

<https://codelabs.developers.google.com/codelabs/cloud-app-engine-springboot/index.html?index=..%2F..%2Fspringone#0>

# 1. Overview

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go.

App Engine applications automatically scale based on incoming traffic. load balancing, microservices, authorization, SQL and NoSQL databases, Memcache, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

App Engine's environments, the [Standard Environment](#) and the [Flexible environment](#), support a host of programming languages, including Java, Python, PHP, NodeJS, Go, etc.. The two environments give users maximum flexibility in how their application behaves since each environment has certain strengths. Read [The App Engine Environments](#) for more information.

In this codelab, you will learn how to deploy a Spring Boot application into App Engine Standard environment on Google Cloud Platform via the web. This environment will scale down to 0 instances when no one is using it, and automatically scale up!

## What you'll learn

- How to create a Spring Boot Java application on Google App Engine.

## What you'll need

- A Google Cloud Platform Project
- A Browser, such [Chrome](#) or [Firefox](#)
- Familiarity with standard Linux text editors such as Vim, EMACs or Nano

# 2. Setup and Requirements

## Self-paced environment setup

If you don't already have a Google Account (Gmail or Google Apps), you must [create one](#). Sign-in to Google Cloud Platform console ([console.cloud.google.com](https://console.cloud.google.com)) and create a new project:



## Select

Create project +

### New Project

**Project name** ?

my-kubernetes-codelab

Your project ID will be my-kubernetes-codelab-1217 ? [Edit](#)

[Show advanced options...](#)

Create Cancel

Remember the project ID, a unique name across all Google Cloud projects (the name above has already been taken and will not work for you, sorry!). It will be referred to later in this codelab as PROJECT\_ID.

Next, you'll need to [enable billing](#) in the Developers Console in order to use Google Cloud resources. Running through this codelab shouldn't cost you more than a few dollars, but it could be more if you decide to use more resources or if you leave them running (see "cleanup" section at the end of this document). Google Kubernetes Engine pricing is documented [here](#). New users of Google Cloud Platform are eligible for a [\\$300 free trial](#).

## Google Cloud Shell

While Google Cloud and Kubernetes can be operated remotely from your laptop, in this codelab we will be using [Google Cloud Shell](#), a command line environment running in the Cloud.

### Activate Google Cloud Shell

From the GCP Console click the Cloud Shell icon on the top right toolbar:



Then click "Start Cloud Shell":

## Google Cloud Shell

Free, pre-installed with the tools you need for the Google Cloud Platform. [Learn More](#)

```
example-vm-2    europe-west1-b f1-micro      10.240.119.112 104.155.36.122 RUNN
example-vm-3    us-central1-f  f1-micro      10.240.57.1    104.154.76.241 RUNN
google77703_student@cloudshell:~$
google77703_student@cloudshell:~$ git clone https://github.com/GoogleCloud/appengine-examp
Cloning into 'appengine-example'...
remote: Counting objects: 476, done.
remote: Total 476 (delta 0), reused 0 (delta 0), pack-reused 476
Receiving objects: 100% (476/476), 432.65 KiB | 0 bytes/s, done.
Checking connectivity... done.
google77703_student@cloudshell:~$ cd appengine-example
google77703_student@cloudshell:~/appengine-example$
```

### Real Linux environment

- Linux Debian-based OS
- 5GB persisted home directory
- Add, edit and save files

### Configured for Google Cloud

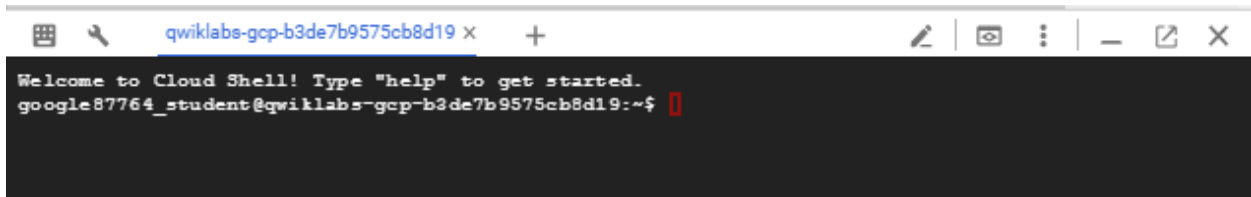
- Google Cloud SDK
- Google App Engine SDK
- Docker
- Git
- Text editors
- Build tools
- View more [↗](#)

### Popular language support

- Python
- Java
- Go
- Node.js

[CANCEL](#) [START CLOUD SHELL](#)

It should only take a few moments to provision and connect to the environment:



This virtual machine is loaded with all the development tools you'll need. It offers a persistent 5GB home directory, and runs on the Google Cloud, greatly enhancing network performance and authentication. Much, if not all, of your work in this lab can be done with simply a browser or your Google Chromebook.

