

# How to publish a Spring Boot app (with a database) on the Google Cloud Platform



Wojciech Krzywiec

Oct 15, 2018 · 9 min read

*I've found myself in such situation many times. I was done with my web service project and I wanted to share it with my friend, but my only options were either send a link to my GitHub repository (to deploy it on a local machine), or I bring my laptop to her/him. Not so convenient, right? In this blog post I'll try to fix it, so everyone could check my awesome apps over the Internet. And for that I'll use the Google Cloud Platform.*



"white clouds during daytime" by Kaushik Panchal on Unsplash

As already mentioned I've recently worked on my first Spring Boot RESTful web service Library API. It's a simple app that process few HTTP requests to access/modify

resources that are in MySQL database. It works fine on a local machine, but I want to move into next level to publish it on the Google Cloud Platform.

## But, what's the Google Cloud Platform?

If you already has a Google account you might be familiar with Google Drive service which offers “free” storage space for your files and it can be accessible from any part of the planet. It works, as we call it *Software-as-a-Service (SaaS)*, which means that Google hosts Google Drive application on their servers and allows users to access it from the Internet.

Other delivery model, in cloud computing world, is called *Platform-as-a-Service (PaaS)*, and on the contrary to *SaaS* the provider gives the infrastructure to the customers, so they can deploy and run their application in the Cloud. Google Cloud Platform (GCP), along with Amazon Web Services (AWS) and Microsoft Azure, provides couple services that could be assigned to this category.

One of them is Cloud SQL which is used for MySQL and PostgreSQL database management. Which provides not only the storage, but also other features like backup.

Another one is Google App Engine (GAE). It provides an easy way to deploy an app that is written in languages like Java, Python, C#, PHP, Node.js, Ruby and Go. It also supports Docker images deployment.

With Google App Engine we don't need to worry about:

- infrastructure — Google Cloud Platform is taking care of it,
- versioning — we can deploy and run multiple versions of our app, which is very useful for A/B testing or phased rollout,
- scalability — GAE automatically scales the number of instances of our app based on incoming traffic, so the customer won't experience any slower connection during rush hours,
- logging — we can debug and monitor our application with available tools.

If you want to read more about Cloud SQL and App Engine capabilities check their official documentation [here](#) and [here](#), respectively. Except explaining how they works they also guide how to make a use of each feature with step-by-step approach.

• • •

## First steps

Before I jump into next section, make sure that you have go thru all these steps:

- Create your own Google Account. Registration link is [here](#).
- Create new Google Cloud Platform project [here](#). It's rather straight-forward, but [here](#) is the instruction for more advanced cases.
- You need to have billing enabled for your project. At the time when I'm writing it Google offers a trial version, which is 300\$ for 1 year. It means that you won't be charged for at least a year, if the total cost of all services that you use doesn't exceed 300\$. Also they claim that they won't charge you anything before asking for it. When you read this post trial terms might be different, so check the conditions and pricing for the services (both Cloud SQL and App Engine).
- Enable Cloud SQL Admin API.
- Install Google SDK. On your local machine you need to have a set of tools that allows you to connect with GCP and configure it's properties. It's a simple command line (there is no visual tool for this available).

## Create Cloud MySQL instance

Once we are done with basic set up we can add MySQL instance to the project.

Therefore go to you project dashboard and select SQL from *Navigation menu* (top left corner).

You should get a page with a single window. Click on **Create instance** button and then choose **MySQL** as database engine. On a next page select **MySQL Second Generation** (my database is MySQL 5.7, which is supported only by this generation).

Finally we need to configure the database. My application won't be handling big traffic on run, so I decided to pick the lowest set up I can get (pricing for such are much lower).

Below there are screenshots of my config. To see the whole configuration options click on **Show configuration options**. I've kept almost all properties as defaults, except for *Machine type and storage* and *Backup* feature.



SQL



## Create a MySQL Second Generation instance

## Instance ID

Choice is permanent. Use lowercase letters, numbers, and hyphens. Start with a letter.

library-spring

## Root password

Set a password for the root user. [Learn more](#)

library-spring



Generate

☐ No password

## Location ?

For better performance, keep your data close to the services that need it.

## Region

Choice is permanent

europe-west3

## Zone

Can be changed at any time

Any

[Show configuration options](#)

## 3 Configure machine type and storage

## Machine type ?

For better performance, choose a machine type with enough memory to hold your largest table.



db-f1-micro

vCPUs

1

Memory

614.4 MB

[Change](#)

## Network throughput (MB/s) ?

250 of 2,000

## Storage type ?

Choice is permanent.

