JAVAMS11 Working with Spanner

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 a previous lab, you modified the application to use Cloud SQL for database services. Cloud SQL provides a managed database service for applications that require robust relational database services. But when higher performance and transactions are critical to your application, you can use Cloud Spanner to provide high-performance, relational database services. Spanner is an enterprise-grade, globally distributed, strongly consistent database service built for the cloud specifically to combine the benefits of relational database structure with non-relational horizontal scale. This combination delivers high-performance transactions and strong consistency across rows, regions, and continents with enterprise-grade security.

In this lab, you update your application to integrate your Cloud Spanner instance with Spring Data. You will then test the changes locally in Cloud Shell, and then redeploy the backend service to App Engine.

Objectives

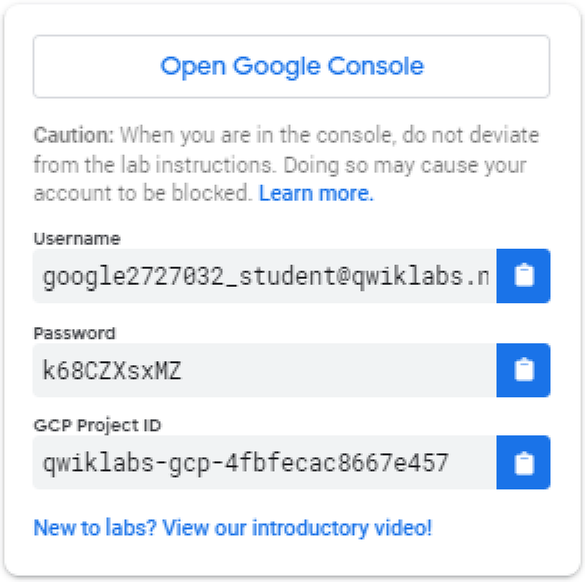
In this lab, you learn how to perform the following tasks:

- Create a Spanner instance and database
- Use the data definition language (DDL) to create a Spanner table
- Use Spring to add support for Spanner to an application
- Modify a Java application to use Spanner instead of Cloud SQL

Setup and requirements

How to start your lab and sign in to the Console

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



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

Username
google2727032_student@qwiklabs.n

Password
k68CZXsxMZ

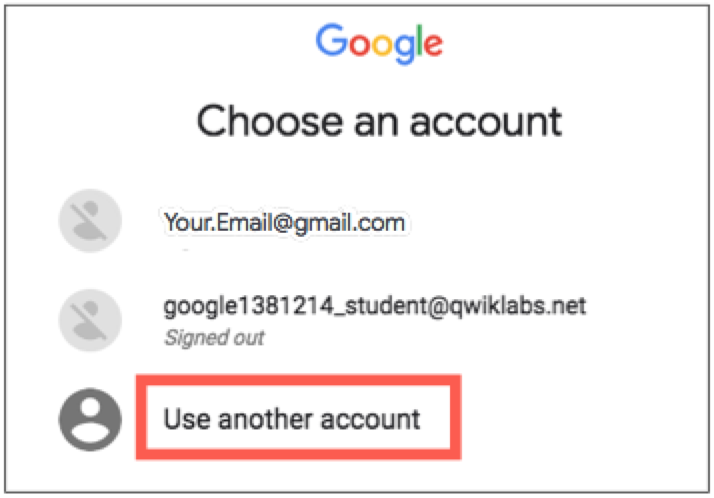
GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

New to labs? View our introductory video!

