App Dev: Developing a Backend Service - Python

GSP187

Google Cloud Self-Paced Labs

Overview

Google App Engine lets you manage resources from the command line, debug source code in production and run API backends. This lab concentrates on the backend service, putting together Pub/Sub, Natural Language, and Spanner services and APIs to collect and analyze feedback and scores from an online Quiz application.

Objectives

In this lab, you 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.
- Use the Cloud Natural Language Machine Learning API.
- Create and configure a Cloud Spanner database instance, 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:

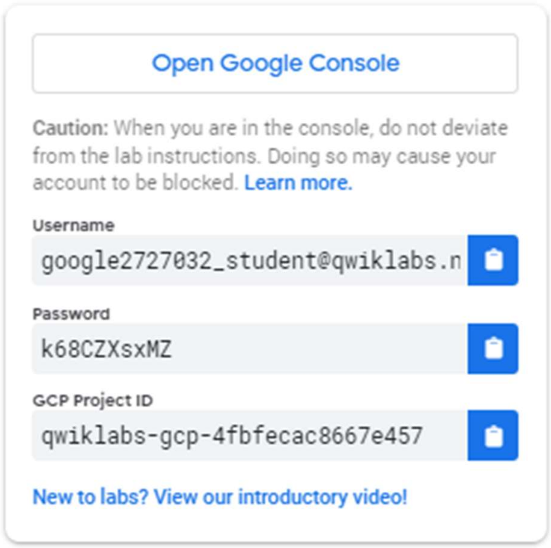
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
  Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

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.



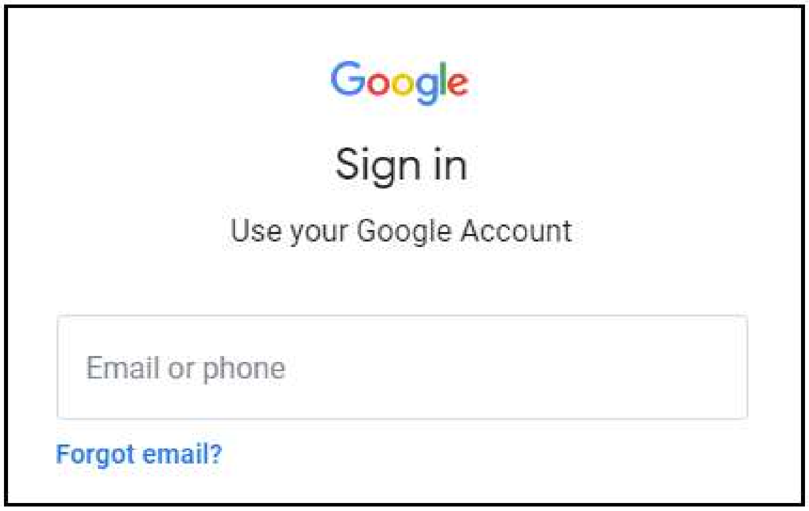
Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username
google2727032_student@qwiklabs.n

Password
k68CZXsxMZ

GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

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

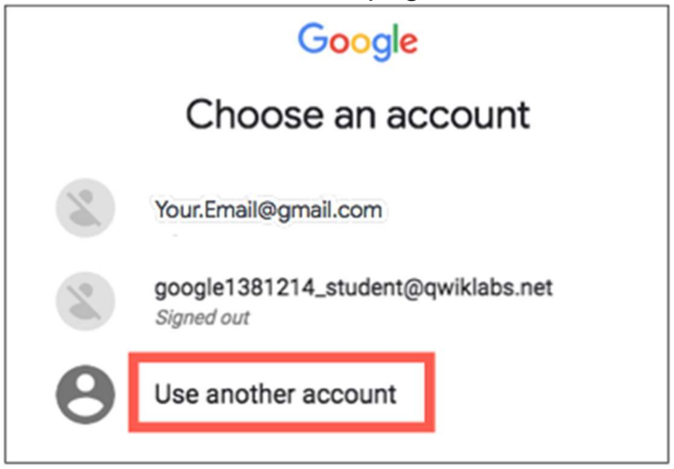
Use your Google Account

Email or phone

Forgot email?

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

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


Choose an account

Your.Email@gmail.com

google1381214_student@qwiklabs.net

Signed out

Use another account

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

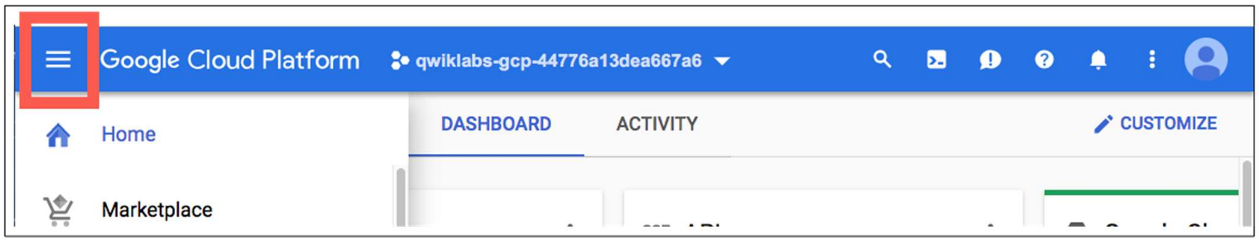
   Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

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.

