

VUE.JS ENTERPRISE CRASH COURSE - PDF

LESSON 5

CONTINUOUS DEPLOYMENT



Lesson 5

Continuous Deployment

Welcome to the fifth and final lesson in this crash course on enterprise web development with Vue.js where we'll be discussing deployment.

A good deployment process will allow you to push changes to your app easily and as often as you need to, while at the same time minimizing downtime and making it hard for you to screw it up.

One way to achieve this is through a system called *Continuous Deployment*, more commonly shortened to "CD".

In this lesson, we're going to learn how continuous deployment works and also see how to set it up in a full-stack Vue project.

What is continuous deployment?

CD provides a reliable deployment process by:

- Automating deployment steps
- Preventing deployment without a successful build and passing tests

The easiest way to explain continuous deployment is that it's a workflow that helps ensure the integrity of the live app as new code changes are introduced.

It does this in two main ways...

Automate deployment steps

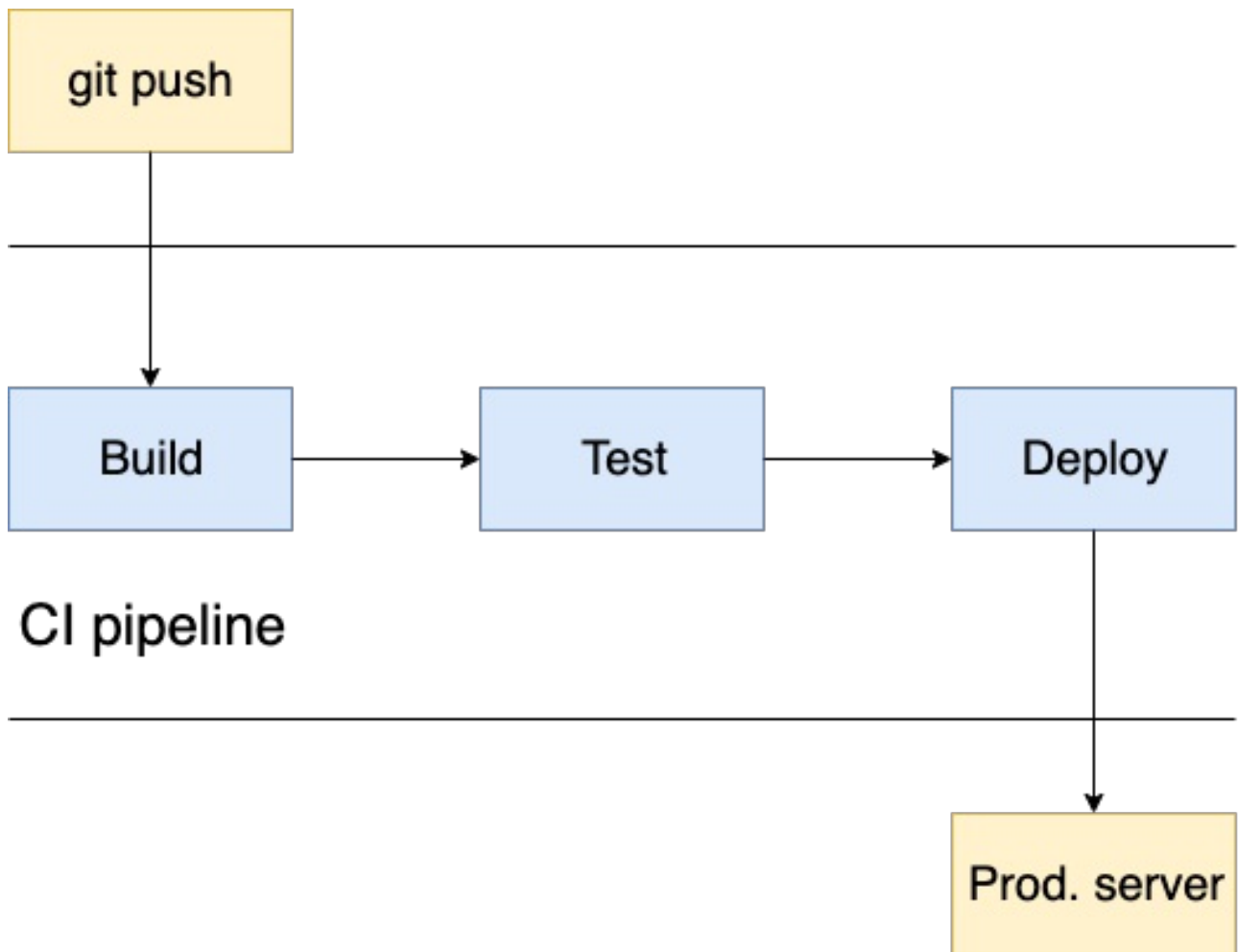
Firstly, CD allows us to automate all the deployment steps, including checking out of the code, installing dependencies, setting environment variables, copying files and so on.

