Setting up a CI/CD pipeline for free using Travis CI & Heroku

Andreas Guldborg Heick, Mohammad Hariri, Rasmus Jarnborg Friis

December 11, 2020

Abstract

With CI/CD[1] growing in demand, and somewhat mandatory in larger projects, it is important for a developer to have a simple pipeline ready for a project. CI/CD can be expensive and not all developers need a fancy technical solution. In this article we showcase a free solution, that gives an example of how easy CI/CD can be done free-of-charge using Travis CI and Heroku. This solution allows the developer to focus on the productivity aspect of software development instead of the manually repeating tasks, like deployment.

Introduction

In this article we want to take a look at and solve the common issue that is Continuous Integration (CI) and Continuous Deployment (CD), and how to do it fast, easy and most importantly free-of-charge. This article will explore how a student or a small-time developer can create a CI/CD pipeline for a smaller non-commercial open-source project, without spending any money. There are many paid solutions out there, but if you are a student, hobby developer or you are working on something where you just need to deploy without too much hazzle, then you might need a free solution.
For this we have chosen to look at Travis CI and Heroku with GitHub. We won't look into many of the features that both services provide, we will only look at how to build with Travis CI and deploying upon success of the build to Heroku, giving the developer a public deployed environment to work on. It is expected that the reader knows basic development concepts and technologies, like CI/CD and GitHub. We will however explain the basics of Travis CI and Heroku, and give an example of how easy they are to use.

[1]Continuous Integration (CI) / Continuous Deployment (CD)

Travis CI

Travis CI is a hosted continuous integration service that embraces open source projects. It is built on containerization architecture with docker to take advantage of isolation and the ability of reproducing the same result multiple times. Travis accepts a wide spectrum of languages which makes it a valid option for most cases.
Travis was used because it is free [1] to use for open source and private projects, and it is easy to integrate with GitHub, and we have used it for a previous project where we needed something fast and easy, that didn't cost anything.

Heroku

Heroku is a cloud platform as a service (PaaS) provider, that lets companies and individuals build, deploy, monitor and scale apps in a fast and effortless way. Heroku is built to make all these things easy and fast using their behind the scenes infrastructure, that removes the difficult setups normally needed for a deployment. Heroku applications run in a collection of lightweight Linux containers called dynos, that contain the users' apps. Heroku provides different tiers of dynos, for every type of deployment, from test and hobby projects to professional production apps.

We chose to look at and use Heroku for continuous deployment, because it offers a free tier [2], that provides most of the basic functions needed for a simple and easy test deployment. We liked Heroku because it has some tools that are very easy to use, and because it auto-detects your setup for deployment.

Continuous Integration with Travis CI

The basics for setting up Travis CI are very simple, which is also why we have chosen Travis. Firstly you need to connect Travis with GitHub, something you can do as you sign up for Travis CI. After you have connected Travis, you can select which GitHub repositories you want to use Travis for, and then you can finally use Travis for your project. All you have to do after connecting with GitHub, is adding a .travis.yml file to your GitHub repository, which is the file that Travis looks for and uses to do its magic.

If you have any issues, want a detailed walk-through or want to understand and learn more about how Travis CI works, then go to: https://docs.travis-ci.com/user/tutorial/

Here is an example of a .travis.yml file:

```yaml
language: java
jdk:
  - openjdk15
before_install:
  - chmod +x mvnw
script:
  - ./mvnw clean install
```

Listing 1: .travis.yml

In this example we are using a Java Spring Boot setup, and have defined language, JDK and some scripts for installing everything correctly on the Travis container. The exact code is not that important, because it depends on your project setup, and every Travis configuration will look different depending on what languages and frameworks etc. you are using.
That's it, now you have a basic and free continuous integration setup, that will run on every commit to a connected GitHub repository. To see if the build passes you need to check out the build status page, by opening the Travis CI dashboard, and selecting your projects GitHub repository from the list.

Continuous Deployment with Heroku

Now that we have a continuous integration setup, it is time to add the continuous deployment part.