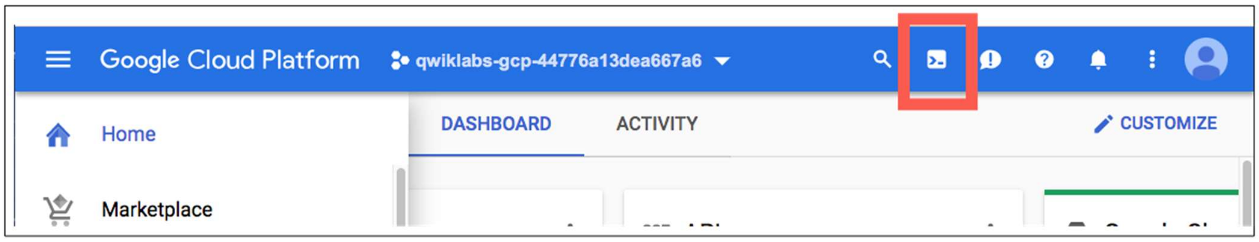


Cloud Shell

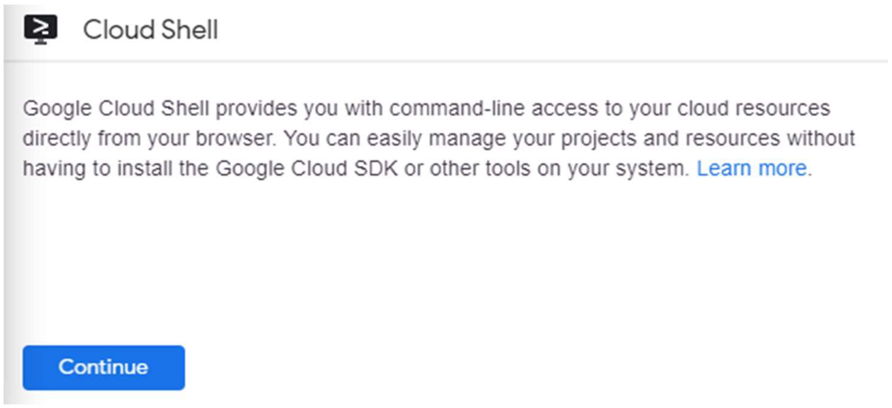
Activate Cloud Shell

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.

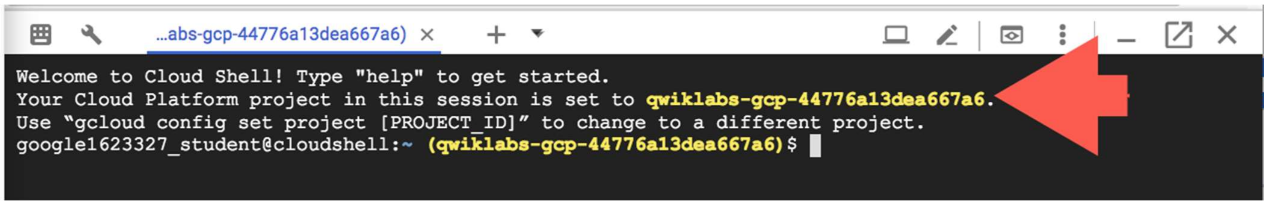


Click **Continue**.



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.

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:



```
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>
```

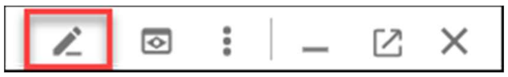
(Example output)

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

For full documentation of `gcloud` see the gcloud command-line tool overview.

Launch the Cloud Shell code editor

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.

Run the following command to configure your Project ID, replacing `YOUR-PROJECT-ID` with your Qwiklabs Project ID:

```
gcloud config set project <YOUR-PROJECT-ID>
```

Prepare the Quiz Application

In this section, you access Cloud Shell and enter commands to:

- Clone the git repository containing the Quiz application
- Configure environment variables
- Run the application

Clone source code in Cloud Shell

Clone the repository for the class:

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

Configure and run the Quiz application

In this section you'll open two Cloud Shell windows, one for the web part of the Quiz application, the other the worker part of the application that handles the console.

1. Change the working directory:

```
2. cd ~/training-data-analyst/courses/developingapps/v1.2/python/pubsub-languageapi-
   spanner/start
```

3. Enter a script file to configure the web application:

   This script file:

   - Creates an App Engine application.
   - Exports environment variables: `GCLOUD_PROJECT` and `GCLOUD_BUCKET`.
   - Updates pip then runs pip install -r requirements.txt.
   - Creates entities in Cloud Datastore.
   - Prints out the Project ID.

```
. prepare_web_environment.sh
```

Ignore the incompatibility messages.

Click *Check my progress* to verify the objective.

4. Run the web application:

```
5. python run_server.py
```

The application is running when you see a message similar to the example output:

Example output

```
* Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 110-448-781
```

6. Click **Add Cloud Shell session** (**+**) on the right of the Cloud Shell tab to add a second Cloud Shell window. This window runs the Worker (console) application.