This means we no longer have to rely on our developers remembering to do all this in each deploy, and will therefore minimize human error.

Conditional deploy

Secondly, CD will only complete a deploy conditional on the app building and tests passing. So, if a developer updates the code but this causes, say, a unit test to fail, the pipeline will abort before the deployment process begins, preventing the bug from entering the live app.

Continuous deployment workflow



Let's now talk more specifically about how CD works.

Once we've set up our pipeline, we can commit a code change to a special deployment git branch, then push it to the CD server.

This push triggers the various jobs we've set up, collectively known as the CD pipeline.

The first stage of the pipeline is to build the app inside a virtual machine on the CD server.

The second stage is to run all the tests, including both client and server unit tests, and the E2E tests.

Finally, the CD server will initiate a deployment of the app to our production server.

Continuous Deployment services

- Circle CI
- GitLab CI
- Bitbucket Pipelines
- Travis CI

And more.

While you can set up your own CD server, the easiest way to get started with CD is to use a Continuous Integration SaaS product.

In this video, I'm going to show you an example of CD using GitLab CI, but there are other similar products including Circle CI, and Bitbucket Pipelines.

YAML config file

.gitlab-ci.yml

```
image: node:10.15.3

cache:
  paths:
    - node_modules
  key: ${CI_COMMIT_REF_SLUG}

variables:
  MONGO_DB_URI: mongodb://mongo:27017/full-stack-vue

build:
  script: npm install
...
```

I said before that an important part of how continuous deployment works is by automating the steps of deployment.

The means to this is the CD configuration file that goes in the root of your project directory.

In the case of GitLab, this YAML file will be named *.gitlab-ci.yml*, and is in the YAML format.

If you've never used the YAML format before, it's a declarative file format that's great for configuration. The syntax is essentially just a short-hand of JSON.

Let's now go through some of the main configuration you'll want to add to this file.

Image

.gitlab-ci.yml

```
image: node:13.8.0
```

The first stage of the CD pipeline is to build the app in a virtual machine. So first you need to tell your CD service the virtual environment it should use for this purpose.

Most CD services will allow you to do this by providing a Docker image. If you aren't familiar with Docker, it's a tool for packaging software for use in virtual environments.

Docker Hub provides an open source collection of Docker images, so you can select one from there that fits your needs, or even upload your own.

I've added an `image` property to my YAML file, and specified an image here called `node:13.8.0`.

If you check Docker Hub, you'll see that this image is an Ubuntu Linux image with Node.js version 13.8.0, which is the key piece of software we need in a full-stack Vue + Node app.

You ideally want to choose an image here that is identical, or at least as close as possible, to your production environment.

Jobs

.gitlab-ci.yml

```
image: node:13.8.0
```

```
build:
```

```
  script: npm install
```

Now that we've selected a virtual environment, we need to tell our CD server what it needs to do. The main way we do that with GitLab CI, at least, is through "jobs".

Name

A job is declared as a top-level property in the config file. In this case, I've created one called `build`.

Script

All jobs need to have a property `script` which is a command that should be run from the terminal of your virtual environment.

For the build job, all we need to do is simply run `npm install`.

Jobs

.gitlab-ci.yml

```
...

build:
  ...

unit-test-client:
  script: npm run test:unit:client

unit-test-server:
  ...

e2e-test:
  ...
```

In addition to the build job, we'd also want to define jobs to run our various tests as well.

Deploy

.gitlab-ci.yml

```
...

deploy:
  stage: deploy
  image: tmaier/dpl
  script: dpl --provider=heroku --app=$HEROKU_APP_NAME --api-key=$HEROKU_API_KEY
```

And finally, we need to define a deploy job which initiates the deployment.

dpl

I've used a special image for this job which includes a tool called **dpl** which is great for deploying to cloud platforms like Heroku.

