Most of us agree that C*ontinuous Integration (CI)* , C*ontinuous Delivery (CD)*, cloud infrastructure, test automation, and configuration management make up the basics of devops. Depending on the scale of your project, CI/CD may be a hassle to set up and difficult to implement. Nevertheless, it is definitely necessary and here's why:

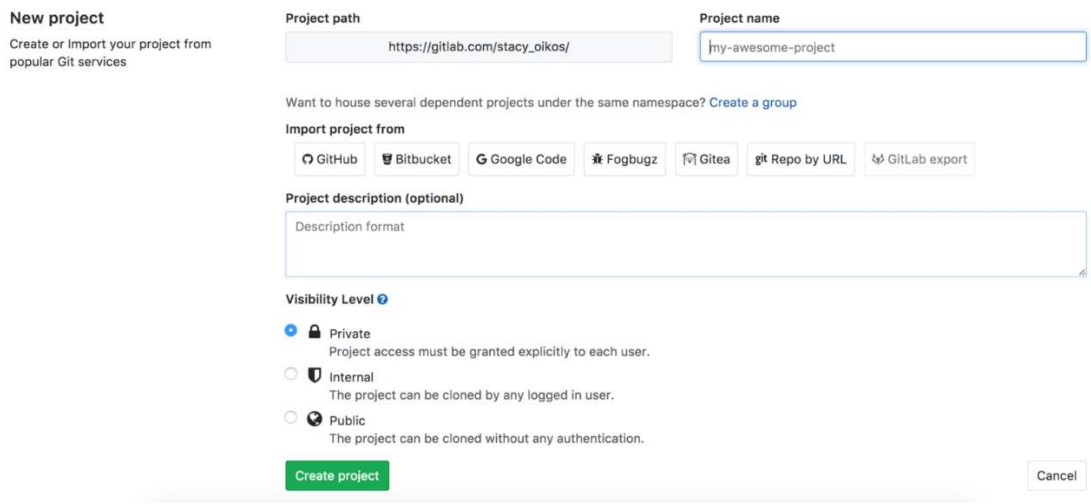
In an Agile environment, requirements evolve quickly over time and to ship out features which are bug-free, having a suite of automated tests and integrating code continuously is of utmost important. Having a proper CI/CD environment provides you the *confidence* to experiment, implement new features, and push out updates quickly.

In fact, with proper CI, bugs can be caught earlier and developers can review code faster as changes to code in a repository are usually merged a few times a day so it can be continuously validated.

By following this article, you will be equipped with the basic knowledge to be able to **set up a CI/CD environment** on **Gitlab** and deploy your **NodeJS**project to **Heroku** in less than an hour. Of course, we will not be going in-depth on how to write unit tests and that can be further explored in another article.

Step 1: Setup a Gitlab account

Sign up for an account and create a new project. Name your project and choose whether it is private/public.



When you first create your project, you will need to add your ssh keys to it.

Step 2: Add SSH keys

To create a pair of SSH keys, follow the steps below:

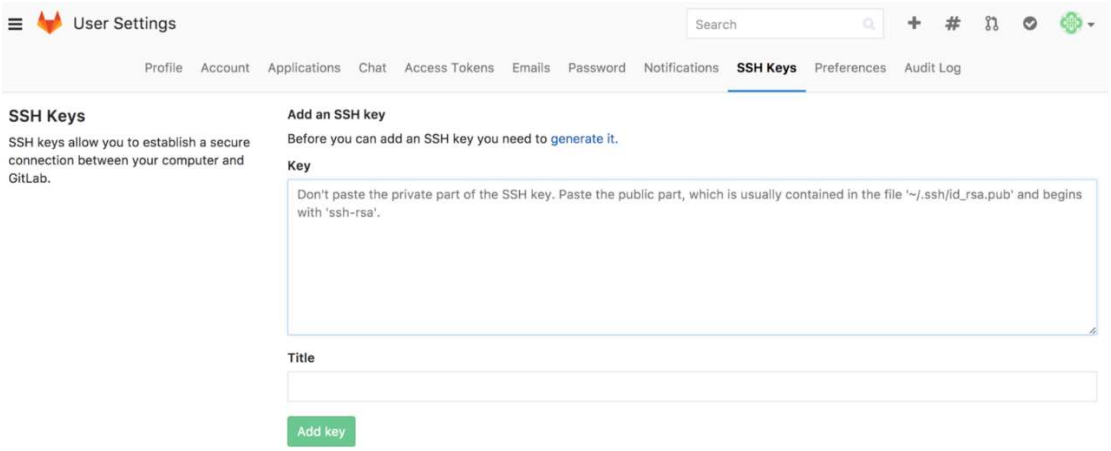
```
ssh-keygen -t rsa
```

Then enter the file to save your SSH keys in.

```
Enter file in which to save the key (/home/demo/.ssh/id_rsa):
```

Lastly, you can choose to have a passphrase or not. Once done, copy the public key and paste it in gitlab.

```
cat ~/.ssh/id_rsa_gitlab.pub | pbcopy
```



After clicking “Add key”, store the key in your keychain with the command:

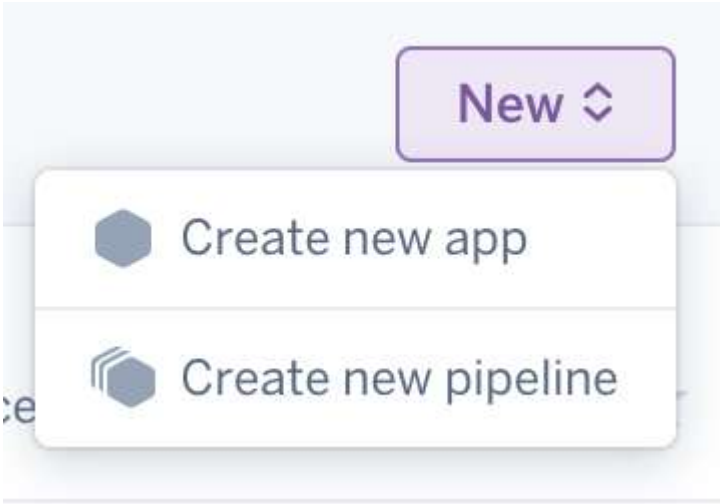
```
ssh-add  ~/.ssh/id_rsa_gitlab
```

Configure SSH to always use the keychain. If you do not already have a `config` file in `.ssh`, create an `~/.ssh/config` file. In other words, in the `.ssh`directory in your home dir, make a file called `config`. In that `.ssh/config` file, add the following lines:

```
Host gitlab.com
  UseKeychain yes
  AddKeysToAgent yes
  IdentityFile ~/.ssh/id_rsa_gitlab
```

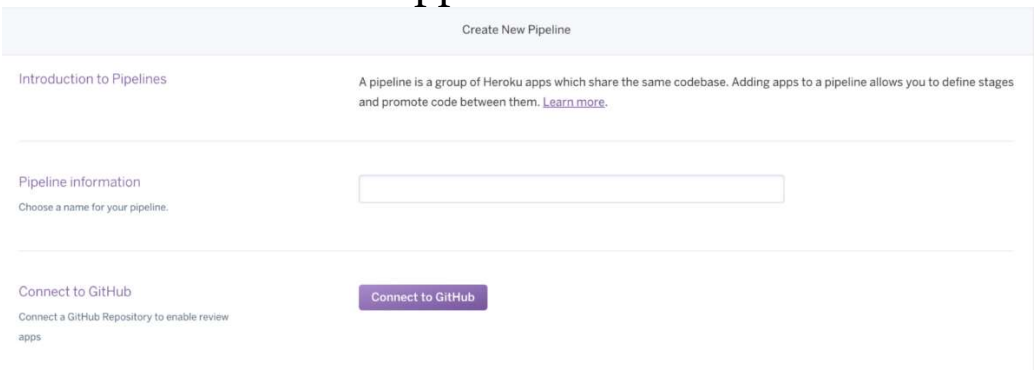
sources: https://apple.stackexchange.com/questions/48502/how-can-i-permanently-add-my-ssh-private-key-to-keychain-so-it-is-automatically