☐ SSD (Recommended)

Most popular choice. Lower latency than HDD with higher QPS and data throughput.

☒ HDD

Lower performance than SSD with lower storage rates.

## Storage capacity ?

10 – 3062 GB. Higher capacity improves performance, up to the limits set by the machine type. Capacity cannot be decreased later.

10

GB

☐ Enable automatic storage increasesWhenever you're near capacity, space will be incrementally increased. All increases are permanent. [Learn more](#)

## Disk throughput (MB/s) ?

Read: 1.2

Max: 180.0

Write: 1.2

Max: 75.8

## IOPS ?

Read: 8

Max: 3,000

Write: 15

Max: 15,000

[Close](#)

## 4 Enable auto backups and high availability

## Backups and binary logging

Both options add small performance and monetary costs. [Learn more](#)☐ Automate backups

Automated backups required for binary logging.

☐ Enable binary logging (required for replication and earlier position point-in-time recovery)

## High availability

**Recommended for all production instances to improve fault tolerance. Failover replica is hosted in a different zone from the master and is billed as a separate instance.** [Learn more](#)

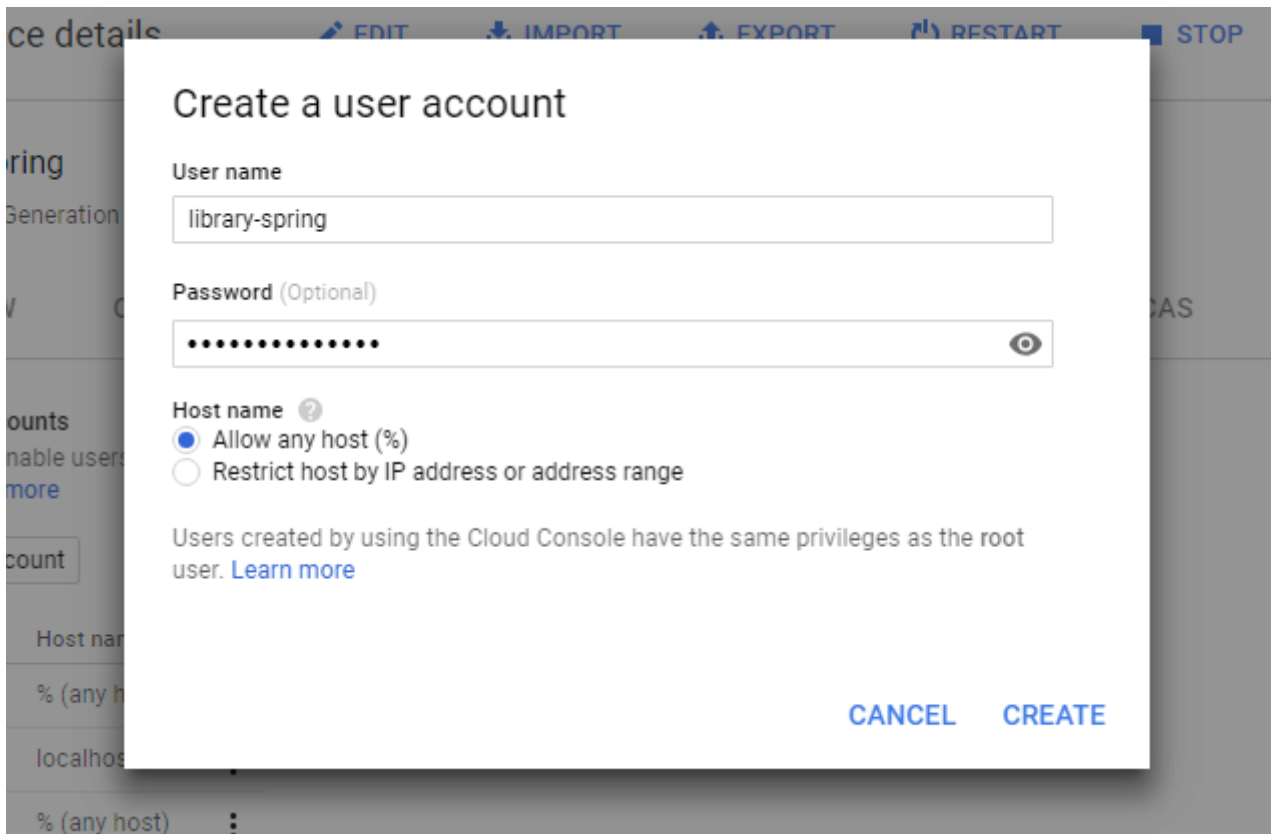
☐ Create failover replica

To create a failover replica, you must first enable automated backups and binary logging. See the Backups and binary logging section below.