Stages

.gitlab-ci.yml

```
build:
  stage: build
  ...

unit-test-client:
  stage: test
  ...

unit-test-server:
  stage: test
  ...

e2e-test:
  stage: test
  ...

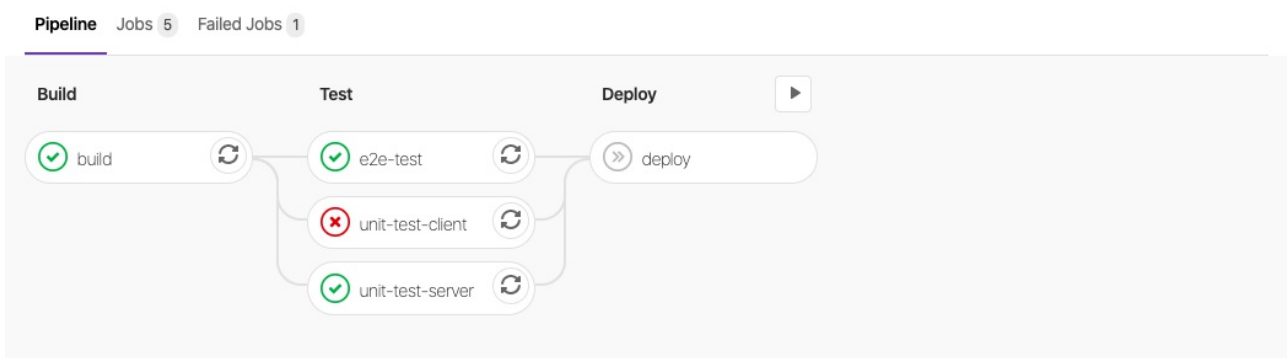
deploy:
  stage: deploy
  ...
```

It's important that the various jobs of our pipeline are grouped into stages, and that these stages happen in a defined order.

Why is this important? Well, each stage should only begin if and when the previous one completes.

In our case, we'll want a build stage, a test stage, and a deploy stage, to complete in that order. We can tell the CD server which stage each job belongs to by adding a `stage` property.

Failed deploy



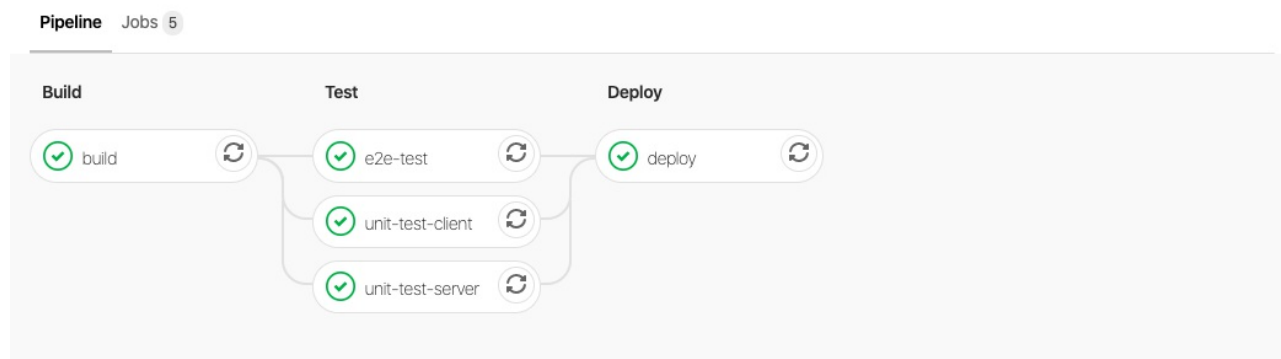
Here's a little snippet from the user interface of GitLab CI, where we can see the outcome of a recent attempt to deploy changes to an app.

You can see here that the build stage passed, then in the test stage the e2e-test and unit-test-server jobs passed.

But the unit-test-client job failed. And, and this is important, the deploy step was skipped as a result.

That's how continuous deployment provides a fail safe against faulty deploys - as the pipeline is abandoned if any job fails.

Successful deploy



After I fixed the error that caused the client unit tests to fail, and re-ran the pipeline, it has now passed, allowing the deployment of the code to the production environment where it gets built into the live app.

Two special announcements...

1. My new program launches next week!
2. A contest with prizes!

That's brings us to the end of the discussion on continuous deployment but we're not at the end of the course just yet.

There are two more very exciting things I want to now announce.

Firstly, I have a new program coming out next week that expands on everything we've been discussing in this crash course. I'll tell you more about that in a moment.

Secondly, I'm announcing a special contest for students of this crash course which I encourage you to join.

The winner is going to receive a free place in my new program. But most importantly, this contest is going to require you to implement what you've learned in the course.

Entering the contest is simple, but it's not necessarily easy. It will probably take you an hour or so to create your entry.

We'll come back to the contest in a moment.

What will be your next step...

in learning enterprise web development?

Hopefully what we've covered in the crash course has given you a strong foundational knowledge of enterprise web development topics.

Of course, there's a lot more to know about these topics, and we didn't have time to cover every topic - for example, we didn't cover authentication, or documentation, or a lot of other important things.

So what should you do next if you're serious about taking a leap in your career and not only learning but mastering a whole range of enterprise web development topics?

You have two options...

1. Go it alone
2. Let me guide and fast track you

Well, there are two options.