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

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

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



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

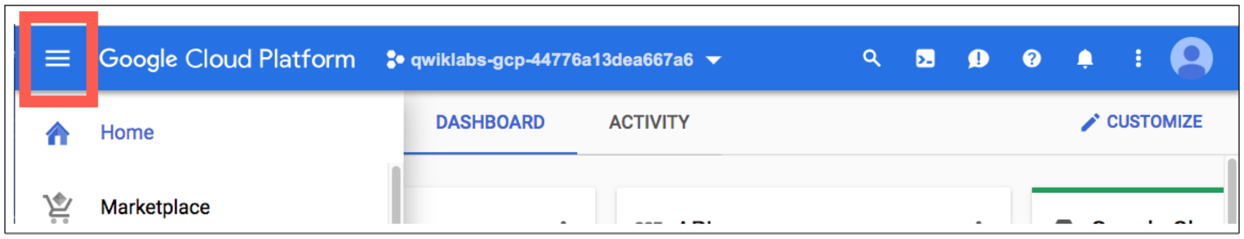
Note: You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:

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

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

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



After you complete the initial sign-in steps, the project dashboard 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. Run the below command to create an App Engine app.

```
gcloud app create --region=us-central
```

Copied!
content_copy

3. To activate the code editor, click the `Open Editor` button on the toolbar of the Cloud Shell window.

You can switch between Cloud Shell and the code editor by using `Open Editor` and `Open Terminal` icon as required.

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 once the Cloud SQL server is ready and both application microservices components have been deployed to App Engine. You might have to wait up to 10 minutes for the deployment tasks to complete.

4. 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

5. Verify that the demo application files were created:

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

Copied!
content_copy

6. Copy the application folders to Cloud Shell:

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

Copied!
content_copy

7. Make the Maven wrapper scripts executable:

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

Copied!

content_copy

8. Check that the frontend application is running. You can find the URL of the frontend application that should now be running on App Engine via the following command:

```
gcloud app browse
```

Copied!
content_copy

9. Click the link to open a browser tab to the frontend URL. You will come back to this later.

Task 2. Enable Spanner API

In this task, you enable Spanner API so that you can create a Spanner database for your application.

- Switch back to Cloud Shell and enable the Spanner API:

```
gcloud services enable spanner.googleapis.com
```

Copied!
content_copy

Task 3. Create a new Spanner instance

In this task, you create a Spanner instance, a database and a database table.

Create a Spanner instance

You create a Spanner instance, and then create a database on that instance for the demo application.

1. Create a Spanner instance:

```
gcloud spanner instances create guestbook --config=regional-us-central1 \
  --nodes=1 --description="Guestbook messages"
```

Copied!
content_copy

2. Create a `messages` database in the Spanner instance:

```
gcloud spanner databases create messages --instance=guestbook
```

Copied!
content_copy

3. Confirm that the database exists by listing the databases of the Spanner instance:

```
gcloud spanner databases list --instance=guestbook
```

Copied!
content_copy
The output indicates that the database is ready:

```
NAME      STATE
messages  READY
```

Create a table in the Spanner database

You create a table in the `messages` database by creating a file that contains a DDL statement and then running the command.

1. In the `guestbook-service` folder, create the `db` folder:

```
cd ~/guestbook-service
mkdir db
```

Copied!
content_copy

2. In the Cloud Shell code editor, create a file named `spanner.ddl` in the `~/guestbook-service/db/` directory.

3. Add the following command to the `spanner.ddl` file:

```
CREATE TABLE guestbook_message (
    id STRING(36) NOT NULL,
    name STRING(255) NOT NULL,
    image_uri STRING(255),
    message STRING(255)
) PRIMARY KEY (id)
```

Copied!
content_copy

4. In Cloud Shell, use `gcloud` to run the DDL command to create the table:

```
gcloud spanner databases ddl update messages \
  --instance=guestbook --ddl-file=$HOME/guestbook-service/db/spanner.ddl
```

Copied!
content_copy

5. In the Google Cloud console, use the navigation menu to navigate to **Spanner** in the **Databases** section.
6. Click the name of the **Guestbook messages** instance to open it.
7. Click the name of the **messages** database to open it.

You should see the **guestbook_message** table if the `spanner.ddl` file was processed successfully.