7. In the second window, change the working directory:

```
8. cd ~/training-data-analyst/courses/developingapps/v1.2/python/pubsub-languageapi-
   spanner/start
```

9. Run the worker application in the second Cloud Shell window:

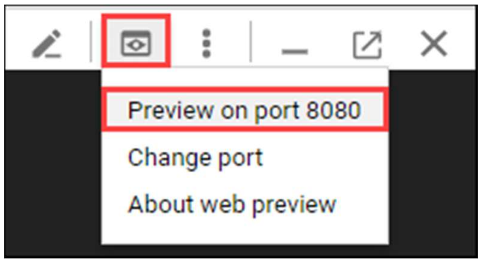
```
10.  . run_worker.sh
```

This script file:

- Exports environment variables `GCLOUD_PROJECT` and `GCLOUD_BUCKET`.
- Creates and configures a Google Cloud Service Account.
- Prints out the Project ID.
- Runs the worker application `python -m quiz.console.worker`.

Check out the Quiz application

1. In **Cloud Shell**, 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.

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

   You can put information in the form, but the **Send Feedback** button does not yet work.

5. Return to the first Cloud Shell window, and press **Ctrl+c** to stop the web application.

Examine the Quiz application code

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 the Cloud Shell code editor.

This lab uses the Cloud Shell code editor to review the Quiz application code.

Review the Google Cloud application code structure

1. Navigate to the `/training-data-analyst/courses/developingapps/v1.2/python/pubsub-languageapi-spanner/start` folder using the file browser panel on the left side of the editor.

2. Select the `pubsub.py` file in the `.../quiz/gcp` folder.

   This file contains a module that allows applications to publish feedback messages to a Cloud Pub/Sub topic and register a callback to receive messages from a Cloud Pub/Sub subscription.

3. Select the `languageapi.py` file in the `.../quiz/gcp` folder.

   This file contains a module that allows users to send text to the Cloud Natural Language ML API and to receive the sentiment score from the API.

4. Select the `spanner.py` file.

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

Review the web application code

1. Select the `api.py` file in the `.../quiz/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.

2. Select the `worker.py` file in the `.../quiz/console` folder.

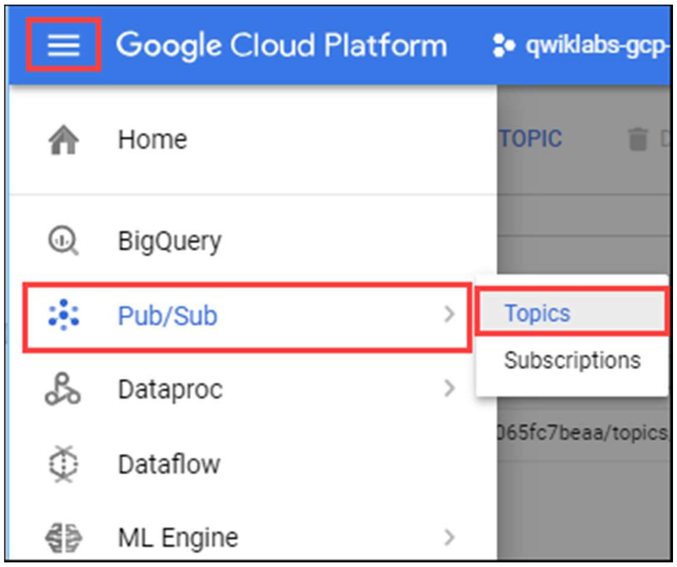
   This file runs as a separate console application to consume the messages delivered to a Pub/Sub subscription.

Work with Cloud Pub/Sub

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

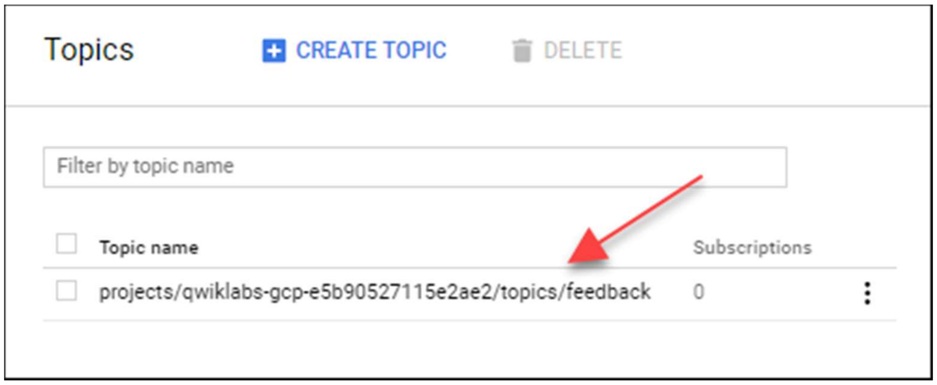
Create a Cloud Pub/Sub topic

1. In the Cloud Platform Console, click **Navigation menu** > **Pub/Sub** > **Topics**.



2. Click **CREATE TOPIC**.

3. For **Topic ID**, type `feedback`, and then click **CREATE TOPIC**.



Topic name	Subscriptions
projects/qwiklabs-gcp-e5b90527115e2ae2/topics/feedback	0

