



---

[HOME](#) > [BLOG](#) > Automatic deployment to H...

# Automatic deployment to Heroku CI/CD Spring Boot + Maven + Github Actions

6/1/21

In this article, I'm going to show you how to automatically deploy your SpringBoot apps, test with Github Actions, and then, once all your tests have successfully executed, **deploy to your Heroku account**.

By Omar Bautista

Omar Bautista is a Software Engineer with 10+ years of experience developing with Java. One of his passions is to constantly learn about different topics, his favorite language for the JVM is Groovy.

---

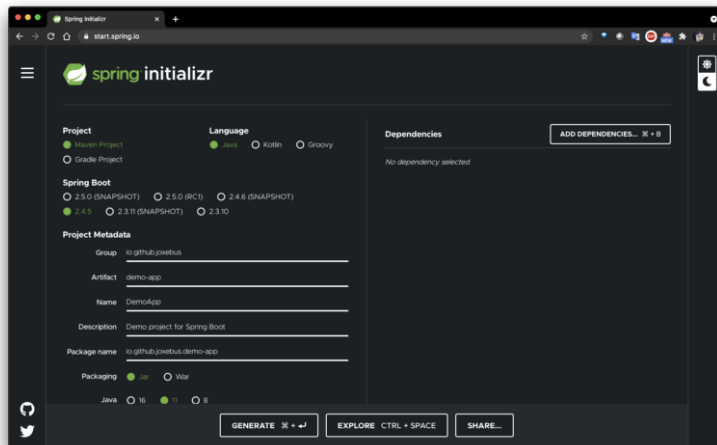
TAGS



different kinds of projects such as Java, Ruby, PHP, Angular, React, etc. They have free tiers and payment tiers, for our purposes, we are going to use a free tier.

- **SpringBoot:** We should not confuse Spring with SpringBoot, the former is the framework which is composed of several modules, persistence, web development, dependency injections and much more. The latter is an extension of Spring that helps us create Spring apps quickly and have by default certain behaviors that we can override if we need to. For this example, we are going to use SpringBoot to create a simple web app.
- **Github:** Is a git server on the cloud and managed by Microsoft, we can create free repositories to store our projects there. For this example, we are going to use a free Github Account to create the repository where we are going to store our project.
- **Github Actions:** Github provides a set of actions that can help us have a full development environment to deploy our project, execute tests, create pipelines or workflows and execute it on different environments such Linux, Windows or even MacOS. For this example, we are going to use it to execute our tests.
- **CI/CD:** Continuous integration and continuous delivery are two DevOps concepts that help us to have better control of the quality of our development process, execute tests, have automatic deployments, etc. In this example, we are going to see how to automatically deploy our code once the tests have been executed successfully using all the technologies mentioned above.

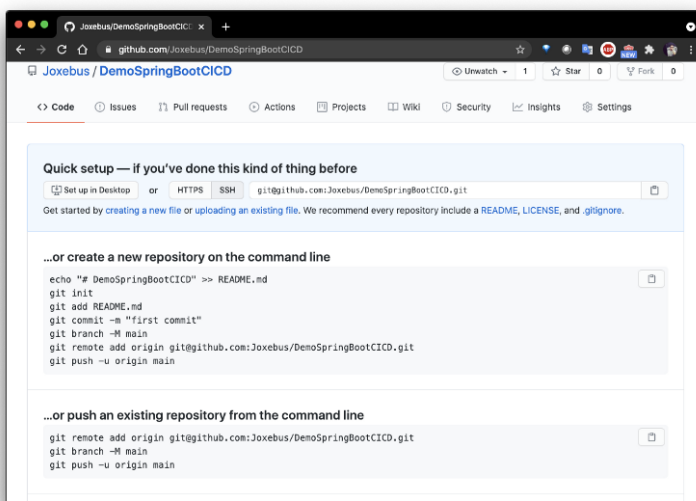
First, you need to create or use an existing SpringBoot project. if you don't have a SpringBoot project, then go to this page: <https://start.spring.io/>



Spring Initializr

## Create a project on Github

Go to your GitHub account and create a new project, follow the instructions and push your project here.



Sample GitHub project

Add a controller to your application. If you don't have a controller in your application, then first you need to



pom.xml

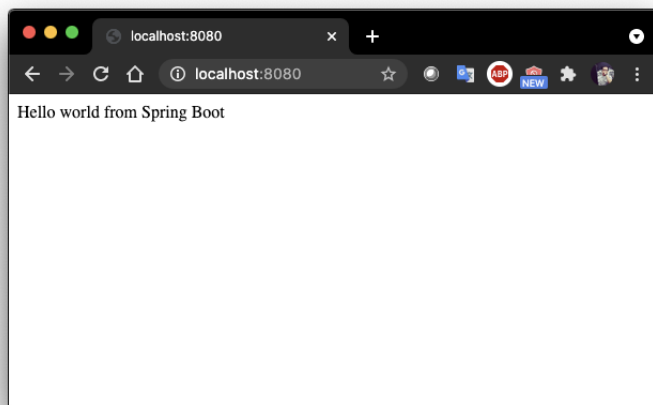
```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-web</artifactId>
4 </dependency>
```

Then, you need to create a controller to return a string when accessing the root of your project.