The first thing that is needed is a Heroku account, after you have an account you can start making apps. There are several ways to add a project to Heroku and create an app, the recommended way is to use Heroku CLI. If you prefer a GUI you can also use Heroku's dashboard, it displays your apps and allows you to do the same things as the Heroku CLI. We suggest that you authorize Heroku on GitHub, as this allows you to add projects directly from GitHub and create an app from a repository.

For a detailed walk-through and explaination of Heroku CLI, use this link: https://devcenter.heroku.com/start

One clever thing about Heroku is that it can auto-detect your app setup using buildpacks that are used to install dependencies and configure your environment, as long as you use a supported language and setup. This happens automatically when you create and deploy an app.

Linking Travis with Heroku

Adding continuous deployment to an application is really simple after Travis CI is set up and a Heroku app has been deployed. All that is needed is to add a deploy command to your project's .travis.yml file, and get your Heroku API key.

```yaml
language: java
jdk:
  - openjdk15
before_install:
  - chmod +x mvnw
script:
  - ./mvnw clean install

deploy:
  provider: heroku
  api_key: $HEROKU_API_KEY
  app:
    branch-name: app-name
```

Listing 2: .travis.yml v2

The difference between this .travis.yml and the old one, is that we added a deploy section to the config. The deploy section contains the configuration for deployment, and contains a provider, an API key and an app containing branch- and the app name. The provider is the provider of the deployment, in our case Heroku. The API key is used by Heroku, to verify who you are, and in a Travis context, it is used by Travis to tell Heroku that you are behind this deployment, allowing Travis to send deploy commands to Heroku.

Getting the Heroku API key is really simple, and only requires you to go to your Heroku account settings, and copying the API key listed in the bottom.

The app is where you specify which app to deploy to, and this is defined by providing the branch name of the branch you wish to deploy, and the app name which defines exactly which app you wish to deploy to.
One important consideration for the deploy configuration, is how you present the API key. We recommend that you serve it as an environment variable, otherwise your API key will be presented in plain text in your GitHub repository, allowing anyone else to copy and abuse it. This can be achieved by using Travis environment variables, found in the settings section of your Travis repository. You can then call the variable as we have shown in our .travis.yml example, by writing $variablename at the api_key field. In our example we made an environment variable named HEROKU_API_Key.

And that's how you create a CI/CD pipeline using Travis CI and Heroku, all there is left to do is triggering a build on Travis CI, by either doing it manually or committing a change to your GitHub repository, and then everything will be deployed to your Heroku app automatically, without spending as much as a dime.

Conclusion

As we have demonstrated in our example, this solution is very easy and fast to implement and use, and it doesn't require much technical expertise. Most of the setup is signing up for services, and connecting them. The actual technical setup is very easy and only requires knowledge about your own project, and what you want to write it in. The solution we provided is 100% free which was the main objective we stated, so both ease of use, speed and cost was within our solution scope.
There are of course other solutions that tries to solve the same problems. The most obvious one is that Heroku them-selves have a CI tool available however that tool does not solve our problem fully because it is not free. Another tool you could use would be Jenkins this tool is also free but the problem here is that you need to host the CI tool yourself making it a lot harder to setup and use for nontechnical people. So other tools are definitely available, but they do not necessarily solve all the problems we wanted to solve, which makes them irrelevant.

We managed to find some CI and CD services and tools, that even combine CI/CD and provide everything needed free:

- Netlify
- Buddy.works

Both seem pretty promising and definitely deserve a look, but we haven't looked very much into them.

What's next?

After you have set up a CI/CD pipeline, you are ready to code and you can now use more of the potential that Travis CI and Heroku provide you with. For Travis CI it would very much make sense, if you added some tests to your build setup, as this is when Travis will really begin to shine. It will allow you to make sure that nothing is deployed, before your tests have all been successfully run.

References

[1] Travis ci - plans, 2020. https://travis-ci.com/plans.

[2] Heroku - pricing, 2020. https://www.heroku.com/pricing.