Once connected to the cloud shell, you should see that you are already authenticated and that the project is already set to your *PROJECT\_ID*:

`gcloud auth list`

Command output

Credentialed accounts:

```
- <myaccount>@<mydomain>.com (active)
```

Note: gcloud is the powerful and unified command-line tool for Google Cloud Platform. Full documentation is available from <https://cloud.google.com/sdk/gcloud>. It comes pre-installed on CloudShell. You will notice its support for tab-completion.

```
gcloud config list project
```

Command output

```
[core]
```

```
project = <PROJECT_ID>
```

If it is not, you can set it with this command:

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

Command output

```
Updated property [core/project].
```

### 3. Create a new Spring Boot Web Application

After Cloud Shell launches, you can use the command line to generate a new Spring Boot application with Spring initializr:

```
$ curl https://start.spring.io/starter.tgz -d packaging=war \
```

```
-d dependencies=web -d baseDir=gae-standard-example | tar -xzvf -
```

```
$ cd gae-standard-example
```

### 4. Update Maven pom.xml

There are multiple ways to deploy a Java server application - either by using a Maven or Gradle plugin, or by deploying the war package directory. In the code lab, we'll use Maven to deploy the application.

#### Add App Engine Plugin

Update the pom.xml to include a Google Cloud Platform plugin that simplifies the deployment process. You can use vim,nano,or emacs to edit the file.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
```

```
...
```

```
<build>
```

```

<plugins>
...
<plugin>
  <groupId>com.google.appengine</groupId>
  <artifactId>appengine-maven-plugin</artifactId>
  <version>1.9.60</version>
</plugin>
...
</plugins>
</build>
</project>

```

## Remove Tomcat Starter

App Engine Standard uses Jetty web server underneath. Spring Boot includes Tomcat in the WAR package that will conflict with Jetty. Exclude Tomcat from the Spring Boot Starter, and add back Servlet API as a provided dependency:

pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
...
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>

```

```
<scope>provided</scope>
</dependency>
</dependencies>
</project>
```

## Exclude JUL to SLF4J Bridge

Spring Boot also includes a JUL to SLF4J bridge that interferes with App Engine's log handler that's provided through Jetty server. To get proper log entries, you must exclude the jul-to-slf4j artifact by marking it as provided.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
...
<dependencies>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jul-to-slf4j</artifactId>
    <scope>provided</scope>
  </dependency>
</dependencies>
</project>
```

## 5. Add App Engine Descriptor

To deploy the application into App Engine standard, you must add create a new src/main/webapp/WEB-INF/appengine-web.xml descriptor file:

```
$ mkdir -p src/main/webapp/WEB-INF/
$ touch src/main/webapp/WEB-INF/appengine-web.xml
```

Edit src/main/webapp/WEB-INF/appengine-web.xml file and add the following content:

```
src/main/webapp/WEB-INF/appengine-web.xml
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>REPLACE_WITH_YOUR_PROJECT_ID</application>
  <version>1</version>
  <threadsafe>true</threadsafe>
```

```
<runtime>java8</runtime>
```

```
</appengine-web-app>
```

Your project ID can be retrieved from the command line: `gcloud config list --format 'value(core.project)'`

## 6. Add a Controller

Add a new controller that returns "hello" in `DemoApplication.java`.

