 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.

```
// TODO: Declare the publishService

@Autowired
private PublishService publishService;

// END TODO
```

2. In the `processFeedback(...)` method that handles POST requests to the `'/feedback/:quiz'` route, invoke the `publishService.publishFeedback(feedback)` method.

```
// TODO: Publish the feedback to Pub/Sub

publishService.publishFeedback(feedback);

// END TODO
```

3. 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
```

3. Preview the web application, click **Take Test > Places**.

4. Answer the question, select the rating, enter some feedback text, and click **Send Feedback**.

5. In the second Cloud Shell window, to pull a message from the `cloud-shell-subscription`:

```
6. gcloud beta pubsub subscriptions pull cloud-shell-subscription --auto-ack
```

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.
4.      SubscriptionName subscription =
5.          SubscriptionName.create(projectId,
6.                                  "worker1-subscription");
7.
8.      // END TODO
```

9. 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 name and topic

    subscriptionAdminClient.createSubscription(
        subscription, topic,
        PushConfig.getDefaultInstance(), 0);

    // END TODO

}

// END TODO
```

10. Move to the code that creates a `MessageReceiver`, and in the `receiveMessage(...)` override, extract the message data into a `String`.

```
11.      // TODO: Extract the message data as a JSON String
12.
```

```
13.         String fb = message.getData().toStringUtf8();
14.
15.         // END TODO
```

16. Use the consumer to acknowledge the message.

```
17.         // TODO: Ack the message
18.
19.         consumer.ack();
20.
21.         // END TODO
```

22. After the code that initializes an `ObjectMapper`, deserialize the JSON String message data into a feedback object.

```
23.         // TODO: Deserialize the JSON String
24.         // representing the feedback
25.         // Print out the feedback
26.
27.         Feedback feedback = mapper.readValue(
28.             fb, Feedback.class);
29.         System.out.println("Feedback received: "
30.             + feedback);
31.
32.         // END TODO
```

33. After the block that creates the `MessageReceiver`, declare a `Subscriber` and initialize it to null.

```
34.         // TODO: Declare a subscriber
35.
36.         Subscriber subscriber = null;
37.
38.         // END TODO
```

39. Move to the try block, and initialize the `Subscriber` using its default builder. This requires the subscription and receiver.

```
40.         // TODO: Initialize the subscriber using
41.         // its default builder
42.         // with a subscription and receiver
43.
44.         subscriber = Subscriber.defaultBuilder(
45.             subscription, receiver).build();
46.
47.         // END TODO
```

48. Add a listener to the subscriber to display errors.

```
49.         // TODO: Add a listener to the subscriber
50.
51.         subscriber.addListener(
52.             new Subscriber.Listener() {
53.                 @Override
54.                 public void failed(
55.                     Subscriber.State from,
56.                     Throwable failure) {
57.                     System.err.println(failure);
58.                 }
59.             },
60.             MoreExecutors.directExecutor());
61.
```

```
62. // END TODO
```

63. Start the subscriber.

```
64. // TODO: Start subscribing
65.
66.     subscriber.startAsync().awaitRunning();
67.
68. // END TODO
```

69. Move to the finally block. Write the code to stop the subscriber, and delete the subscription.

```
70. // TODO: Stop subscribing
71.
72.     if (subscriber != null) {
73.         subscriber.stopAsync().awaitTerminated();
74.     }
75.
76. // END TODO
77.
78. // TODO: Delete the subscription
79.
80.     try (SubscriptionAdminClient
81.         subscriptionAdminClient =
82.             SubscriptionAdminClient.create()) {
83.
84.         subscriptionAdminClient.deleteSubscription(
85.             subscription);
86.     }
87.
88. // END TODO
```

89. 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
```

3. In the second Cloud Shell window, start the worker application.

```
4. mvn compile exec:java@worker
```

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

6. Click **Take Test**.

7. Click **Places**.

8. Answer the question, select the rating, enter some feedback text, and then click **Send Feedback**.
 9. 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
- ```
10. Feedback received: Feedback{email='app.dev.student@dummy.org', quiz='places',
 feedback='love the test', rating=5, timestamp=1527564677609, sentimentScore=0.0}
```
11. 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](#).