You can use what I've taught you so far as a basis and go out and start learning the rest by yourself.

Or, you can stick with me and let me guide you and fast-track you.

Introducing "Enterprise Vue"



If the second option sounds good to you, then you'll want to check out my newest program *Enterprise Vue*.

This program is for web developers who are serious about advancing their career by mastering the core enterprise web development skills.

In it, you'll work with me to build a real-world, full-stack Vue enterprise app with testing, authentication, cloud deployment, documentation and a lot more.

What's included

- Online mentoring program

- 10+ hours of video material
- Blueprint of a complete enterprise app

So what's included in the program?

Firstly, we have a weekly mastermind with me and your new developer network to collaborate, discuss, and have all questions answered.

Secondly, there's over 10 hours of video lessons that go deep into the theory of enterprise topics, and guide you step-by-step through the case-study project.

You're also going to get a complete commercial-grade app blueprint that you'll learn intimately over the program and can use as a basis for your own company or startup projects.

Launching next week...

Keep an eye on your email!

As I said before, the course doesn't open until next week, so be sure to keep an eye out for the launch announcement some time next week.

Want to WIN your place in *Enterprise Vue*?

In the meantime, you should work on your entry for the crash course contest. Like I said, I will be giving a free enrollment as a prize to one lucky person.

How the contest works

So how does the contest work?

Well, if you haven't seen it already, I've provided a source code repository for the crash course on GitLab. This source code is a working boilerplate of a full-stack Vue and Node app with tests and continuous deployment config.

To enter the contest, you need to make a fork of that repository in GitLab, and you need to get the continuous deployment pipeline passing.

To make sure it's not too easy, I also want you to enhance the project in some way. For example, you can add a new unit or E2E test - just make sure the pipeline still passes.

So don't worry if you're not sure what to do yet, because for the remainder of this video I'm going to cover the steps you need to take.

1. Create a GitLab account

Sign up · GitLab

gitlab.com/users/sign_up

Speed up your DevOps with GitLab

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

Register for GitLab

First name Last name

Username

Email

Password

Minimum length is 8 characters

☐ I accept the [Terms of Service and Privacy Policy](#)

☐ I'd like to receive updates via email about GitLab.

If you haven't already, you'll firstly need to create a free GitLab account by registering at:

https://gitlab.com/users/sign_up (https://gitlab.com/users/sign_up).

2. Fork source code repo

Anthony Gore / mevn-enterprise-boilerplate

gitlab.com/anthonygore/mevn-enterprise-boilerplate

mevn-enterprise-boilerplate

Project ID: 17618995 | [Request Access](#)

12 Commits 1 Branch 0 Tags 338 KB Files 524 KB Storage

master mevn-enterprise-boilerplate / + History Find file Web IDE Clone

Update README.md
Anthony Gore authored 19 hours ago

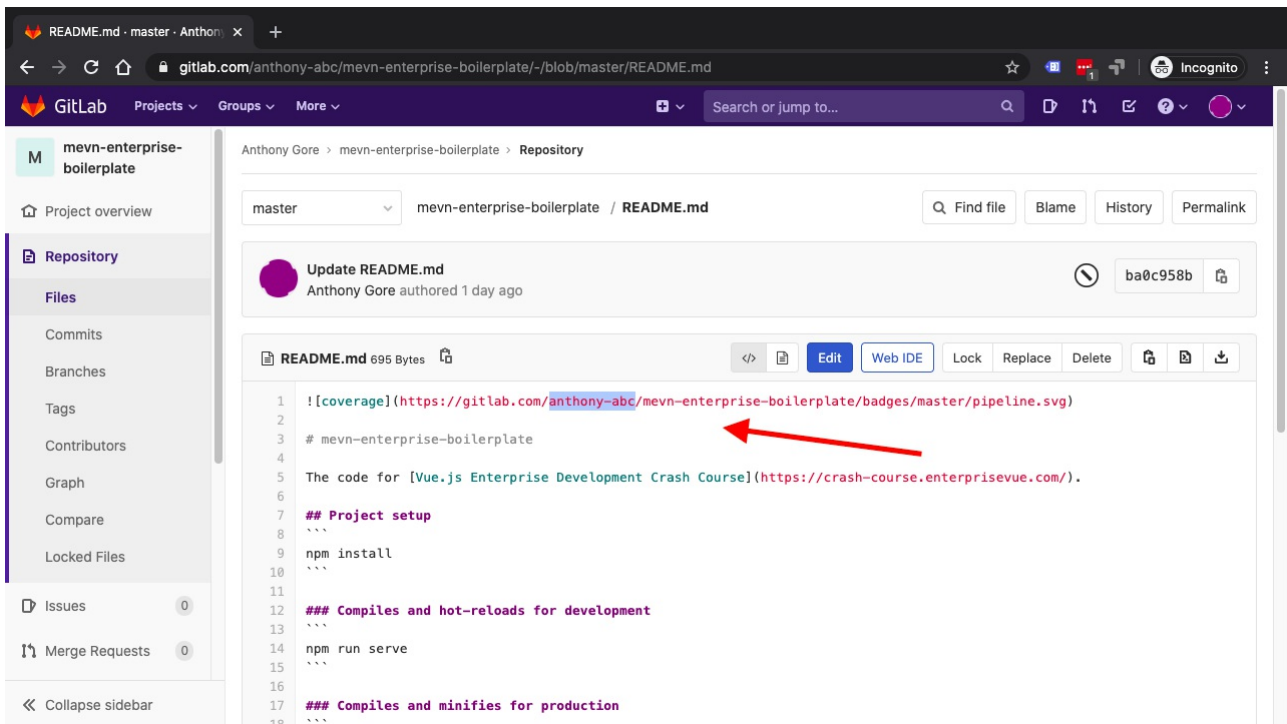
README CI/CD configuration No license. All rights reserved