[Close](#)

After couple of minutes the Cloud SQL will be up and running.

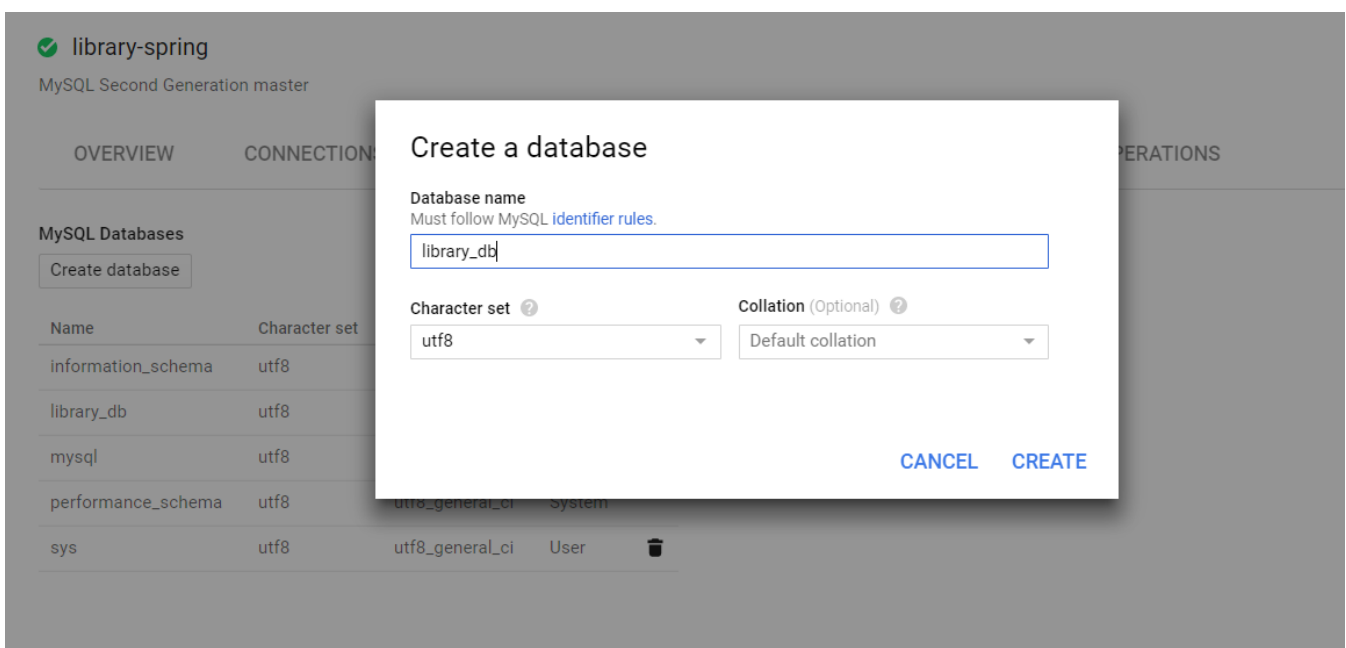
Next, enter SQL instance dashboard, which can be picked from the list of instances. Then, move to **Users** tab and click **Create user account** button, where you can provide username and password.



The screenshot shows a modal dialog titled "Create a user account" overlaid on the Google Cloud SQL console. The dialog has the following fields and options:

- User name:** A text input field containing "library-spring".
- Password (Optional):** A password input field with masked characters (dots) and a toggle icon to show/hide the password.
- Host name:** A section with a help icon and two radio button options:
  - ☒ Allow any host (%)
  - ☐ Restrict host by IP address or address range
- Information:** A note stating "Users created by using the Cloud Console have the same privileges as the root user." with a link to "Learn more".
- Buttons:** "CANCEL" and "CREATE" buttons at the bottom right.

Next go to **Databases** and click **Create database** button. In the pop up provide database name.



The screenshot shows a modal dialog titled "Create a database" overlaid on the Google Cloud SQL console. The dialog has the following fields and options:

- Database name:** A text input field containing "library\_db". A note below the field states "Must follow MySQL identifier rules."
- Character set:** A dropdown menu with "utf8" selected.
- Collation (Optional):** A dropdown menu with "Default collation" selected.
- Buttons:** "CANCEL" and "CREATE" buttons at the bottom right.

