

JAVAMS08 Using Cloud Platform APIs

Overview

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

In addition to the integration you have seen in previous labs using Spring Boot starters, Google Cloud offers many other APIs that you can use directly from your Java applications.

Google Cloud has a set of ready-to-use, idiomatic Java client libraries called `google-cloud-java`. You can consume any of the client libraries for `google-cloud-java` even without a Spring Boot starter.

In this lab, you modify the application to use Cloud Vision API to analyze the images uploaded by the users.

Because the Spring Cloud GCP project does not have a Spring Boot starter for Vision API, in this lab you integrate client libraries without using a Spring Cloud GCP Spring Boot starter. You also configure and use a service account to provide your application with the correct permissions to access Vision API.

Vision API enables developers to understand the content of an image by encapsulating powerful machine-learning models in an easy-to-use REST API. Vision API quickly classifies images into thousands of categories, detects individual objects and faces within images, and reads printed words contained in images. You can build metadata on your image catalog, moderate offensive content, or enable new marketing scenarios

through image sentiment analysis. Vision API enables your applications to easily detect broad sets of objects in your images, from flowers, animals, or transportation to thousands of other object categories commonly found in images.

Objectives

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

- Add a Google Cloud API Java library to an application
- Create a Google Cloud credential scope for Spring
- Create a Java bean that implements Vision API features
- Use Vision API to add image analysis to an application


Setup and requirements


How to start your lab and sign in to the Console


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

[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 

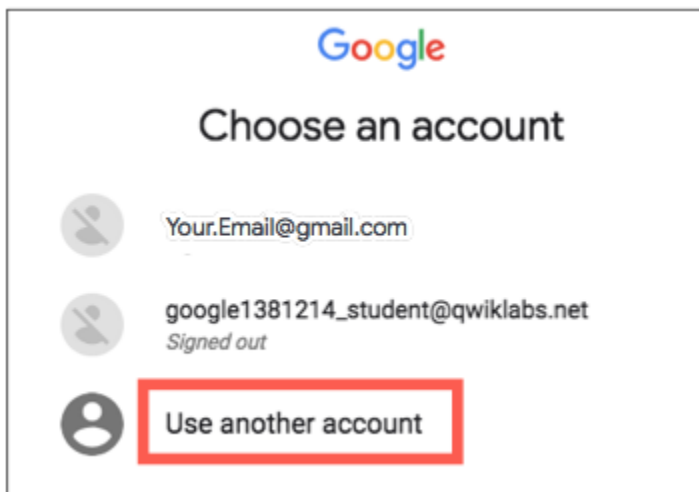
GCP Project ID
qwiklabs-gcp-4fbfecac8667e457 

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

- Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

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

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



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

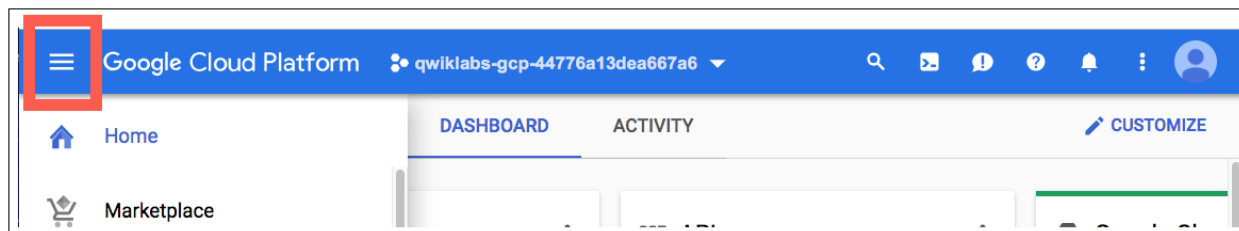
Note: You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:

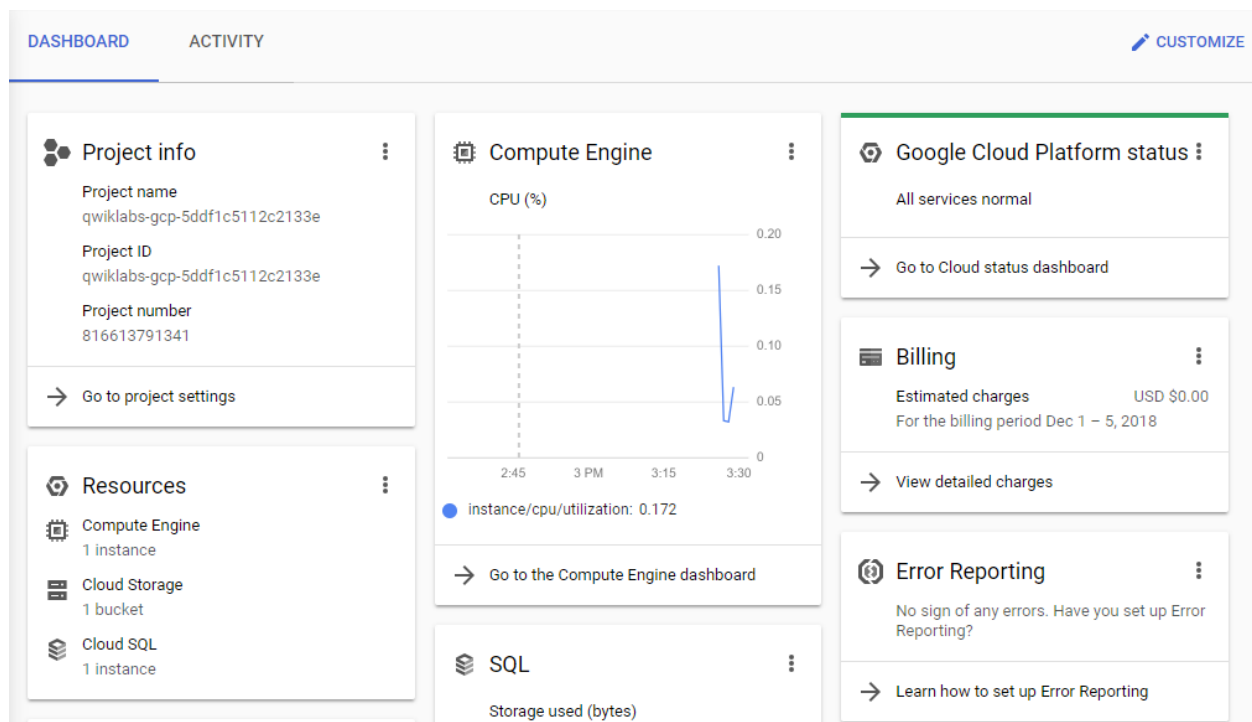
- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

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

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



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



Task 1. Fetch the application source files

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

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

Note: A Cloud Storage bucket that is named using the project ID for this lab is automatically created for you by the lab setup. The source code for your applications is copied into this bucket when the Cloud SQL server is ready. You might have to wait a few minutes for this action to complete.

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

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

Copied!

content_copy

4. Verify that the demo application files were created:

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

Copied!

content_copy

5. Copy the application folders to Cloud Shell:

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

Copied!

content_copy

6. Make the Maven wrapper scripts executable:

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

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

Copied!

content_copy

Now you're ready to go!

Task 2. Enable Vision API

In this task, you enable the Vision API so that you can use it to analyze uploaded images.

- Enter the following command in the Cloud Shell code editor to enable Vision API:

```
gcloud services enable vision.googleapis.com
```

Copied!

content_copy

Task 3. Add the Vision client library

In this task, you add the Vision client library to the guestbook frontend application.

1. In the Cloud Shell code editor, open `~/guestbook-frontend/pom.xml`.
2. Insert the following new dependency at the end of the `<dependencies>` section, just before the closing `</dependencies>` tag:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-vision</artifactId>
</dependency>
```

Copied!

content_copy

Task 4. Add a Google Cloud credential scope for Spring

In this task, you specify the Google Cloud scope in the `application.properties` file.

Without customization, the Spring Cloud GCP starters request permission scopes to use APIs that the starters integrate with. Because you use a new API that is not integrated with the starter, you must specify the scope. An all-purpose scope can be used to request permission for all basic Google Cloud APIs.

1. In the Cloud Shell code editor, open `~/guestbook-frontend/src/main/resources/application.properties`.
2. Add the following entry:

```
spring.cloud.gcp.credentials.scopes=https://www.googleapis.com/auth/cloud-platform
```

Copied!

content_copy

The `application.properties` file should contain the properties shown in the following screenshot:

```
1. server.port=${PORT:8080}
2. spring.cloud.gcp.trace.enabled=false
3. spring.cloud.gcp.credentials.scopes=https://www.googleapis.com/auth/cloud-platform
```

Note: The Google Cloud scope indicates that the application wants to use all of the Google Cloud APIs. However, the application can use the API only if the API is enabled, and if the application has permission to use it (through the roles bound to the service account, or machine credentials, used to run the application).

In a production application, you should always specify the narrowest scopes that the application needs to use the APIs.

Task 5. Analyze the image

In this task, you analyze the uploaded image, label the objects in the image, and print out the response.

Given an image, Vision API can identify objects, landmarks, the location of faces, and facial expressions. It can also extract text and evaluate whether the image is considered safe.

Add a method to the frontend application to use Vision API to analyze an image

You add a method to `FrontendController.java` that sends an image to Google Vision API for analysis.

1. In the Cloud Shell code editor, open `~/guestbook-frontend/src/main/java/com/example/frontend/FrontendController.java`.
2. Insert the following `import` directives immediately below the existing imports:

```
// Add Vision API imports
import org.springframework.cloud.gcp.vision.CloudVisionTemplate;
import com.google.cloud.vision.v1.Feature.Type;
import com.google.cloud.vision.v1.AnnotateImageResponse;
```

Copied!

content_copy

3. Insert the following code into the `FrontendController` class definition immediately above the `@GetMapping("/")` line:

```
@Autowired
private CloudVisionTemplate visionTemplate;
```

Copied!

content_copy

Modify the frontend application to analyze the image once it is written to the Cloud Storage bucket

In `FrontendController.java` you add a call to the new `analyzeImage` method after the code that uploads the file to Cloud Storage.

1. In the Cloud Shell code editor, open `~/guestbook-frontend/src/main/java/com/example/frontend/FrontendController.java`.
2. Insert the following line into the `post` method definition after the `try` block inside the `post` method definition:

```
// After writing to GCS, analyze the image.  
AnnotateImageResponse response = visionTemplate  
.analyzeImage(resource, Type.LABEL_DETECTION);  
log.info(response.toString());
```

Copied!

content_copy

The `post` method definition should look like the screenshot:

```
68 @PostMapping("/post")  
69 public String post(@RequestParam(name="file", required=false) MultipartFile file, @RequestParam String name, @RequestParam String message, Model model)  
70 throws IOException {  
71     model.addAttribute("name", name);  
72  
73     String filename = null;  
74     if (file != null && !file.isEmpty())  
75         && file.getContentType().equals("image/jpeg")) {  
76  
77         // Bucket ID is our Project ID  
78         String bucket = "gs://" + projectIdProvider.getProjectId();  
79  
80         // Generate a random file name  
81         filename = UUID.randomUUID().toString() + ".jpg";  
82         WritableResource resource = (WritableResource)  
83             context.getResource(bucket + "/" + filename);  
84  
85         // Write the file to Cloud Storage using WritableResource  
86         try (OutputStream os = resource.getOutputStream()) {  
87             os.write(file.getBytes());  
88         }  
89  
90         // After writing to GCS, analyze the image.  
91         AnnotateImageResponse response = visionTemplate  
92             .analyzeImage(resource, Type.LABEL_DETECTION);  
93         log.info(response.toString());  
94     }  
95 }
```

Task 6. Set up a service account

In this task, you create a service account with the Owner role, and you create a JSON file containing the authentication keys for the service account.

To make calls to the Vision API from your application, you need a service account with sufficient permissions.

1. Create a service account specific to the guestbook application:

```
export PROJECT_ID=$(gcloud config list --format 'value(core.project)')
gcloud iam service-accounts create guestbook
```

Copied!

content_copy

2. Add the Owner role to this service account:

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

Copied!

content_copy

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

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

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

Copied!

content_copy

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

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

Task 7. Test the application in Cloud Shell

In this task, you run the application components in Cloud Shell to test the new Vision API functionality. When starting the frontend application you use the new service account user credential so that the frontend application can authenticate with Vision API.

1. In Cloud Shell, change to the `guestbook-service` directory:

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

Copied!

content_copy

2. Run the backend service application:

```
./mvnw spring-boot:run -Dspring-boot.run.jvmArguments="-Dspring.profiles.active=cloud"
```

Copied!

content_copy

The backend service application launches on port 8081. This takes a minute or two to complete and you should wait until you see that the `GuestbookApplication` is running.

```
Started GuestbookApplication in 20.399 seconds (JVM running...)
```

3. Open a new Cloud Shell session tab to run the frontend application by clicking the plus (+) icon to the right of the title tab for the initial Cloud Shell session.

4. Change to the `guestbook-frontend` directory:

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

Copied!

content_copy

5. Start the guestbook frontend application using the cloud profile and the guestbook service account credentials:

```
./mvnw spring-boot:run \
-Dspring-boot.run.jvmArguments="-Dspring.profiles.active=cloud \
-Dspring.cloud.gcp.credentials.location=file:/// $HOME/service-account.json"
```

Copied!

content_copy

6. Open the Cloud Shell web preview on port 8080 and post a message with a small JPEG image.
7. In the frontend application Cloud shell tab you should see image labels in the log output similar to the following example:

```
label_annotations {
  mid: "/m/09ggk"
  description: "purple"
  score: 0.8982213
```

```
    topicality: 0.8982213
  }
  label_annotations {
    mid: "/m/07vwy6"
    description: "street art"
    score: 0.86210686
    topicality: 0.86210686
  }
  label_annotations {
    mid: "/m/04rd7"
    description: "mural"
    score: 0.81835103
    topicality: 0.81835103
  }
}
```

Task 8. Review

In this lab you added a Google Cloud API Java library to an application. You created a Google Cloud credential scope for Spring. You also created a Java bean that implements Vision API features. Finally, you used the Vision API to add image analysis to an application.