Name	Last commit	Last update
public	init	1 month ago
server	Getting tests working	1 month ago
src	init	1 month ago
tests	Getting tests working	1 month ago

Once you've created an account, go to the source code repository I made for the crash course, and make a fork by clicking the fork button.

<https://gitlab.com/anthonygore/mevn-enterprise-boilerplate> (<https://gitlab.com/anthonygore/mevn-enterprise-boilerplate>)

3. Add pipeline badge to fork README file



Next, you need to edit the README file and edit the pipeline status badge. This badge tells visitors at a glance whether or not the CD pipeline is passing.

What you need to do is modify the URL so that it points to your fork. To do that, change the account name to your account name.

<https://gitlab.com/<your-account-name>/mevn-enterprise-boilerplate/badges/master/pipeline.svg>

Now you can save the README file.

```
14
15 unit-test-client:
16   stage: test
17   script: npm run test:unit:client
18
19 unit-test-server:
20   stage: test
21   script: npm run test:unit:server
22   services:
23     - mongo:3.6.11
24   variables:
25     NODE_ENV: test
26
27 e2e-test:
28   stage: test
29   script: npm run test:e2e
30   image: circleci/node:10.15.3-browsers
31   services:
32     - mongo:3.6.11
33   variables:
34     NODE_ENV: production
35
36 deploy:
37   stage: deploy
38   image: tmaier/dpl
39   script: dpl --provider=heroku --app=$HEROKU_APP_NAME --api-key=$HEROKU_API_KEY
```

Next, you'll see in the root of the source code that we have a GitLab CI YAML file. This is the file that defines the CD pipeline.

In the deploy stage of the pipeline, you'll see that it deploys to Heroku, and it uses two GitLab environment variables - `$HEROKU_APP_NAME` and `$HEROKU_API_KEY`.

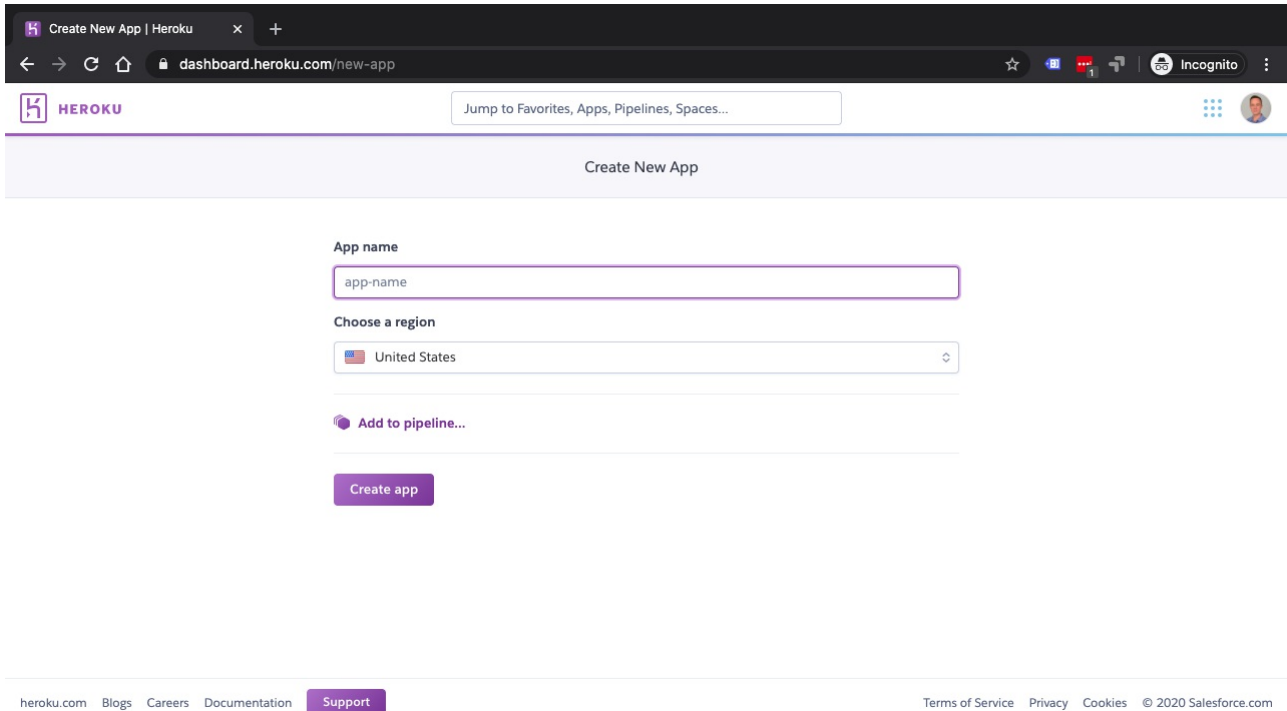
You'll need to create a Heroku app and set values for those variables in GitLab in order for the pipeline to pass.