Create a Cloud Pub/Sub subscription

1. Return to the second Cloud Shell window and press **Ctrl+c** to stop the application.

2. Create a Cloud Pub/Sub subscription named `worker-subscription` against the `feedback` topic:

   ```
   gcloud pubsub subscriptions create worker-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:

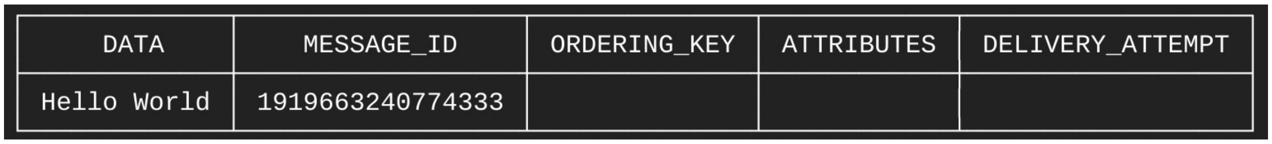
```
gcloud pubsub topics publish feedback --message "Hello World"
```

Retrieve a message from a Cloud Pub/Sub subscription

Now pull the message from the `feedback` topic with automatic acknowledgement of the message:

```
gcloud beta pubsub subscriptions pull worker-subscription --auto-ack
```

Output:



```
┌─────────────┬──────────────────┬──────────────┬────────────┬──────────────────┐
│    DATA     │    MESSAGE_ID    │ ORDERING_KEY │ ATTRIBUTES │ DELIVERY_ATTEMPT │
├─────────────┼──────────────────┼──────────────┼────────────┼──────────────────┤
│ Hello World │ 1919663240774333 │              │            │                  │
└─────────────┴──────────────────┴──────────────┴────────────┴──────────────────┘
```

Publish Messages to Cloud Pub/Sub Programmatically

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.

Import and use the Python Cloud Pub/Sub module

In this section, you'll update `...quiz/gcp/pubsub.py` to do the following:

1. Open the `...quiz/gcp/pubsub.py` file in the editor.
2. Load the `pubsub_v1` module from the `google.cloud` package.
3. Construct a Cloud Pub/Sub Publisher client.
4. Get the fully qualified path referencing the feedback Pub/Sub topic you created earlier.

quiz/gcp/pubsub.py

```
# TODO: Load the Cloud Pub/Sub module

from google.cloud import pubsub_v1

# END TODO

# TODO: Create a Pub/Sub Publisher Client

publisher = pubsub_v1.PublisherClient()

# END TODO

# TODO: Create Topic Object to reference feedback topic

topic_path = publisher.topic_path(project_id, 'feedback')

# END TODO
```

Write code to publish a message to Cloud Pub/Sub

1. In the `publish_feedback(feedback)` function, publish a message to the feedback topic.

quiz/gcp/pubsub.py

```
"""
Publishes feedback info
- jsonify feedback object
- encode as bytestring
- publish message
- return result
"""
def publish_feedback(feedback):

# TODO: Publish the feedback object to the feedback topic

    payload = json.dumps(feedback, indent=2,
                                     sort_keys=True)
    data = payload.encode('utf-8')
    future = publisher.publish(topic_path, data=data)
    return future.result()

# END TODO
```

2. Save the file.

Write code to use the Pub/Sub publish functionality

1. In the `.../quiz/api/api.py` file, load the `pubsub` module from the `quiz.gcp` package.

quiz/api/api.py

```
# TODO: Add pubsub to import list

from quiz.gcp import datastore, pubsub

# END TODO
```

2. In the `publish_feedback(...)` function, remove the placeholder `pass` statement.

3. Invoke the `pubsub.publish_feedback(feedback)` function.

4. Then, return a response to the client indicating that feedback was received.

quiz/api/api.py, publish_feedback(...) function

```
"""
Publish feedback
- Call pubsub helper
- Compose and return response
"""
```

```
def publish_feedback(feedback):
    # TODO: Publish the feedback using your pubsub module,
    # return the result

    result = pubsub.publish_feedback(feedback)
    response = Response(json.dumps(
                        result, indent=2, sort_keys=True))
    response.headers['Content-Type'] = 'application/json'
    return response

    # END TODO
```

5. Save the file.

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. Preview the web application.

3. Click **Take Test**.

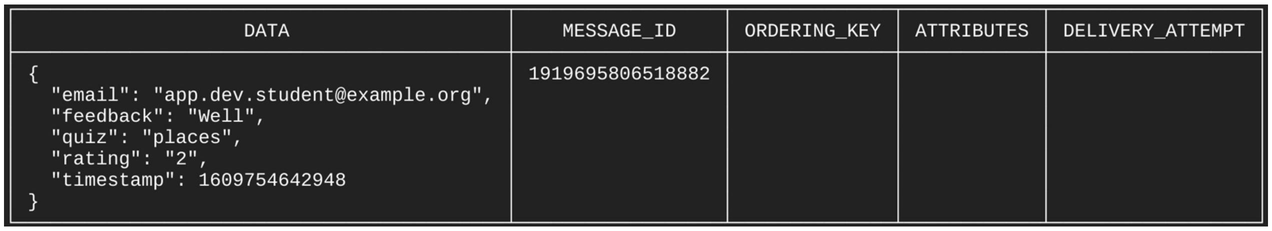
4. Click **Places**.

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

6. In the second Cloud Shell window, to pull a message from the `worker-subscription`, execute the following command:

```
gcloud pubsub subscriptions pull worker-subscription --auto-ack
```

Output:



```
┌──────────────────────────────────────────────┬──────────────────┬──────────────┬────────────┬──────────────────┐
│                     DATA                     │    MESSAGE_ID    │ ORDERING_KEY │ ATTRIBUTES │ DELIVERY_ATTEMPT │
├──────────────────────────────────────────────┼──────────────────┼──────────────┼────────────┼──────────────────┤
│ {                                            │ 1919695806518882 │              │            │                  │
│   "email": "app.dev.student@example.org",    │                  │              │            │                  │
│   "feedback": "Well",                        │                  │              │            │                  │
│   "quiz": "places",                          │                  │              │            │                  │
│   "rating": "2",                             │                  │              │            │                  │
│   "timestamp": 1609754642948                 │                  │              │            │                  │
│ }                                            │                  │              │            │                  │
└──────────────────────────────────────────────┴──────────────────┴──────────────┴────────────┴──────────────────┘
```

7. Stop the web and console applications.

Subscribe to Cloud Pub/Sub Topics Programmatically

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

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

The code you add performs these actions:

1. Return to the `...quiz/gcp/pubsub.py` file.

2. Declare a Cloud Pub/Sub Subscriber Client.

3. Get the fully qualified path referencing the `'worker-subscription'`.

4. Move to the `pull_feedback(callback)` function, and remove the placeholder `pass` statement.

5. Use the subscriber client to subscribe to the worker subscription, invoking the callback when a message is received.

   /quiz/gcp/pubsub.py

```
# TODO: Create a Pub/Sub Subscriber Client

sub_client = pubsub_v1.SubscriberClient()

# END TODO

# TODO: Create a Subscription object named
# worker-subscription

sub_path = sub_client.subscription_path(project_id, 'worker-subscription')

# END TODO

def pull_feedback(callback):
    # TODO: Subscribe to the worker-subscription,
    # invoking the callback

    sub_client.subscribe(sub_path, callback=callback)

    # END TODO
```

6. Save the file.

Write code to use the Pub/Sub subscribe functionality

The code you add performs these actions:

1. In the `...quiz/console/worker.py` file, load the `pubsub` module from the `quiz.gcp` package.

2. In the `pubsub_callback(message)` function, acknowledge the message

3. Log the message to the console.

4. In the `main()` function, register the handler function as the Pub/Sub subscription callback.

console/worker.py

```
# TODO: Load the pubsub, languageapi and spanner modules from
# from the quiz.gcp package

from quiz.gcp import pubsub

# END TODO

def pubsub_callback(message):
    # TODO: Acknowledge the message

    message.ack()

    # END TODO

    log.info('Message received')

    # TODO: Log the message

    log.info(message)

    # END TODO

def main():
    log.info('Worker starting...')

    # TODO: Register the callback

    pubsub.pull_feedback(pubsub_callback)

    # END TODO

    while True:
        time.sleep(60)
```

5. Save the file.

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

1. In the first Cloud Shell window, start the web application if it's not already running.

```
2. python run_server.py
```

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

```
4. . run_worker.sh
```

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.

   The worker application should show that it has received the feedback message via its handler and displayed details it in the window. The message itself is truncated.

```
INFO:root:Worker starting...
INFO:root:Message received
INFO:root:Message {
 data: b'{\n  "email": "app.dev.student@example.org",\n  "fee...'
 ordering_key: ''
 attributes: {}
}
```

10. 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

The code you add performs these actions:

1. In the editor, move to the top of the `...quiz/gcp/languageapi.py` file.

2. Load the `language` module from the `google.cloud` package.

3. Load the `enums` and `types` modules from the `google.cloud.language` package.

4. Create a [Cloud Natural Language client object](#).
5. Move to the `analyze(...)` function, and create a Document object to pass to the Natural Language client.

6. Configure this object with two properties: `content` and `type`.

7. Assign the feedback text to this object's `content` property.

8. Set the `type` property value to the type that corresponds to `PLAIN_TEXT`.

9. Use the Natural Language client object to analyze the sentiment of the document.

10. Then, return the sentiment score from the Natural Language API.

quiz/gcp/languageapi.py

```
# TODO: Import the language module

from google.cloud import language_v1

# END TODO

# TODO: Create the Language API client

lang_client = language_v1.LanguageServiceClient()

# END TODO

def analyze(text):

    # TODO: Create a Document object
    doc = language_v1.types.Document(content=text,
                    type_='PLAIN_TEXT')

    # END TODO

    # TODO: Analyze the sentiment

    sentiment = lang_client.analyze_sentiment(
                    document=doc).document_sentiment