### 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.
5. try (LanguageServiceClient language =
6. LanguageServiceClient.create()) {
```

7. Create a new `Document` object using its builder. Configure this object with the document content and type.

```
8. // TODO: Create a new Document object
9. // using the builder
10. // Set the content and type
11.
12. Document doc = Document.newBuilder()
13. .setContent(feedback)
14. .setType(Document.Type.PLAIN_TEXT)
15. .build();
16.
```

```
17. // END TODO
```

18. Use the Natural Language client object to [analyze the sentiment](#) of the document, assigning the result to a `Sentiment` object.

```
19. // TODO: Use the client to analyze
20. // the sentiment of the feedback
21.
22. Sentiment sentiment = language
23. .analyzeSentiment(doc)
24. .getDocumentSentiment();
25.
26. // END TODO
```

27. Then, return the sentiment score from the sentiment object.

```
28. // TODO: Return the sentiment score
29.
30. return sentiment.getScore();
31.
32. // END TODO
33. }
```

34. 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.
6. SubscriptionName subscription =
7. SubscriptionName.create(projectId,
8. "worker2-subscription");
9.
10. // END TODO
11. ``
12.
```

13. At the point indicated by the comments, create the `LanguageService` instance using its static `create()` method.

```
14. // TODO: Create the languageService
15.
16. LanguageService languageService = LanguageService.create();
17.
18. // END TODO
```

19. At the point indicated by the comments, use the `languageService` object to perform sentiment detection on the feedback.

```

20. // TODO: Use the Natural Language API to analyze sentiment
21.
22. float sentimentScore = languageService.analyzeSentiment(
23. feedback.getFeedback());
24.
25. // END TODO

```

26. Then, log the score to the console and assign a new score property to the feedback object.

```

27. // TODO: Set the feedback object sentiment score
28.
29. feedback.setSentimentScore(sentimentScore);
30. System.out.println("Score is: " + sentimentScore);
31.
32. // END TODO

```

33. Save the file.

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

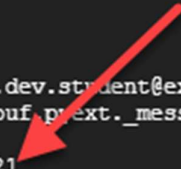
1. Return to the first Cloud Shell window and restart the web application.
2. Switch to the second Cloud Shell window and restart the worker application.
3. Preview the web application, then click **Take Test > Places**.
4. Answer the questions, select the rating, enter some feedback text, and then click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

```

Starting worker
INFO:root:Worker starting...
INFO:root:Message received
INFO:root:Message {
 data: '{\n "email": "app.dev.student@example.org", \n "fe...'
 attributes: <google.protobuf.pyext._message.ScalarMapContainer object at 0x7f88ab82ea80>
}
INFO:root:Score: 0.800000011921

```



6. 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 **Instance 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 **Database name**, type **quiz-database**.
3. Under **Define your schema**, for **DDL statements**, type the following SQL statement:

```
4. CREATE TABLE Feedback (
5. feedbackId STRING(100) NOT NULL,
6. email STRING(100),
7. quiz STRING(20),
8. feedback STRING(MAX),
9. rating INT64,
10. score FLOAT64,
11. timestamp INT64)
12. PRIMARY KEY (feedbackId);
```



## ← Create a database in quiz-instance

### Name your database

Enter a permanent name for your database of at least two characters, starting with a letter.

Database name \*

quiz-database

Lowercase letters, numbers, hyphens, underscores allowed

### Define your schema (optional)

Add Spanner Data Definition Language SQL statements below. Separate statements with a semicolon. [Learn more](#)

```
1 CREATE TABLE Feedback (
2 feedbackId STRING(100) NOT NULL,
3 email STRING(100),
4 quiz STRING(20),
5 feedback STRING(MAX),
6 rating INT64,
7 score FLOAT64,
8 timestamp INT64)
9 PRIMARY KEY (feedbackId);
```

✓ SHOW ENCRYPTION OPTIONS

CREATE

CANCEL

13. Click **Create**.

## Write code to persist data into Cloud Spanner

You can view the API documentation for Cloud Spanner at:

<https://cloud.google.com/spanner/docs/apis>

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](#).

```
3. // TODO: Get a reference to the Spanner service
4.
5. SpannerOptions options =
```

```

6. SpannerOptions.newBuilder().build();
7. Spanner spanner = options.getService();
8.
9. // END TODO

```

10. Get a reference to the [Spanner database](#) via the Database Id.

```

11. // TODO: Get a reference to the quiz-instance
12. // and its quiz-database
13.
14. DatabaseId db = DatabaseId.of(
15. options.getProjectId(),
16. "quiz-instance",
17. "quiz-database");
18.
19. // END TODO

```

20. Get a reference to the [Cloud Spanner Database client](#).

```

21. // TODO: Get a client for the quiz-database
22.
23. DatabaseClient dbClient =
24. spanner.getDatabaseClient(db);
25.
26. // END TODO

```

27. Create a new `List<Mutation>` to reference all the changes that will be made to the database.

```

28. // TODO: Create a list to hold mutations
29. // against the database
30.
31. List<Mutation> mutations = new ArrayList<>();
32.
33. // END TODO

```

34. Add the `Mutation` that represents an [insert](#) against the feedback table, using data from the feedback object.

```

35. // TODO: Add an insert mutation
36.
37. mutations.add(
38. // TODO: Build a new insert mutation
39.
40. Mutation.newInsertBuilder("Feedback")
41. .set("feedbackId")
42. .to(feedback.getEmail() + '_' +
43. feedback.getQuiz() + "_" +
44. feedback.getTimestamp())
45. .set("email")
46. .to(feedback.getEmail())
47. .set("quiz")
48. .to(feedback.getQuiz())
49. .set("feedback")
50. .to(feedback.getFeedback())
51. .set("rating")
52. .to(feedback.getRating())
53. .set("score")
54. .to(
55. feedback.getSentimentScore())
56. .set("timestamp")
57. .to(feedback.getTimestamp())
58. .build());
59.
60. // END TODO

```

61. Use the database client to write the mutations.

```
62. // TODO: Write the change to Spanner
63.
64. dbClient.write(mutations);
65.
66. // END TODO
```

67. 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.
5. SubscriptionName subscription =
6. SubscriptionName.create(projectId,
7. "worker3-subscription");
8.
9. // END TODO
```

10. At the point indicated by the comments, create the `SpannerService` instance.

```
11. // TODO: Create the spannerService
12.
13. SpannerService spannerService = SpannerService.create();
14.
15. // END TODO
```

16. 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.

```
17. // TODO: Insert the feedback into Cloud Spanner
18.
19. spannerService.insertFeedback(feedback);
20. System.out.println("Feedback saved");
21.
22. // END TODO
```

23. 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.

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

```
SELECT * FROM Feedback
```

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.

All Instances >

INSTANCE  
quiz-instance: Overview >

DATABASE  
quiz-database: Query

▶

RUN

▼

CLEAR QUERY

FORMAT QUERY

SHORTCUTS

[SQL query help](#)

1

SELECT \* FROM Feedback

SCHEMA

RESULTS

EXPLANATION

Query completed (9 ms elapsed)

| feedbackId                                      | email                      | quiz   | feedback | rating | score | timestamp     |
|-------------------------------------------------|----------------------------|--------|----------|--------|-------|---------------|
| app.dev.student@dummys.org_places_1618330609749 | app.dev.student@dummys.org | places | ok       | 2      | 0.7   | 1618330609749 |

# Congratulations!

You've developed a backend service for an online Quiz application to process user feedback and save scores.

## Finish your Quest



This lab is part of the [Application Development - Java](#) and [Cloud Development](#) Quests. A Quest is a series of related labs that form a learning path. Completing this Quest earns you the badge above, to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in this Quest and get immediate completion credit if you've taken this lab. See other available [Qwiklabs Quests](#).

## Next steps /learn more

- Learn more about [Backend Services](#).
- Check out more cloud services, see [About the Google Cloud Services](#)

Manual last updated April 13, 2021

Lab last tested April 13, 2021

Copyright 2021 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.