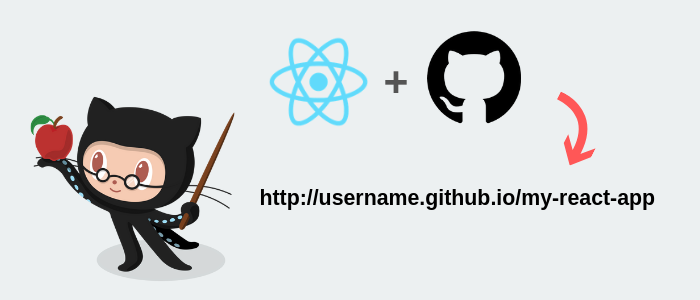
# Lab 06 – Static Hosting with GitHub Pages



We have our SPA in a state where we can run builds and successfully compile the application that we need deployed. Now it’s time to start the first part of our pipeline – the continuous integration process. We’ll be learning about branch specific deploys which follows GitHub’s convention using the **gh-pages** branch.

When we hooked up Travis and tested the integration with our first build, the Travis CI configuration file created the branch for us. Before moving forward, let’s take a closer look at the config file and see what exactly Travis is doing for us.

Text

Description automatically generated

It doesn’t matter if you’re working with a CI/CD platform like Jenkins, GitLab, Circle CI or Travis. There’ll always be a configuration file to instruct the platform how to handle integrations. With Travis, as we’ve learned in previous labs, this comes in the form of a .travis.yml config file.

The config file is rather declarative but for the sake of clarity, let’s go through each section together.

On **line 1**, we’re defining the language of the runtime we want used which in our case is Node/JavaScript. **Line 2** is defining the version of Node which in our case, is the latest stable version.

**Lastly**, is our deploy section. Here is a breakdown of each key, please pay close attention to the “skip\_cleanup” key as this can be somewhat of a Travis gotcha and a need to know when using the platform.

* **provider** - This let’s Travis know where we want our project deployed. In our case, it’s GitHub Pages thus the “pages” value
* **skip\_cleanup: true -** **Prevents the reset of the current working directory and the deletion of all changes made during the build.**
* **github\_token**: this key holds an environment variable containing the token we generated in our GitHub account
* **local\_dir** - Once the build completes this will be the physical path where the output artifact goes. In our case, it is the build directory we created earlier.
* **on**: branch: master/main - allows us to specify the branch(es) we want to deploy. Any branch outside this config will build but won’t be deployed.

Now that we have a defined explanation for what’s inside the config file, let’s start extending it for GitHub Page deployments by adding some new keys. Below you’ll find a copy of the newly extended config file followed by an image of what it should look like in your editor and an explanation for each addition.