    # END TODO

    # TODO: Return the sentiment score

    return sentiment.score

    # END TODO
```

11. Save the file.

Write code to use the Natural Language API functionality

1. In the `.../quiz/console/worker.py` file, load the `languageapi` module.
2. In the `pubsub_callback(message)` function, and after the existing code, perform sentiment detection on the feedback.
3. Then, log the score to the console.
4. Assign a new score property to the feedback object.
5. Return the message data.

console/worker.py

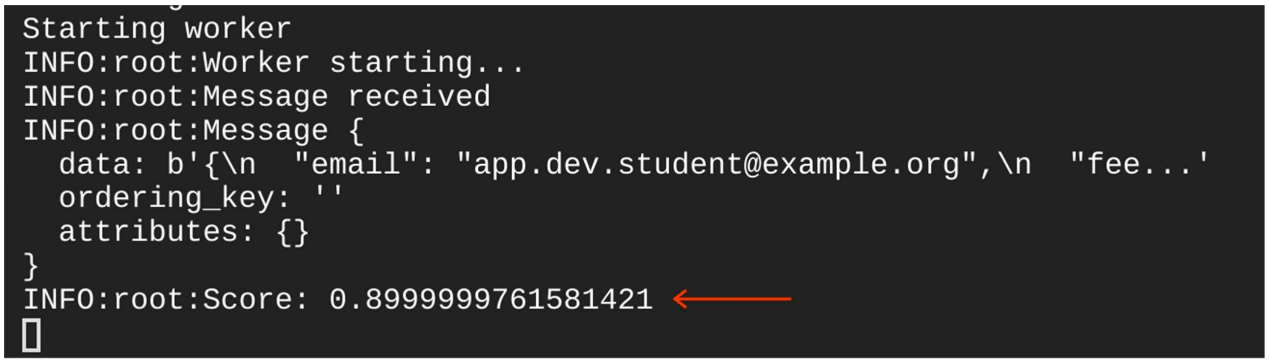
```
# TODO: Load the pubsub, languageapi and spanner modules from
# from the quiz.gcp package

from quiz.gcp import pubsub, languageapi

# END TODO

def pubsub_callback(message):

    # TODO: Acknowledge the message

    message.ack()

    # END TODO

    log.info('Message received')

    # TODO: Log the message

    log.info(message)
    # END TODO

    data = json.loads(message.data)

    # TODO: Use the languageapi module to
    # analyze the sentiment

    score = languageapi.analyze(str(data['feedback']))

    # END TODO

    # TODO: Log the sentiment score

    log.info('Score: {}'.format(score))

    # END TODO

    # TODO: Assign the sentiment score to
    # a new score property

    data['score'] = score

    # END TODO
```

6. Save the file.

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

1. Return to the first Cloud Shell window.
2. Run the web application.
3. Switch to the second Cloud Shell window.
4. Restart the worker application.
5. Preview the web application.
6. Click **Take Test**.
7. Click **Places**.
8. Answer the questions, 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 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: b'{\n  "email": "app.dev.student@example.org",\n  "fee...'
  ordering_key: ''
  attributes: {}
}
INFO:root:Score: 0.8999999761581421
```

10. Stop the web and console applications.

Persist Data to Cloud Spanner

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

Create a Cloud Spanner instance

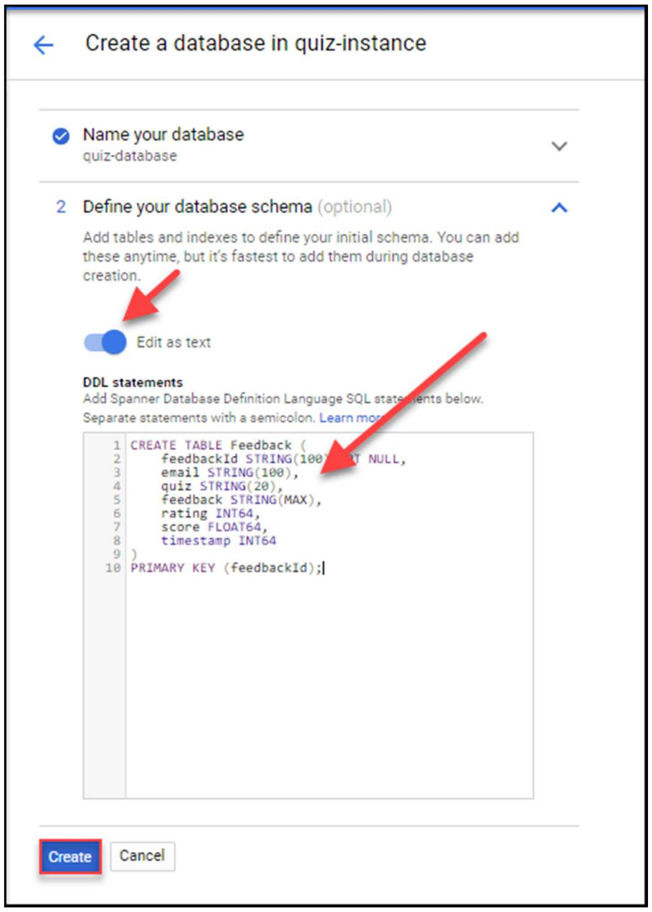
1. Return to the Cloud Console and click **Navigation menu** > **Spanner** > **Create instance**.
2. For **Instance name**, type **quiz-instance**
3. In the **Configuration** section, select **us-central1** as the region.
4. Click **Create**.