8. Click the **guestbook_message** table to open it.

The database opens showing the schema details tab. You can click on **Show Equivalent DDL** button, the schema should match the schema you created in the `spanner.ddl` file. Click **Close**.

9. Click on the **Data** tab. There is no data in the table yet.

Task 4. Add Spring Cloud GCP Spanner Starter

In this task, you update the Guestbook Service's `pom.xml` file to remove the Cloud SQL starter, and add the Spanner Starter dependency.

1. Switch back to the tab running the Cloud Shell code editor.
2. In the editor, open `~/guestbook-service/pom.xml`.
3. Delete the `Spring Data JPA` by removing these lines:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

4. Delete the `Cloud SQL Starter` by removing these lines:

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

5. Delete `HSQL` by removing these lines:

```
<dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
        <scope>runtime</scope>
</dependency>
```

6. Now, add the following code at the end of the `<dependencies>` section, immediately before the closing `</dependencies>` tag:

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

Copied!
content_copy

Task 5. Update configuration

In this task, you add the Spanner instance and database configuration properties to `application.properties` and `application-cloud.properties` for the guestbook backend service application. You will also delete the Cloud SQL configuration properties.

There is no Spanner emulator, meaning both Dev and Prod will always need a real Spanner instance running. For the purpose of this lab, we'll use the same Spanner instance.

1. In the editor, open `guestbook-service/src/main/resources/application.properties`. Delete the two Cloud SQL Configuration lines and add the following Spanner configuration:

```
# Add Spanner configuration
spring.cloud.gcp.spanner.instance-id=guestbook
spring.cloud.gcp.spanner.database=messages
```

Copied!
content_copy

2. Next, open `guestbook-service/src/main/resources/application-cloud.properties` and remove the following Spring properties for Cloud SQL:

```
spring.cloud.gcp.sql.enabled=true
spring.cloud.gcp.sql.database-name=messages
spring.cloud.gcp.sql.instance-connection-name=...
```

Task 6. Update the backend service to use Spanner

In this task, you modify `GuestbookMessage.java` to use the Spanner annotations.

You can use the `@Table` annotation to map a Java class to a Spanner table. And you can use the `@Column` annotation to map properties to table columns. You use the `@Table` annotation to map to the `guestbook_message` table that was created when you ran the DDL statement with `gcloud`.

The `id` property is specified as the primary key. In the class constructor, the `id` property is auto-populated with a random UUID. The UUIDv4 format is recommended over a monotonically increasing ID. This format helps Spanner avoid creating hotspots when it automatically shards the data.

The other class properties included match the table's schema in the DDL statement, except for `imageUri`, which uses the `@Column` annotation to map the table column name `image_uri` to the property name `imageUri`.

1. In the Cloud Shell code editor, open `guestbook-service/src/main/java/com/example/guestbook/GuestbookMessage.java`.

2. Replace the entire contents of this file with the following code:

```
package com.example.guestbook;
import lombok.*;
import org.springframework.cloud.gcp.data.spanner.core.mapping.*;
import org.springframework.data.annotation.Id;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
@Data
@Table(name = "guestbook_message")
@JsonIgnoreProperties(value={"id"}, allowSetters = false)
public class GuestbookMessage {
        @PrimaryKey
        @Id
        private String id;
        private String name;
        private String message;
        @Column(name = "image_uri")
        private String imageUri;
        public GuestbookMessage() {
                this.id = java.util.UUID.randomUUID().toString();
        }
}
```
Copied!
content_copy

Task 7. Add a method to find messages by name

In this task, you update the `GuestbookMessageRepository.java` file to use `String` as the ID type.

Spring Data Spanner implements many commonly used Spring Data patterns, such as creating simple methods that can be automatically translated to corresponding SQL queries.

One example is a simple method signature: `List<GuestbookMessage> findByName(String name);`. The Spring framework enables querying the Spanner table with the SQL query `SELECT * FROM guestbook_message WHERE name = ?`.