4. Create a Heroku account

If you haven't already, create a free account with Heroku at:

<https://signup.heroku.com> (<https://signup.heroku.com>).

5. Create a Heroku app



HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Create New App

App name

app-name

Choose a region

United States

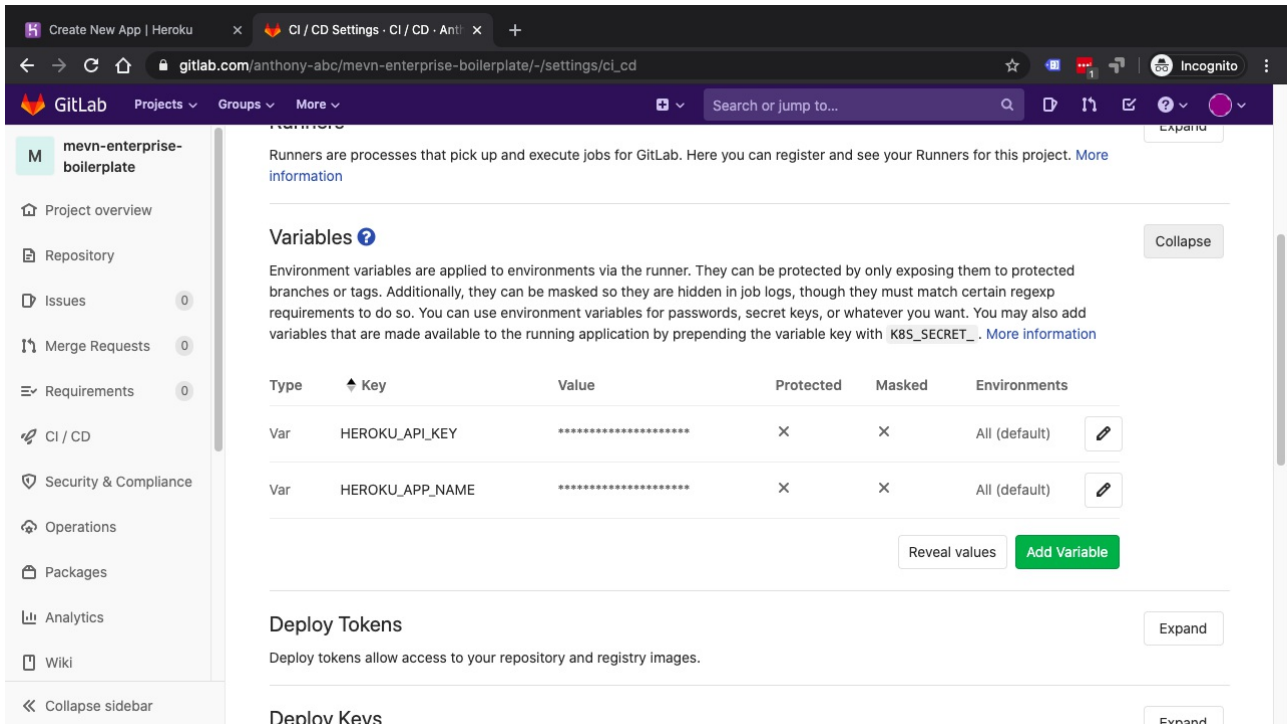
Add to pipeline...