`

**language: node\_js**

**node\_js:**

**- "stable"**

**branches:**

**only:**

**- master**

**cache:**

**directories:**

**- node\_modules**

**install:**

**- npm install**

**- npm install -g yarn**

**script:**

**#- yarn lint**

**- yarn build**

**- mv build/index.html build/404.html**

**deploy:**

**provider: pages**

**skip\_cleanup: true**

**github\_token: $github\_token**

**local\_dir: build**

**on:**

**branch: master**

`

Text

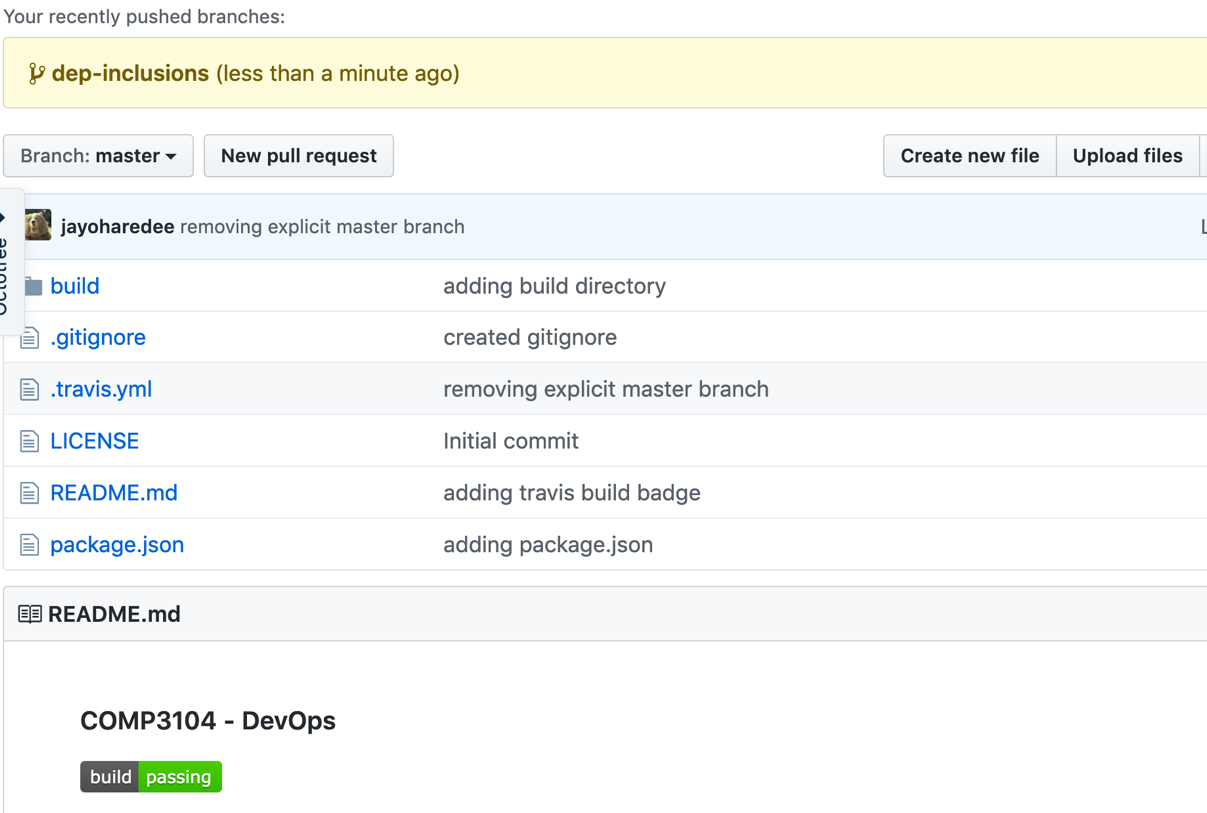
Description automatically generated

Let’s look at the additions to our config file which is only lines 5 - 19.

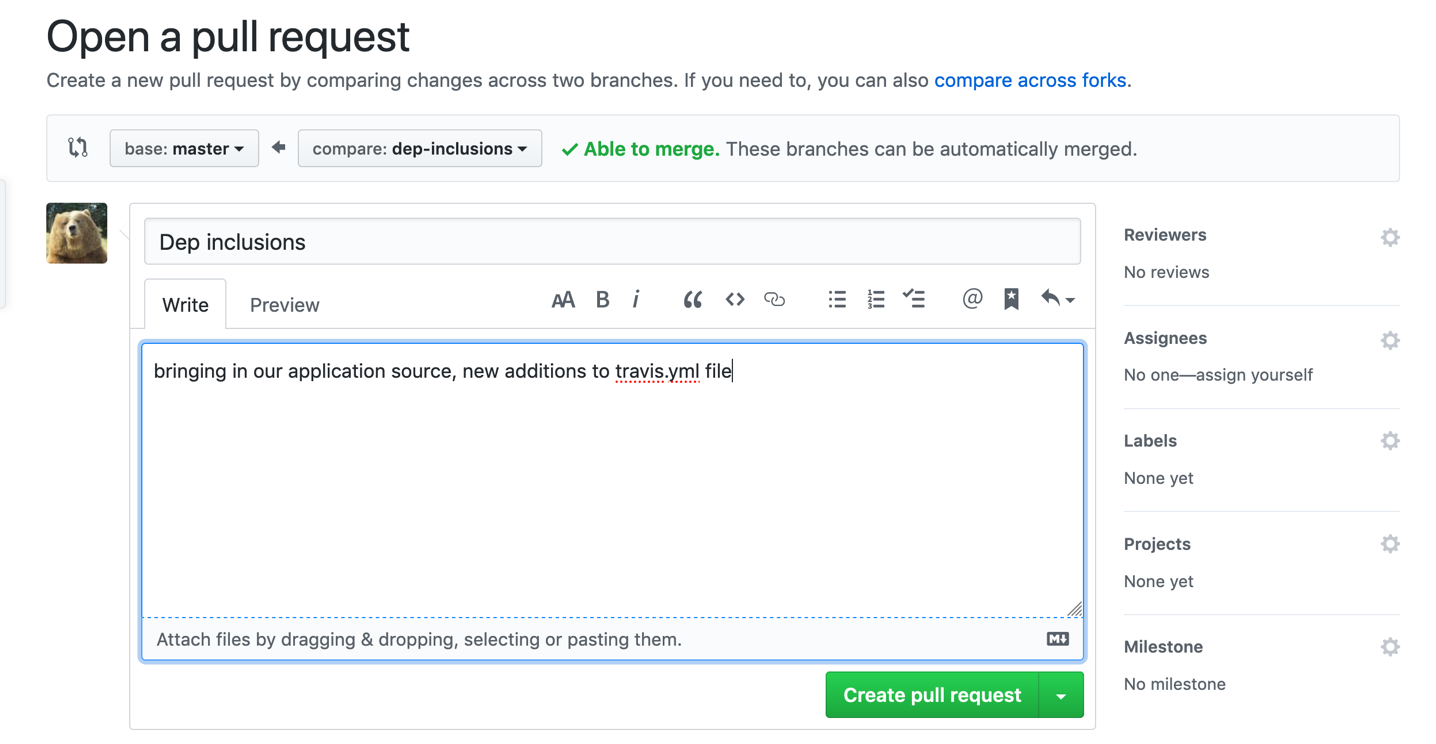
* Lines 5 – 7, signifies the master branch as the only branch to deploy to when a build is triggered through a merge.
* Lines 9 - 11, here we’re caching the node dependency folder “node\_modules” which will help us achieve quicker builds.
* Lines 13 & 14 allow for us to install a new dependency called Yarn. Yarn is a package manage like npm but has some advantages to its use with production applications. We'll get into this a bit more later on but **please be aware that the install key on line 13 allows proceeding lines prefixed with a “-” to run a command to install node dependencies.**
* Lines 16 – 19 allow for more command line execution but this time we’re performing executions of scripts with yarn. Line 17 is commented out for now but we’ll be integrating a linter at a later time. Please note that comments in YML are done with the “#” key. We are building the application using yarn on line 18, just as we would be if we were to perform an “npm run build” command execution