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**.
4. For **DDL statements**, type the following SQL statement:

```
CREATE TABLE Feedback (
    feedbackId STRING(100) NOT NULL,
    email STRING(100),
    quiz STRING(20),
    feedback STRING(MAX),
    rating INT64,
    score FLOAT64,
    timestamp INT64
)
PRIMARY KEY (feedbackId);
```

5. Click **Create**.


Create a database in quiz-instance
Name your database
quiz-database
2 Define your database schema (optional)
Edit as text
Create
Cancel

Add tables and indexes to define your initial schema. You can add these anytime, but it's fastest to add them during database creation.

DDL statements

Add Spanner Database Definition Language SQL state[illegible]ents below.
Separate statements with a semicolon. Learn mor[illegible]

```
CREATE TABLE Feedback (
    feedbackId STRING(100) [illegible]T NULL,
    email STRING(100),
    quiz STRING(20),
    feedback STRING(MAX),
    rating INT64,
    score FLOAT64,
    timestamp INT64
)
PRIMARY KEY (feedbackId);
```

Write code to persist data into Cloud Spanner

The code you add performs these actions:

1. Return to the code editor, and move to the top of the `.../quiz/gcp/spanner.py` file.

2. Load the `spanner` module from the `google.cloud` package.

3. Construct a Cloud Spanner client.
4. Get a reference to the Spanner instance.
5. Get a reference to the Spanner database.

quiz/gcp/spanner.py

```
import re

# TODO: Import the spanner module

from google.cloud import spanner

# END TODO

# TODO: Create a spanner Client
```

```
spanner_client = spanner.Client()

# END TODO

# TODO: Get a reference to the Cloud Spanner quiz-instance

instance = spanner_client.instance('quiz-instance')

# END TODO

# TODO: Get a reference to the Cloud Spanner quiz-database

database = instance.database('quiz-database')

# END TODO
```

6. Move to the `saveFeedback(...)` function.

7. Create a database.batch object using a `with` block. This can be used to perform multiple operations against a Spanner database.

8. Create a key for the feedback record from the email, quiz, and timestamp properties from the data. For the email property, use the `reverse_email(...)` function to take the input email and create a reversed string. For example: Input: app.dev.student@example.com Output: com_example_student_dev_app

9. Use the batch object to insert a record, using a set of columns and values.

quiz/gcp/spanner.py

```
def save_feedback(data):
    # TODO: Create a batch object for database operations

    with database.batch() as batch:

    # END TODO

        # TODO: Create a key for the record
        # from the email, quiz and timestamp

        feedback_id = '{}_{}_{}'.format(
                    reverse_email(data['email']),
                    data['quiz'],
                    data['timestamp'])

        # END TODO

        # TODO: Use the batch to insert a record
        # into the feedback table
        # This needs the columns and values

        batch.insert(
            table='feedback',
            columns=(
                'feedbackId',
                'email',
                'quiz',
                'timestamp',
                'rating',
                'score',
                'feedback'
            ),
            values=[
                (
                    feedback_id,
                    data['email'],
                    data['quiz'],
                    data['timestamp'],
                    data['rating'],
                    data['score'],
                    data['feedback']
                )
            ]
        )

        # END TODO
```

10. Save the file.

Write code to use the Cloud Spanner functionality

The code you add performs these actions:

1. In the `.../quiz/console/worker.py` file, load the `spanner` module.
2. After the existing code in the `pubsub_callback(message)` function, save the feedback into Spanner.
3. Log a message to the console to say that the feedback has been saved.

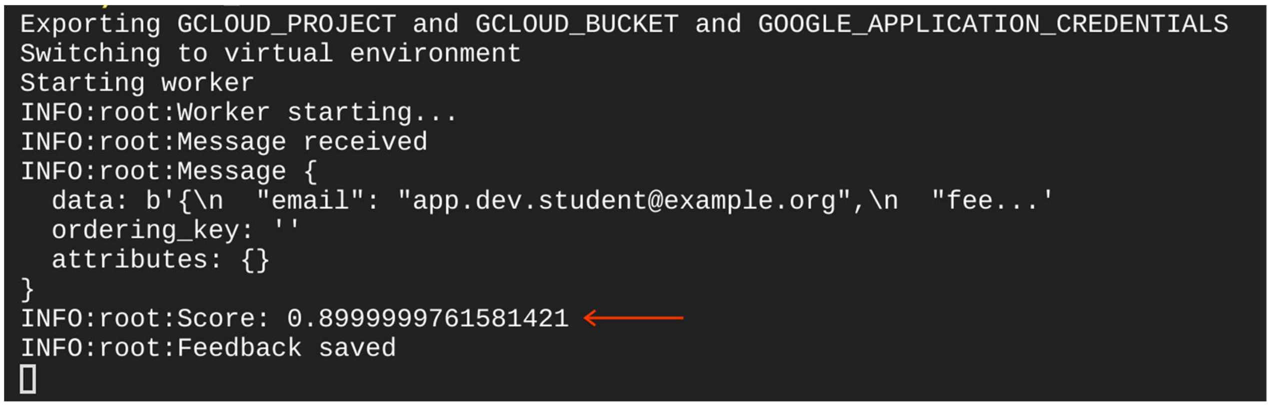
quiz/console/worker.py

```
# TODO: Load the pubsub, languageapi and spanner modules
# from the quiz.gcp package

from quiz.gcp import pubsub, languageapi, spanner

# END TODO

logging.basicConfig(stream=sys.stdout, level=logging.INFO)
log = logging.getLogger()

def pubsub_callback(message):

    # TODO: Acknowledge the message

    message.ack()

    # END TODO

    log.info('Message received')

    # TODO: Log the message

    log.info(message)

    # END TODO

    data = json.loads(message.data)
    # TODO: Use the languageapi module to
    # analyze the sentiment

    score = languageapi.analyze(str(data['feedback']))

    # END TODO

    # TODO: Log the sentiment score

    log.info('Score: {}'.format(score))

    # END TODO

    # TODO: Assign the sentiment score to
    # a new score property

    data['score'] = score

    # END TODO

    # TODO: Use the spanner module to save the feedback

    spanner.save_feedback(data)

    # END TODO

    # TODO: Log a message to say the feedback
    # has been saved

    log.info('Feedback saved')

    # END TODO
```