Create app

heroku.com Blogs Careers Documentation Support Terms of Service Privacy Cookies © 2020 Salesforce.com

Next, create a new Heroku app.

6. Add environment variables to GitLab



GitLab

mevn-enterprise-boilerplate

Project overview

Repository

Issues

Merge Requests

Requirements

CI / CD

Security & Compliance

Operations

Packages

Analytics

Wiki

Collapse sidebar

Runners

Runners are processes that pick up and execute jobs for GitLab. Here you can register and see your Runners for this project. [More information](#)

Variables

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they can be masked so they are hidden in job logs, though they must match certain regex requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. [More information](#)

Type	Key	Value	Protected	Masked	Environments
Var	HEROKU_API_KEY	*****	X	X	All (default)
Var	HEROKU_APP_NAME	*****	X	X	All (default)

Reveal values Add Variable

Deploy Tokens

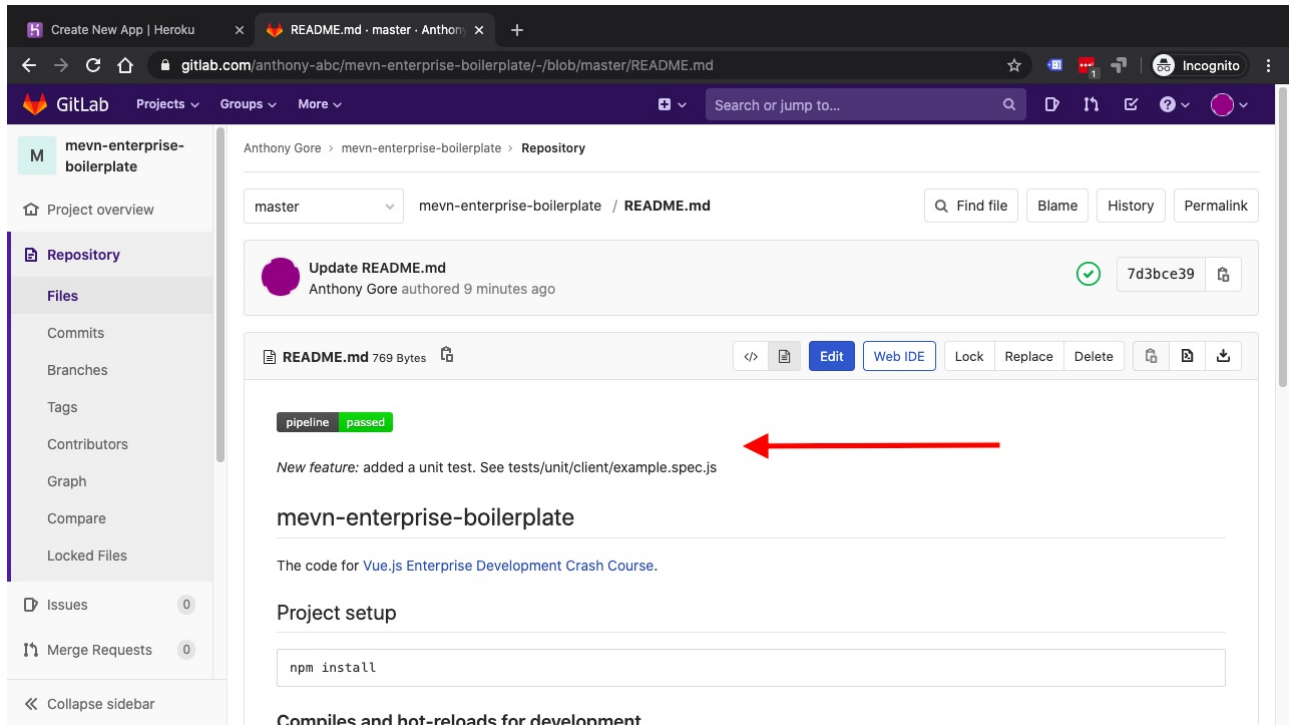
Deploy tokens allow access to your repository and registry images.

Deploy Keys

Back in GitLab, go to your fork and then go to *Settings > CI > Variables*, and set a new variable `HEROKU_APP_NAME`. The value for this will be whatever you named your free Heroku app.

Then set a second environment variable, `HEROKU_API_KEY`. The value for this can be found in your Heroku account settings.

7. Explain change in the README file



With that done, the build should now be passing. Note, though, that Heroku app won't actually be working just yet. You'll first need to set environment variables, and create a database in Heroku.