In the background, the "MySQL Databases" section is visible, showing a table with columns "Name" and "Character set". The table lists several databases: "information\_schema", "library\_db", "mysql", "performance\_schema", and "sys", all with "utf8" as the character set.

My project has Flyway implemented which will create all necessary tables during app deployment, so I don't need to run any script at this point. But if you would like to run them manually, here are the instructions.

The database is now set up. The only thing that we'll need from this point is the **Instance Connection Name**, which can be read from the SQL Instance Dashboard, and is required to establish connection between app and database.

## Deploy Spring Boot app on Google App Engine

The base project is a Spring Boot app which cannot be simply packed into WAR file and copy-paste into Google App Engine. It requires some modifications, but luckily not so much.

First of all, GAE service uses Jetty webserver/servlet container, but Spring Boot per default uses Tomcat. Therefore we need to update **build.gradle** file.

```
1 compile "org.springframework.boot:spring-boot-starter-web", { exclude group: 'org.springframework.boot:spring-boot-starter-tomcat' }
2
3 compileOnly group: 'javax.servlet', name: 'javax.servlet-api', version: '3.1.0'
4
5 configurations.all {
6     exclude group: 'org.slf4j', module: 'slf4j-log4j12'
7 }
```

build.gradle hosted with ❤ by GitHub

[view raw](#)

With the first line we tell Gradle that we want to exclude the embedded Tomcat dependencies (they will conflict with Jetty), in second we explicitly add Java Servlet dependencies.

Last thing to do is to remove JUL to SLF4J bridge that interferes with App Engine's log handler that's provided through Jetty server.

The app is now running on Jetty server, so we can move on to add Google Cloud dependencies. Below code snippet presents all required plugins and dependencies that needs to be added to Gradle build file. They enable Gradle tasks for GAE deployment and add dependencies that allows internal connection between the GAE app and Cloud SQL. More information about App Engine Gradle tasks can be found [here](#).

```
1 buildscript {
```



```
2     dependencies {
3         classpath 'com.google.cloud.tools:appengine-gradle-plugin:1.+'
4     }
5 }
6
7 apply plugin: 'war'
8 apply plugin: 'com.google.cloud.tools.appengine'
9
10 configurations {
11     providedRuntime
12 }
13 dependencies {
14     compile group: 'com.google.cloud.sql', name: 'mysql-socket-factory', version: '1.0.11'
15 }
16 appengine {
17     deploy {
18         stopPreviousVersion = true
19         promote = true
20     }
21 }
```

Next we need to add **app.yaml** configuration file into the project (into *src/main/appengine* directory). It contains information about the URL, destination GAE service where it will be deployed or general settings for scaling. In my app it looks like as follows:

```
1 runtime: java
2 env: flex
3
4 service: library
5
6 handlers:
7 - url: /*
8   script: this field is required, but ignored
9
10 automatic_scaling:
11     min_num_instances: 1
12     max_num_instances: 5
13
14 resources:
15     cpu: 2
16     memory_gb: 2.3
17     disk_size_gb: 10
18     volumes:
19     - name: hwdisk1
```

```
17 - name: tmpdisk
20   volume_type: tmpfs
21   size_gb: 0.5
```

app.yaml hosted with ❤ by GitHub

[view raw](#)

**Note.** Most of the tutorials over the Internet doesn't include lines with resource settings. I'm doing it because I've faced some difficulties during start up when they have not been provided. But if you work on your own project you probably won't need that.

Last thing is to create a class that extends `SpringBootServletInitializer` class. It is required for traditional WAR file deployment and which is not generated automatically by Spring Boot Initializr. I've decided to not create a new class but to extend the main.

```
1  @SpringBootApplication
2  public class LibraryRestApplication extends SpringBootServletInitializer {
3
4      @Override
5      protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
6          return application.sources(LibraryRestApplication.class);
7      }
8
9      public static void main(String[] args) {
10         SpringApplication.run(LibraryRestApplication.class, args);
11     }
12 }
```

Project is set up, so the only thing to do is to deploy it on GAE. To do that we need to first authenticate ourselves with the Google Cloud Platform using Google Cloud SDK Shell (installed on a locally) and typing following command:

```
gcloud auth application-default login
```

Above command will trigger web-based authentication process, after which you will be able to access GCP from the command line and be able to deploy an app.