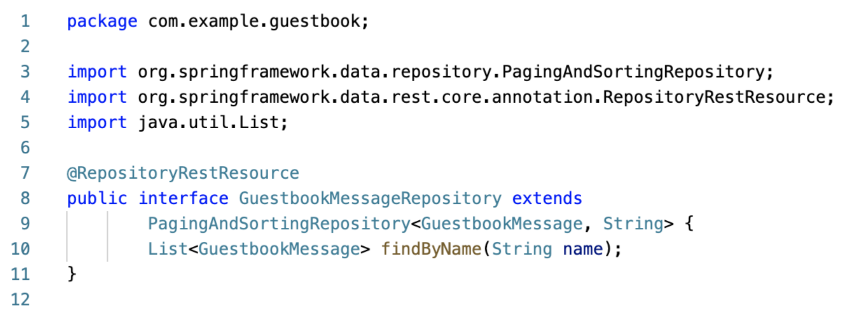
1. In the Cloud Shell code editor, open `guestbook-service/src/main/java/com/example/guestbook/GuestbookMessageRepository.java`.

2. Insert the following `import` statement below the existing `import` directives:

```
import java.util.List;
```
Copied!
content_copy
3. Change the datatype for the `PagingAndSortingRepository` `GuestbookMessage` parameter from `Long` to `String`:

```
public interface GuestbookMessageRepository extends
        PagingAndSortingRepository<GuestbookMessage, String> {
}
```
Copied!
content_copy
4. Insert the following code into the definition for the `GuestbookMessageRepository` public interface, immediately before the closing brace:

```
List<GuestbookMessage> findByName(String name);
```
Copied!
content_copy
The `GuestbookMessageRepository.java` file should now look like the screenshot:


```
package com.example.guestbook;

import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import java.util.List;

@RepositoryRestResource
public interface GuestbookMessageRepository extends
        PagingAndSortingRepository<GuestbookMessage, String> {
        List<GuestbookMessage> findByName(String name);
}

```

Task 8. Test the backend service application locally in Cloud Shell

In this task, you run the updated guestbook backend service application in Cloud Shell in order to test that the application has been correctly configured to use Spanner for database services.

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

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

Copied!
content_copy

2. Launch the guestbook backend service application locally:

```
./mvnw spring-boot:run
```

Copied!
content_copy

3. In a new Cloud Shell tab, use `curl` to post a message:

```
curl -XPOST -H "content-type: application/json" \
  -d '{"name": "Ray", "message": "Hello Cloud Spanner"}' \
  http://localhost:8081/guestbookMessages
```

Copied!
content_copy

4. List all the messages:

```
curl http://localhost:8081/guestbookMessages
```

Copied!
content_copy

5. List specific messages using the custom `findByName()` search you added above:

```
curl http://localhost:8081/guestbookMessages/search/findByName?name=Ray
```