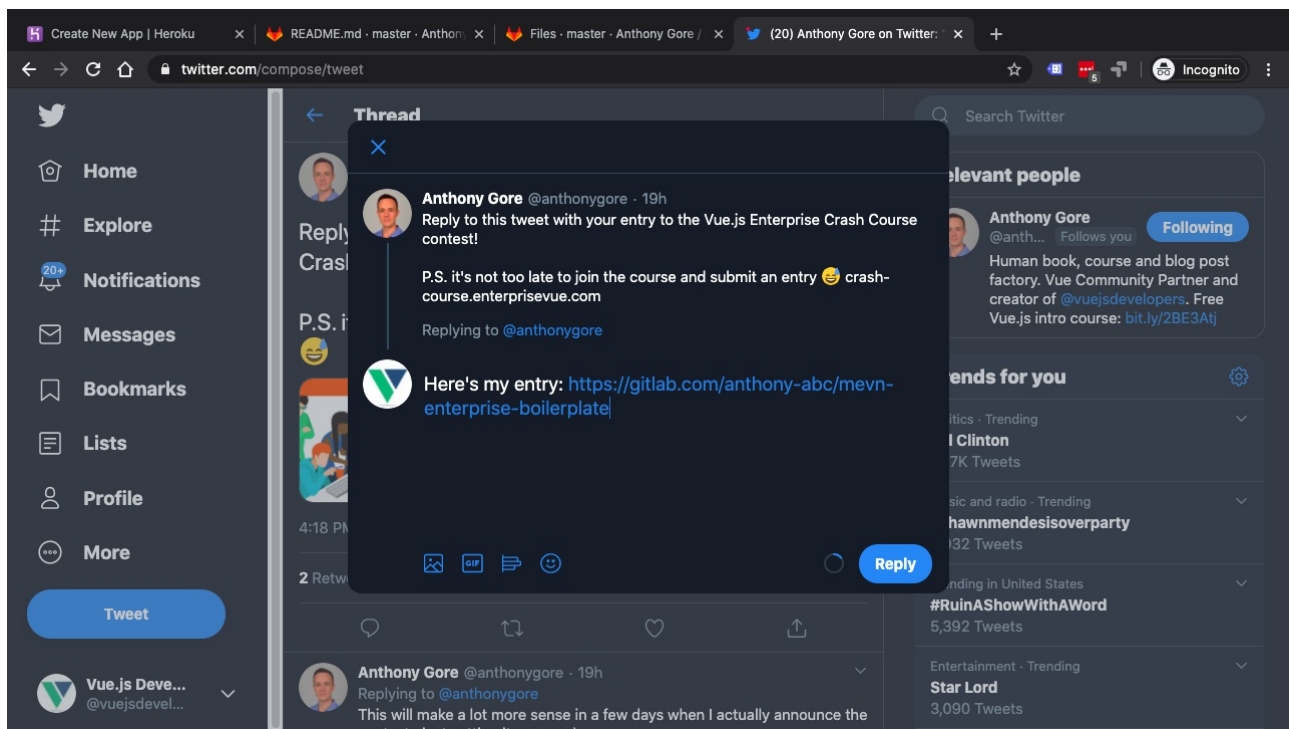
I recommend you figure out how to do this, however, it's not a requirement of the contest so I won't cover that here.

What you need to do now is add a new feature to your fork. It can be anything you like that we covered in the course, so be creative.

For example, you could add another unit test or another E2E test. It doesn't have to be complicated, just make sure the build is passing once you finished.

After you've made your change, edit the README file again and make a note of you change so that I can check it.

8. Tweet your entry to me



Now that you're finished, you can submit your entry. To do that, I want you to reply to this tweet with a link to your repo.

<https://twitter.com/anthonygore/status/1252527201246797825>
(<https://twitter.com/anthonygore/status/1252527201246797825>)

How a winner will be chosen

1. Must submit a valid entry (passing pipeline, description of new feature, reply to my tweet with link)
2. Asking questions and helping other people (in GitLab issues, Facebook, or Disqus)

In order to be eligible for the contest, you'll need to:

Firstly, submit a valid entry, which is a fork of the repo with a passing pipeline, a change that you've described in the README, and posted as a reply to my tweet. If you miss any of those things, your entry will be invalid.

Secondly, I want to see that you're contributing to other students in some way. I want to see you answer a question or ask a question in the Facebook group, in the Disqus comments, or in the GitLab issues on the repo.

So try to post a helpful answer to someone else's question, or ask your own question, or even just share a useful resource or something that you've learned.

It is a contest, but I'm judging it more on how you participate rather than your ability to write the code.

So try to share or be helpful in the comments, and if I see that you're doing that, and you've got a valid entry, then you'll be more likely to be chosen as the winner.

Ready, set...go!

Anything else you need for the contest I will link below this video.

I'll look forward to seeing your entry. Good luck!