Run the web and worker application and test Cloud Spanner

1. Save all the files, and then return to the first Cloud Shell window.
2. Start the web application and then the worker application.
3. Preview the web application.
4. Click **Take Test** > **Places**.
5. Answer the questions, select the rating, enter some feedback text, and then click **Send Feedback**.
6. 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.



```
Exporting GCLOUD_PROJECT and GCLOUD_BUCKET and GOOGLE_APPLICATION_CREDENTIALS
Switching to virtual environment
Starting worker
INFO:root:Worker starting...
INFO:root:Message received
INFO:root:Message {
  data: b'{\n  "email": "app.dev.student@example.org",\n  "fee...'
  ordering_key: ''
  attributes: {}
}
INFO:root:Score: 0.8999999761581421
INFO:root:Feedback saved
```

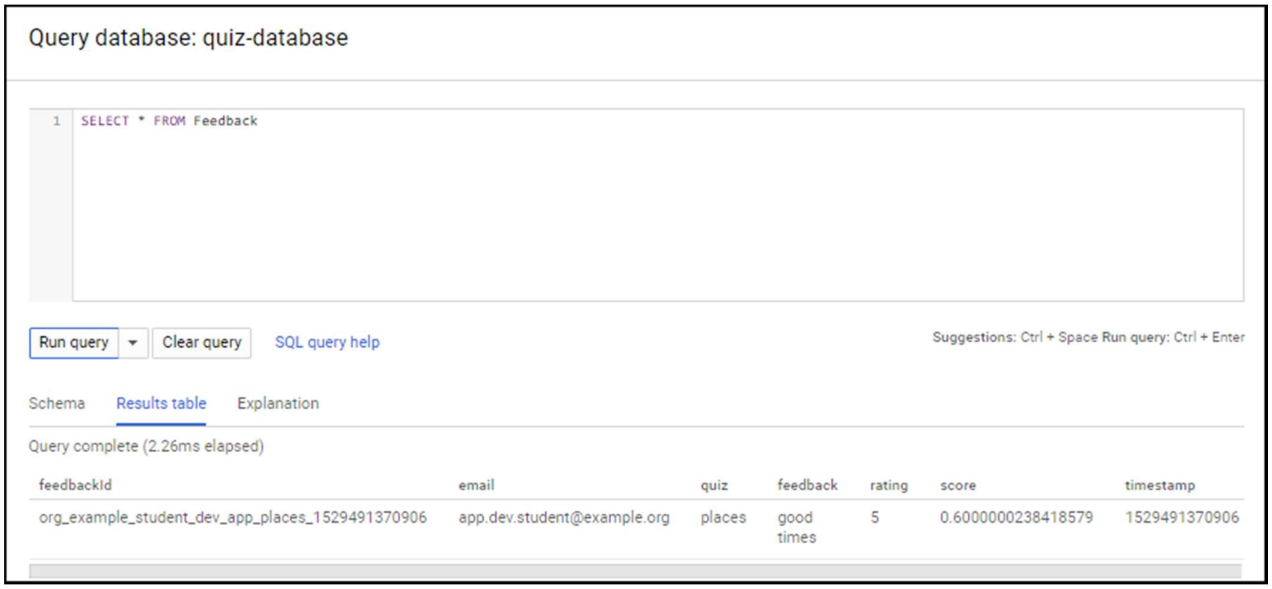
7. Return to the **Cloud Platform Console** and click **Navigation menu** > **Spanner**.

8. Select **quiz-instance > quiz-database > Query**.

9. To execute a query, for **Query**, type `SELECT * FROM Feedback`, and then click **Run query**.

```
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 score from the Cloud Natural Language API.



Query database: quiz-database

```
SELECT * FROM Feedback
```

Query complete (2.26ms elapsed)

feedbackId	email	quiz	feedback	rating	score	timestamp
org_example_student_dev_app_places_1529491370906	app.dev.student@example.org	places	good times	5	0.6000000238418579	1529491370906

Congratulations!

This concludes the self-paced lab, App Dev: Developing a Backend Service - Python. You managed resources from the command line, debuged source code in production and ran API backends.

Finish your Quest

This self-paced lab is part of the Application Development - Python 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 January 4, 2021
Lab last tested January 4, 2021