Step 3: Setup staging and production environments

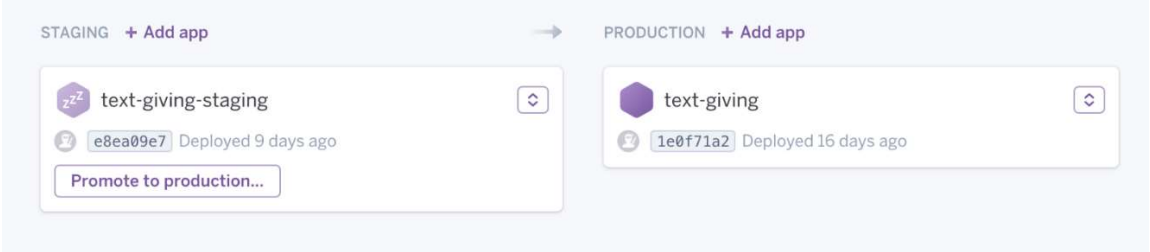


In Heroku, create a pipeline which will hold both staging and production apps.

Inside the pipeline, create both the staging and productions apps. Note, it will be good if the name of the staging app can be appended with the word ‘staging’ at the back to clearly differentiate the two apps.



After setting up the apps, it will look something like this.



Step 4: Setup your tests

In the root directory of the app, create a folder called `tests`, and add a test file in the folder. For me, I use both **chai** (assertion library) and **mocha** (JS framework). You can use other assertion libraries such as Jasmine etc if you prefer.

```
npm install mocha chai --save-dev
```

Add a script to package.json so you can simply run `npm test` next time.

```
"test": ./node_modules/.bin/_mocha --recursive
./tests/*.test.js
```

Add a file in `tests` called `me.test.js`

```
describe('me', () => {
  it('is awesome', () => {
   expect(...).to.be....
  })
});
```

As I mentioned earlier, we will not be going in-depth into how to write the tests.

Step 5: Setting up your CI environment (like finally!)

To setup a CI environment on Gitlab, add a `.gitlab-ci.yml` in the root of your repository.

This file contains definitions of how your project should be built and your Gitlab Runner (which you will be setting up later) will search for this file in your repository and execute it within the environment of the Runner.

This is how I setup my `.gitlab-ci.yml`. You may wish to tweak it depending on your project. But of course, before you push up your project to Gitlab, it is good practice to run it by the YAML validator first to make sure it's valid, if not, it will result in errors. I will explain the various components in my YAML file.

```
image: node:6.10.3

stages:
  - ver
  - init
  - tests
  - deploy

ver:
  stage: ver
  script:
    - node --version
    - whoami
init:
  stage: init
  script:
    - npm cache clean
    - rm -rf node-modules
    - npm install
run_tests:
  stage: tests
  script:
    - npm test

deploy_staging:
  stage: deploy
  script:
    - git remote add heroku
https://heroku:$HEROKU_API_KEY@git.heroku.com/text-giving-
staging.git
    - git push heroku master
    - echo "Deployed to staging server"
  environment:
  name: staging
  url: https://text-giving-staging.herokuapp.com/
  only:
    - master

deploy_production:
  stage: deploy
  script:
```

```
    - git remote add heroku
https://heroku:$HEROKU_API_KEY@git.heroku.com/text-giving.git
    - git push heroku master
    - echo "Deployed to production server"
  environment:
  name: production
  url: https://text-giving.herokuapp.com/
  when: manual
  only:
    - master
```