```
src/main/java/com/example/demo/DemoApplication.java
```

```
package com.example.demo;
```

```
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.web.bind.annotation.*;
```


```
@SpringBootApplication  
@RestController  
public class DemoApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(DemoApplication.class, args);  
    }  
}
```

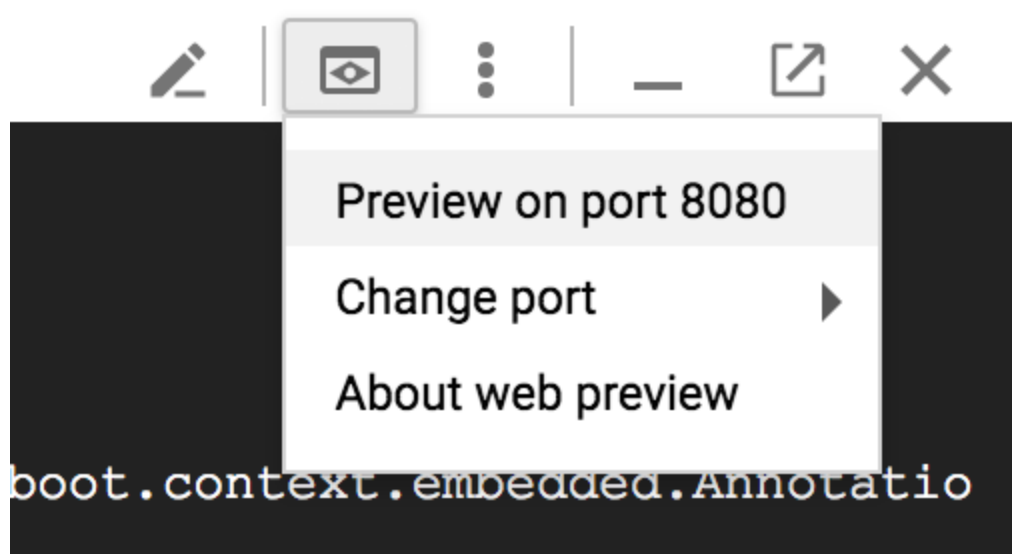
```
@GetMapping("/")  
public String hello() {  
    return "hello world!";  
}  
}
```

## 7. Run the Application Locally

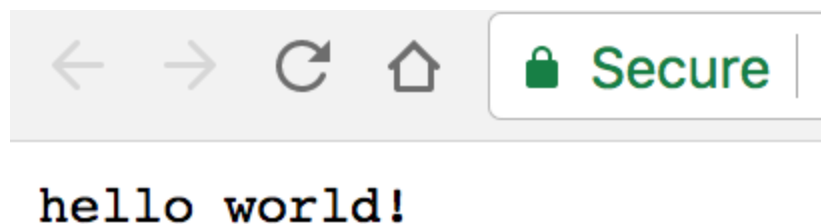
You can start the Spring Boot application normally with the Spring Boot plugin:

```
$ ./mvnw -DskipTests appengine:devserver
```

Once the application started, click on the Web Preview icon  in the Cloud Shell toolbar and choose preview on port 8080.



A tab in your browser opens and connects to the server you just started.



## 8. Deploying the Application into App Engine

First, initialize the Project to be able to run App Engine applications. We'll initialize the project to run in the US Central region:

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

```
You are creating an app for project [...].
```

```
WARNING: Creating an App Engine application for a project is irreversible and the region
```



cannot be changed. [More](#) information about regions is at <https://cloud.google.com/appengine/docs/locations>

You can alternatively list all available regions ``gcloud app regions list`` and pick one.

Then, deploy your application into App Engine environment, run `mvn appengine:deploy`:

```
$ ./mvnw -DskipTests appengine:update
```

```
[INFO] Running -A ... -V 1 --oauth2 update ...
```

The following URL can be used to authenticate:

```
https://accounts.google.com/o/oauth2/...
```

Attempting to open it in your browser now.

Unable to open browser. Please open the URL above and copy the resulting code.

[Open the previous URL and paste the code here, press ENTER]

First time deployment may take several minutes. This is because App Engine Flexible environment will automatically provision a Google Compute Engine virtual machine for you behind the scenes, and then installing the application, and starting it. If your deployment timed out, please let the instructor know.

After the application is deployed, you can visit it by opening the URL `http://<project-id>.appspot.com` in your web browser.

If you don't remember your project-id, please refer to the "Setup and Requirements" section or check the deployment logs for something similar to this:

Deployed service [default] to [`https://<project-id>.appspot.com`]

Optionally, you can instead purchase and use a top-level domain name for your app, or use one that you have already [registered](#).

