(/)

# **Deploy a Spring Boot Application to Google** App Engine

Last modified: February 6, 2020

by Corneil du Plessis (https://www.baeldung.com/author/corneil-duplessis/)

**Cloud (https://www.baeldung.com/category/cloud/)**
**DevOps (https://www.baeldung.com/category/devops/)**
**Spring (https://www.baeldung.com/category/spring/)** +

## 1. Overview

In this tutorial, we'll show how to deploy an application from our Bootstrap a Simple Application using Spring Boot (https://www.baeldung.com/spring-boot-start) tutorial to App Engine (https://cloud.google.com/appengine/) on Google Cloud Platform.

As part of this we'll:

- Configure Google Cloud Platform Console and SDK
- Use Cloud SQL to create a MySQL instance
- Configure the application for Spring Cloud GCP
- Deploy the application to App Engine and test it

## 2. Google Cloud Platform Configuration

We can use the GCP Console to get our local environment ready for GCP. We can find the installation process (https://cloud.google.com/sdk/) on the official website.

Let's create a project on GCP using the GCP Console (https://console.cloud.google.com/):

```
gcloud init
```

Next, let's configure the project name:

```
gcloud config set project baeldung-spring-boot-bootstrap
```

Then we'll install the App Engine support and create an App Engine instance:

```
gcloud components install app-engine-java
gcloud app create
```

Our application will need to connect to a MySQL database within the Cloud SQL environment. As Cloud SQL doesn't provide a free tier we'll have to enable billing (https://cloud.google.com/billing/docs/how-to/modify-project) on the GCP account.

We can check available tiers easily:

```
gcloud sql tiers list
```

Before continuing, we should use the GCP Website to enable the Cloud SQL Admin API (https://console.cloud.google.com/flows/enableapi?apiid=sqladmin).

Now we can create a MySQL instance and database in Cloud SQL (https://console.cloud.google.com/sql/instances) using the Cloud Console or the SDK CLI. During this process, we'll choose the region and provide an instance name and database name. It's important that the app and the database instance are in the same region.

Since we're going to deploy the app to *europe-west2*, let's do the same for the instance:

```
# create instance
gcloud sql instances create \
  baeldung-spring-boot-bootstrap-db \
    --tier=db-f1-micro \
    --region=europe-west2
# create database
gcloud sql databases create \
  baeldung_bootstrap_db \
    --instance=baeldung-spring-boot-bootstrap-db
```

## 3. Spring Cloud GCP Dependencies

Our application will need dependencies from the Spring Cloud GCP (https://cloud.spring.io/spring-cloud-gcp/) project for the cloud-native APIs. For this, let's use a Maven profile named *cloud-gcp*:

```
<profile>
  <id>cloud-gcp</id>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-gcp-starter</artifactId>
      <version>1.0.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-gcp-starter-sql-mysql</artifactId>
      <version>1.0.0.RELEASE</version>
    </dependency>
```

Then we add the App Engine Maven plugin:

```
    <build>
      <plugins>
        <plugin>
          <groupId>com.google.cloud.tools</groupId>
          <artifactId>appengine-maven-plugin</artifactId>
          <version>1.3.2</version>
        </plugin>
      </plugins>
    </build>
</profile>
```

## 4. Application Configuration

Now, let's define the configuration that allows the application to use the cloud-native resources like the database.

Spring Cloud GCP uses *spring-cloud-bootstrap.properties* to determine the application name:

```
spring.cloud.appId=baeldung-spring-boot-bootstrap
```

We'll use a Spring Profile named *gcp* for this deployment and we'll need to configure the database connection. Therefore we create *src/main/resources/application-gcp.properties*:

```
spring.cloud.gcp.sql.instance-connection-name=\
    baeldung-spring-boot-bootstrap:europe-west2:baeldung-spring-boot-bootstrap-db
spring.cloud.gcp.sql.database-name=baeldung_bootstrap_db
```

## 5. Deployment

The Google App Engine provides two Java environments:

- the *Standard* environment provides Jetty and JDK8 and the *Flexible* environment provides just JDK8 and
- the Flexible environment is the best option for Spring Boot applications.

We require the *gcp* and *mysql* Spring profiles to be active, so we provide the *SPRING_PROFILES_ACTIVE* environmental variable to the application by adding it to the deployment configuration in *src/main/appengine/app.yaml*:

```
runtime: java
env: flex
runtime_config:
  jdk: openjdk8
env_variables:
  SPRING_PROFILES_ACTIVE: "gcp,mysql"
handlers:
- url: /.*
  script: this field is required, but ignored
manual_scaling:
  instances: 1
```

Now, **let's build and deploy the application using the *appengine* maven plugin**:

```
mvn clean package appengine:deploy -P cloud-gcp
```

After deployment we can view or tail log files:

```
# view
gcloud app logs read

# tail
gcloud app logs tail
```

Now, **let's verify that our application is working by adding a book**:

```
http POST https://baeldung-spring-boot-bootstrap.appspot.com/api/books \
        title="The Player of Games" author="Iain M. Banks"
```

Expecting the following output:

```
HTTP/1.1 201
{
    "author": "Iain M. Banks",
    "id": 1,
    "title": "The Player of Games"
}
```

## 6. Scaling the Application

**The default scaling in App Engine is automatic.**

It may be better to start with manual scaling until we understand the runtime behavior, and the associated budgets and costs involved. We can assign resources to the application and configure automatic scaling in *app.yaml*:

```
# Application Resources
resources:
  cpu: 2
  memory_gb: 2
  disk_size_gb: 10
  volumes:
  - name: ramdisk1
    volume_type: tmpfs
    size_gb: 0.5
# Automatic Scaling
automatic_scaling:
  min_num_instances: 1
  max_num_instances: 4
  cool_down_period_sec: 180
  cpu_utilization:
    target_utilization: 0.6
```

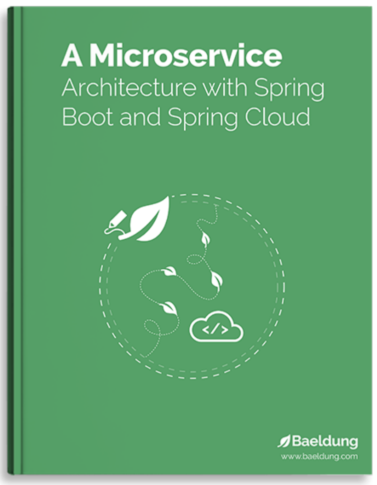
## 7. Conclusion

In this tutorial, we:

- Configured Google Cloud Platform and the App Engine
- Created a MySQL instance with Cloud SQL
- Configured Spring Cloud GCP for using MySQL
- Deployed our configured Spring Boot application, and
- Tested and scaled the application

We can always refer to Google's extensive App Engine documentation (https://cloud.google.com/appengine/docs/flexible/java/) for further details.

The complete source code of our examples here is, as always, over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-boot-modules/spring-boot-bootstrap).



**Build your Microservice Architecture with**

Spring Boot and Spring Cloud

Enter your email address

**>> Download Now**

Comments are closed on this article!

**CATEGORIES**

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

**SERIES**

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

**ABOUT**

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
JOBS (/TAG/ACTIVE-JOB/)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)