DemoController.groovy

```
1 import org.springframework.web.bind.annotation.RequestMapping;
2 import org.springframework.web.bind.annotation.RequestMethod;
3 import org.springframework.web.bind.annotation.RestController;
4
5 @RestController
6 public class DemoController {
7
8     @RequestMapping(value = "/", method = RequestMethod.GET)
9     public String index() {
10         return "Hello world from Spring Boot";
11     }
12 }
```

This is the result of doing that:



Sample message when accessing port 8080 on localhost

Push your changes to your GitHub project.

## Configure GitHub Actions



the following configuration on our **maven.yml**

```
maven.yml

1  name: Demo SpringBoot CI/CD
2
3  on: [push]
4
5  jobs:
6    build:
7      runs-on: ubuntu-latest
8
9      steps:
10       - uses: actions/checkout@v2
11       - name: Set up JDK 11
12         uses: actions/setup-java@v2
13         with:
14           java-version: '11'
15           distribution: 'adopt'
16       - name: Build with Maven
17         run: mvn --batch-mode --update-snapshots verify
```

Push this change to your repository and automatically you will see that Github Action triggered.

Add workflows to execute maven verify



Joxebus committed 1 minute ago ✓

Build triggered automatically after pushed your commit

## Create a project on Heroku

Open your Heroku account and create a new project, if you don't have an account you can create one for free here: <https://signup.heroku.com/>

On your Heroku dashboard, select the option **New > Create new app** and put the name you want.



United States

Add to pipeline...

Create app

Create a new app on Heroku

Once your project has been created, we need to define the build pack for Maven Java applications, first you are going to see a screen like this on the settings tab.

Personal > demo-springboot-ci-cd ☆ Open app More

Overview Resources Deploy Metrics Activity Access Settings

App Information

App Name

demo-springboot-ci-cd

Region

United States

Stack

heroku-20

Framework

No framework detected

Slug size

No slug detected

Heroku git URL

<https://git.heroku.com/demo-springboot-ci-cd.git>

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Reveal Config Vars

Buildpacks

Buildpacks are scripts that are run when your app is deployed. There's one buildpack for each language.

Add buildpack

Heroku project settings tab

Click on the **Add buildpack** button and select Java and save the changes.

Add Buildpack ×

Enter Buildpack URL

heroku/java

Or select from our officially supported buildpacks

nodejs

python

php

ruby

java

go

gradle

scala

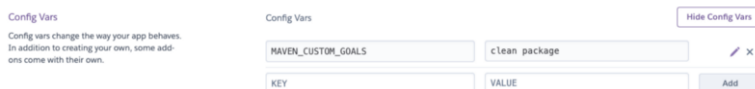
clojure

Save changes



Now click on the **Reveal env vars** button, here we need to configure a variable that will be taken by Heroku to build and deploy our application with maven.

Add the env variable **MAVEN\_CUSTOM\_GOALS** then set the value to **clean package** should look like the image below.



The screenshot shows the Heroku Config Vars interface. On the left, there's a 'Config Vars' section with a 'Reveal Env Vars' button. The main area shows a table with one row: 'MAVEN\_CUSTOM\_GOALS' with the value 'clean package'. There are 'KEY' and 'VALUE' input fields at the bottom, and an 'Add' button.

Environment variables section

## Configure your project to work with Heroku env variables

Now that we have our project created on Heroku, we need to create a **Procfile** in our spring boot project, this is because with that we tell how to run our application once it is created.

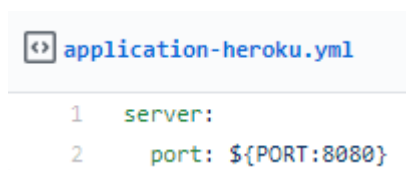
```
1 web: java $JAVA_TOOL_OPTIONS -jar target/demo-app-0.0.1-SNAPSHOT.jar --spring.profiles.active=heroku
```

Your **Procfile** should contain the script to run your application in this case we are going to deploy our artifact that is generated after performing a **package** of our project, this is the jar generated on the target folder with name **<artifactId>-<version>.jar**



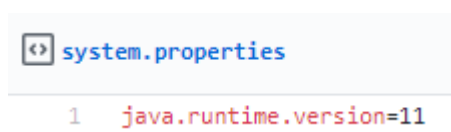
jar in target folder after package command executed