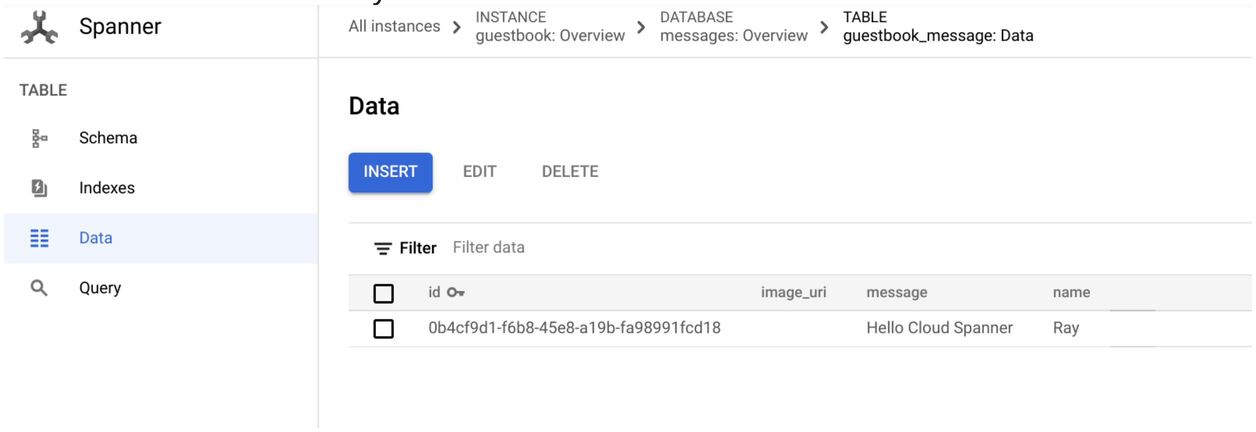
Copied!
content_copy

6. Use the `gcloud` spanner command with a SQL query to validate that messages exist:

```
gcloud spanner databases execute-sql messages --instance=guestbook \
    --sql="SELECT * FROM guestbook_message WHERE name = 'Ray'"
```

Copied!
content_copy

7. Switch back to the Google Cloud console and navigate to your Spanner `guestbook_message` table to see the new entry. Click on the **Data** tab to see the new entry.


Spanner
All instances
INSTANCE guestbook: Overview
DATABASE messages: Overview
TABLE guestbook_message: Data
TABLE
Schema
Indexes
Data
Query
Data
INSERT
EDIT
DELETE
Filter
Filter data

	id	image_uri	message	name
☐	0b4cf9d1-f6b8-45e8-a19b-fa98991fcd18		Hello Cloud Spanner	Ray

8. Click on **Query** tab, and then execute the query given below by clicking on **Run** button:

```
Select * FROM guestbook_message LIMIT 100
```

Copied!
content_copy

Task 9. Redeploy the backend service application to App Engine

In this task, you redeploy the updated guestbook backend service application to App Engine.

1. Switch back to the Cloud Shell tab where the guestbook backend service application is running.

2. Press CTRL+C to stop the application.

3. Make sure you are in the `guestbook-service` directory:

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

4. Use Apache Maven to rebuild the backend service application redeploy it to App Engine:

```
mvn package appengine:deploy -DskipTests
```
Copied!
content_copy
When the deployment completes, the output from Maven provides the URL of the updated backend service application:

```
...
[INFO] GCLOUD: Deployed service [guestbook-service] to [https://guestbook-
service-dot-PROJECT_ID.appspot.com]
[INFO] GCLOUD:
[INFO] GCLOUD: You can stream logs from the command line by running:
[INFO] GCLOUD:   $ gcloud app logs tail -s guestbook-service
[INFO] GCLOUD:
[INFO] GCLOUD: To view your application in the web browser run:
[INFO] GCLOUD:   $ gcloud app browse -s guestbook-service
...
```

5. Use the following command to list the URLfor the updated backend service application:

```
gcloud app browse -s guestbook-service
```
Copied!
content_copy
A clickable URL link to your new backend service appears:

```
Did not detect your browser. Go to this link to view your app:
https://guestbook-service-dot-....appspot.com
```

6. Click the URL link to open the backend guestbook service.

The response lists all the messages in the Spanner database.

7. Switch back to the browser tab for the frontend application.

Note: If you have closed that tab, use the following command to list the URL for the guestbook frontend application that is running on App Engine:`gcloud app browse -s default`. Then click the link to browse to the guestbook frontend application.

8. Enter a message to test that the application is working.

You should now see an updated message list that includes the initial test message you sent from Cloud Shell and the new message you just posted confirming that the updated backend service application is using the new Spanner database.

Task 10. Review

In this lab you created a Spanner instance and database. You used the data definition language (DDL) to create a Spanner table. You also used Spring to add support for Spanner to an application. Finally, you modified a Java application to use Spanner instead of Cloud SQL.