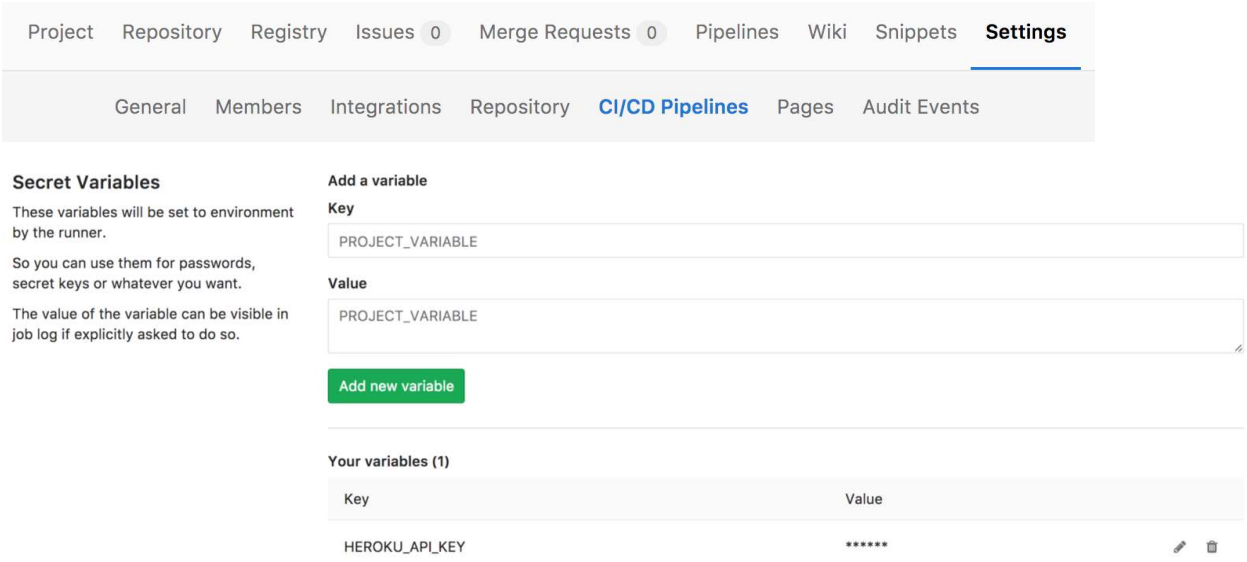
image: allows you to specify a certain version of NodeJS you want to use during build time
stages: define build stages. In this case, we have `ver`, `init`, `tests`, and `deploy`, but you can change the names of the stages to what you deem fit for your project.
stage ver: `node --version` allows us to check the version of NodeJS we are using and `whoami` reveals whether the user has permissions, which makes it easier to debug when things go wrong.

stage init: we clean the npm cache, remove the node modules, install it and then we run the tests which we have written previously.

stage tests: we want to run the tests to make sure all the tests pass.

stage deploy: we will add a remote to the repository using Heroku API key. We can retrieve the Heroku API key from **Accounts setting** page in Heroku.

Then, go to Gitlab settings > CI/CD pipelines > Secret variables and add HEROKU_API_KEY as a Secret Variable.



We can then refer to the Heroku API key as `$HEROKU_API_KEY` in our YAML file.
Notice we have two deploy stages—one for **staging**, one for **production**. We have also set it to manually push to production by adding `when: manual` in the YAML file. Hence, if all is successful, it will push the `master` branch automatically to `staging`, and we will need to push manually to `production`if everything passes.

Step 6: Install a Gitlab Runner

As mentioned, in GitLab, Runners run the jobs that you define in `.gitlab-ci.yml`. Since I'm using a MacOS, I will be following the instructions
here: https://docs.gitlab.com/runner/install/osx.html. It should be fairly straightforward to follow.

Step 7: Register the Gitlab Runner

Next, register the Gitlab Runner by following instructions here: https://docs.gitlab.com/runner/register/index.html. Again, it should be fairly straightforward to follow. For the runner executor, I chose to run it on **shell** which is the easiest

to configure (you may choose other options if you prefer). After registering, install and start the service with the following commands:

```
gitlab-runner install
gitlab-runner start
```

You may check if the runner is running with the following command:

```
gitlab-runner status
gitlab-runner: Service is running!
```