As a GCP user you can have multiple projects within it so there is last thing to — we need to explicitly decide into which project we want to deploy a software. Try this:



```
gcloud config set project <PROJECT_ID>  
gcloud config list project
```

Second command prints the default project to operate on.

Finally we can run the *appengineDeploy* Gradle task (from your IDE). You'll need to wait several minutes but after that your app will be successfully deployed 😊.

. . .

So you think that all of these came up really quick for me? No at all ;)

## Lesson learnt

It was first time when I was playing around Google Cloud and I must say that for a first project it was quite challenging.

At the beginning I thought that the whole transition, from local Spring Boot app to Google Cloud deployed one will go really smooth and will be done in a day or two. But nothing goes as it was planned as it should be (but when it does? 😊).

The problems were that even though *appengineDeploy* Gradle task says that build has been successful but it doesn't run. There could be several reasons for that, but here are the problems I've stomp

- This is a **Gradle-based project**, which has far more less examples over the Internet than Maven-based (including official). Their configuration steps differ from each other a little bit.
- This is a **Spring Boot project**, and such has embedded Tomcat server deployment. on the contrary, Google App Engine runs on Jetty web servers , which has shared dependencies with Tomcat that might cause many troubles, so be sure that all of them are removed. Some of the issues that I've stomped on: No Available Context, JuliLog, JSR-356 support unavailable.

- Apart from Tomcat/Jetty server conflict *No Available Context* seems to appear in other situations. It seems that this error is thrown whenever the internal deployment error occurs. App Engine has cool dashboard with error reports where mentioned before errors shows up. The thing I didn't realize is that they are not full logs! To check them you need to go to separate Google Service — **Stackdriver Logging**, which logs all internal errors.

• • •

As usual here is the link to the entire project on GitHub.

### wkrzywiec/Library-API

A REST API for my other project - Library Spring. Contribute to wkrzywiec/Library-API development by creating an...

github.com

• • •

## References

### Google Cloud including GCP & G Suite - Try Free | Google Cloud

Build, innovate, and scale with Google Cloud Platform. Collaborate and be more productive with G Suite. See what's...

cloud.google.com

### Google Cloud SDK Documentation | Cloud SDK | Google Cloud

Edit description

cloud.google.com

### GoogleCloudPlatform/getting-started-java

Contribute to GoogleCloudPlatform/getting-started-java development by creating an account on GitHub

creating an account on GitHub.

github.com

### **GoogleCloudPlatform/java-docs-samples**

Contribute to GoogleCloudPlatform/java-docs-samples development by creating an account on GitHub.

github.com

### **GoogleCloudPlatform/app-gradle-plugin**

Gradle plugin to build and deploy Google App Engine applications. - GoogleCloudPlatform/app-gradle-plugin

github.com

### **Quickstart for Java in the App Engine Flexible Environment | App Engine flexible environment for...**

Now that you know what it's like to develop and deploy App Engine apps, you can stretch out and see the rest of Google...

cloud.google.com

### **Deploy Spring Boot Application in App Engine standard**

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage...

codelabs.developers.google.com

### **GoogleCloudPlatform/cloud-sql-jdbc-socket-factory**

Contribute to GoogleCloudPlatform/cloud-sql-jdbc-socket-factory development by creating an account on GitHub.

github.com

**Using Cloud SQL for MySQL | App Engine standard environment for Java | Google Cloud**

If your App Engine application and Cloud SQL instance are in different Google Cloud Platform projects, you must use a...

[cloud.google.com](https://cloud.google.com)

**Configuring your App with app.yaml | App Engine flexible environment for Java docs | Google Cloud**

You can specify a unique name for your app.yaml files, but then you must specify the file name with the deployment...

[cloud.google.com](https://cloud.google.com)

**Why (and How) We Left App Engine After It Almost Destroyed Us - DZone Cloud**

Last week, I saw this post, which reminded me a lot of our experience with App Engine a few years ago. I shared that in...

[dzone.com](https://dzone.com)

**Spring Boot Reference Guide**

If you are getting started with Spring Boot, or "Spring" in general, start by reading this section. It answers the...

[docs.spring.io](https://docs.spring.io)

**Jetty fails to start spring boot application in appengine flexible**

Note: This is a cross post because I wasn't sure if this was a technical issue or a bug so the bug can be found here A...

[stackoverflow.com](https://stackoverflow.com)

[Google Cloud Platform](#)

[Spring Boot](#)

[App Engine](#)

[Cloud Computing](#)

[Web Development](#)

[About](#) [Help](#) [Legal](#)