We need to create a YAML file for the Heroku profile, this file should be called **application-heroku.yml** as we can see in the previous example, this is the name of the **spring active profiles** that we are sending in our **Procfile**, inside this YAML file this is the configuration that we must set:



The PORT is a variable provided by Heroku for our project so we don't need to define this env var, once you finish with this configuration, the **Procfile** and **application-heroku.yml**

By default Heroku uses **Java 8** to compile your project if you are using a higher version you need to create a **system.properties** file and setup the Java version there like this:



Add the files to your git project and push to GitHub you will see that a Github Action automatically starts, this will happen with every new commit you push to your repository.

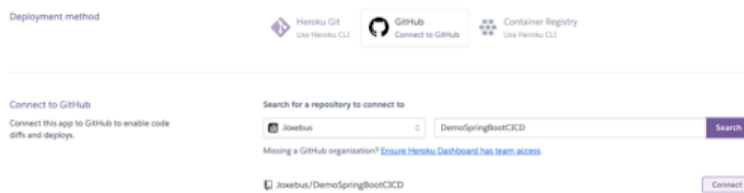




## automatically deploy

Now is time to link our projects and deploy only on successful executions of the Github Action, to do this, go to your Heroku application, and select the **Deploy** tab, under deployment method, select GitHub and give the permissions with your Github Account to connect.

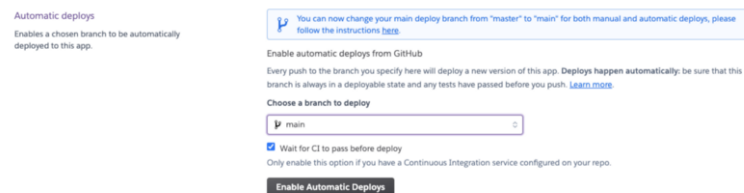
Search your project and click on **connect**.



### Github Deployment Method

After selecting your project, a new section will appear, in this case, the section **Automatic and Manual deployment**. We are going to select **Automatic** and our only branch which is **main** but in case you have a specific branch, for example, **deploy** then you can select that one. This means that every time that branch has a new commit and the test has been executed successfully then the application will be deployed with no need for doing a manual deployment.

To finish just click on the **Enable Automatic Deploy** and let the magic begin!



### Before enabling automatic deploys



### After enabling automatic deploys



Setting up this configuration, you can run manually the first deployment by clicking on the button **Deploy Branch** on the section **Manual Deployment** and next time your branch has a new commit then the deployment will be automatic.

#### Manual deploy

Deploy the current state of a branch to this app.

#### Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more.](#)

Choose a branch to deploy

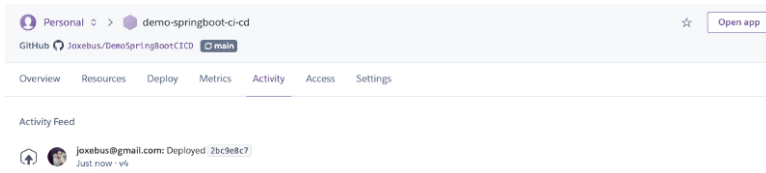
main

Deploy Branch

Manual deploy

## Verify your project has been deployed

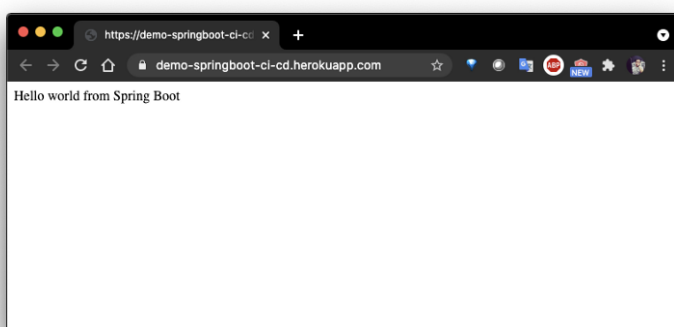
In your **Activity** tab on your Heroku project, you will be able to see if your project has been deployed successfully. If you see that your project has been **deployed**, then you can click on the button **Open app** to see the results.



Deployed successfully

You should be able to see something like this, in the case of my applications this is the URL

<https://demo-springboot-ci-cd.herokuapp.com/>





<https://github.com/Joxebus/DemoSpringBootCI/CD>

## COMMENTS

[Comments](#)[Community](#)[Privacy Policy](#)[Log in](#)[Favorite](#)[Tweet](#)[Share](#)[Sort by latest](#)

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)

Name

Be the first to comment.



+1 788 884 4703



PART OF  
**Digital Wire**  
Group

© Nearsure 2021