After making these changes to our Travis config file, let’s send the file upstream and make our first Pull Request!

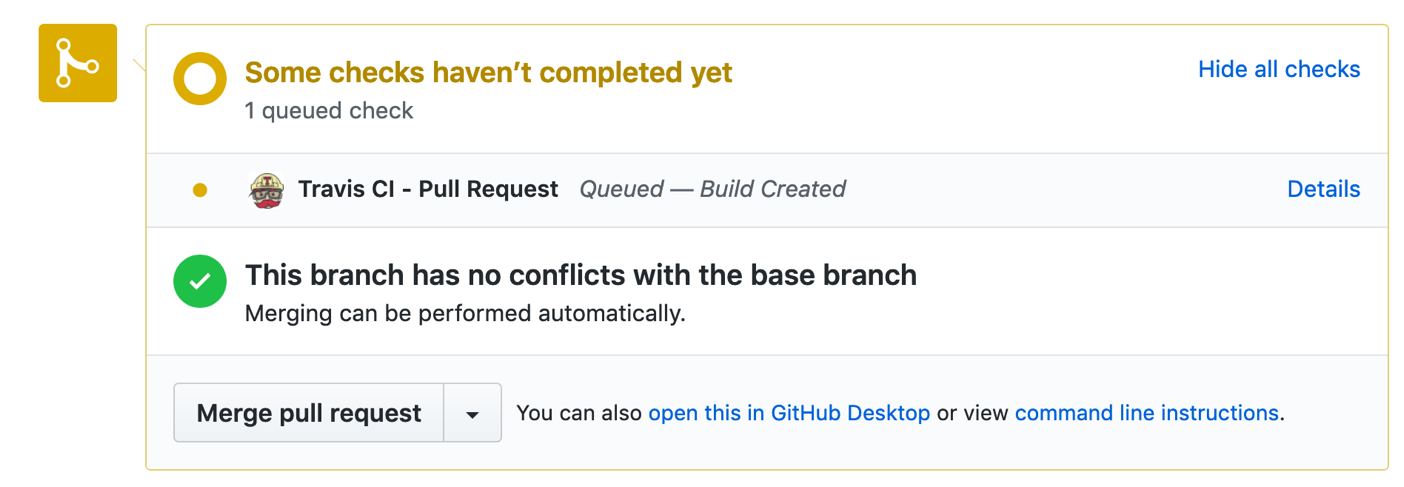
Once we’ve pushed the change upstream, we should see the following in our remote repository on GitHub:



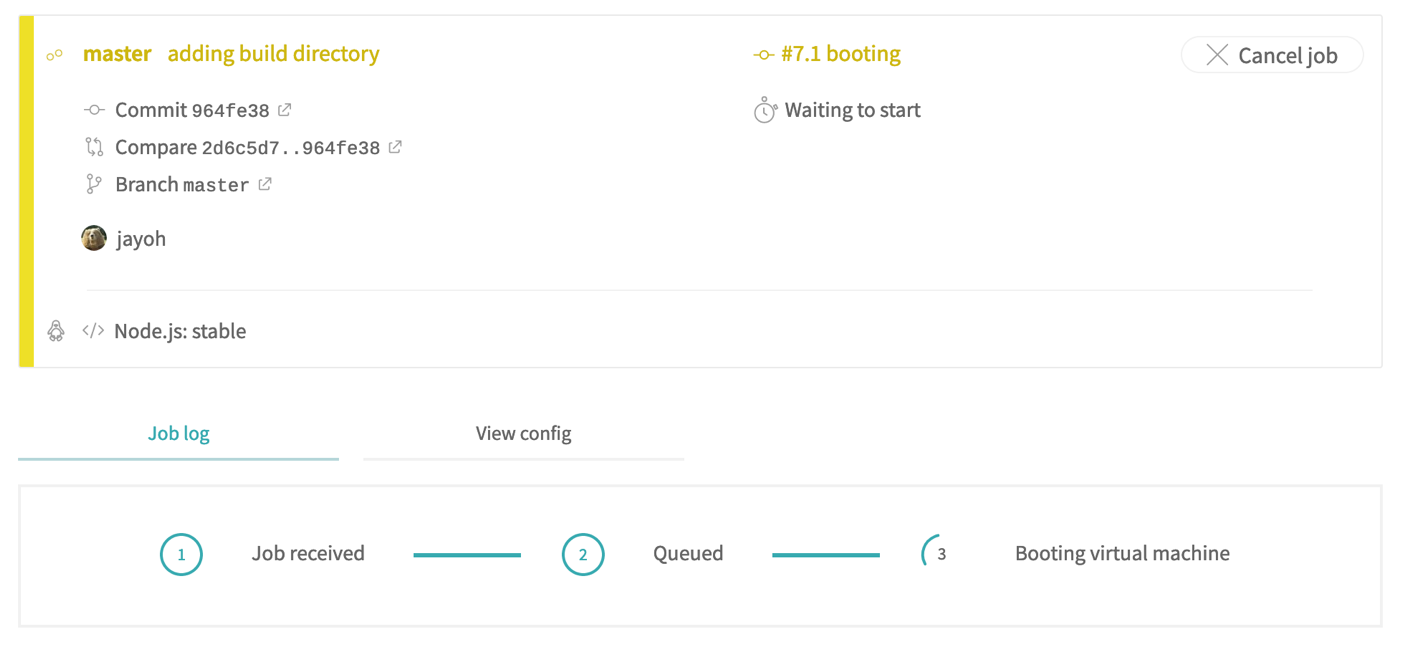
We can go ahead and click on the ‘New Pull Request’ button. From there we’ll be brought to the following page:



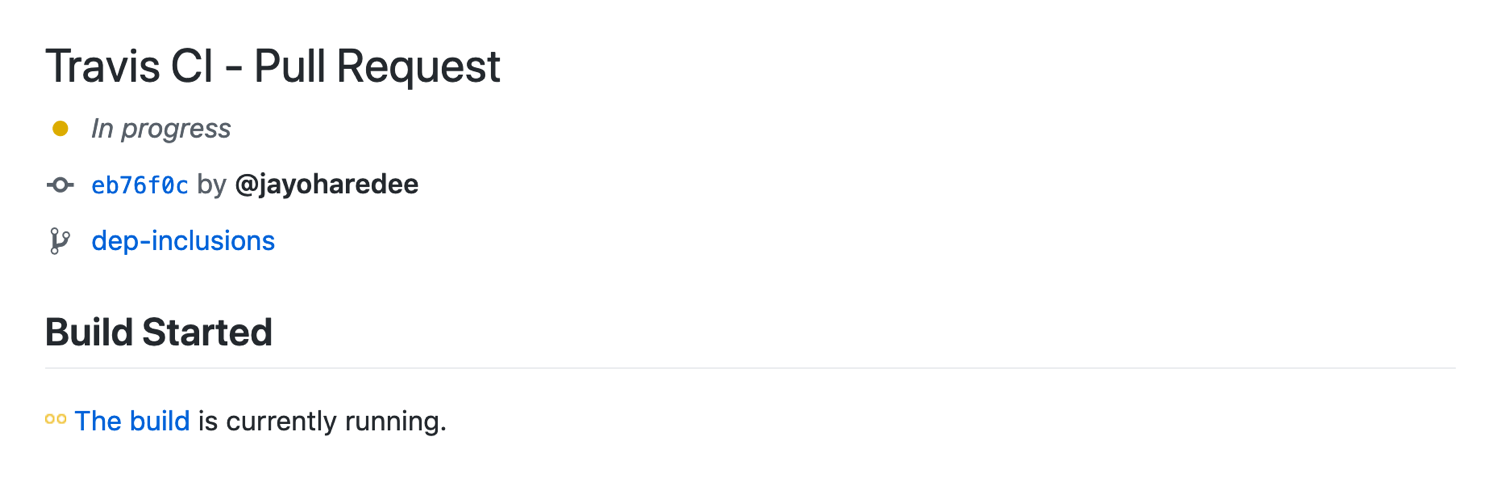
Pictured above is the first part of a process to create a Pull Request on GitHub. It’s best practice to include a detailed description of what the PR contains. Since we’ve only produced changes to our Travis config and also bringing in some application source code, there isn’t much to detail.

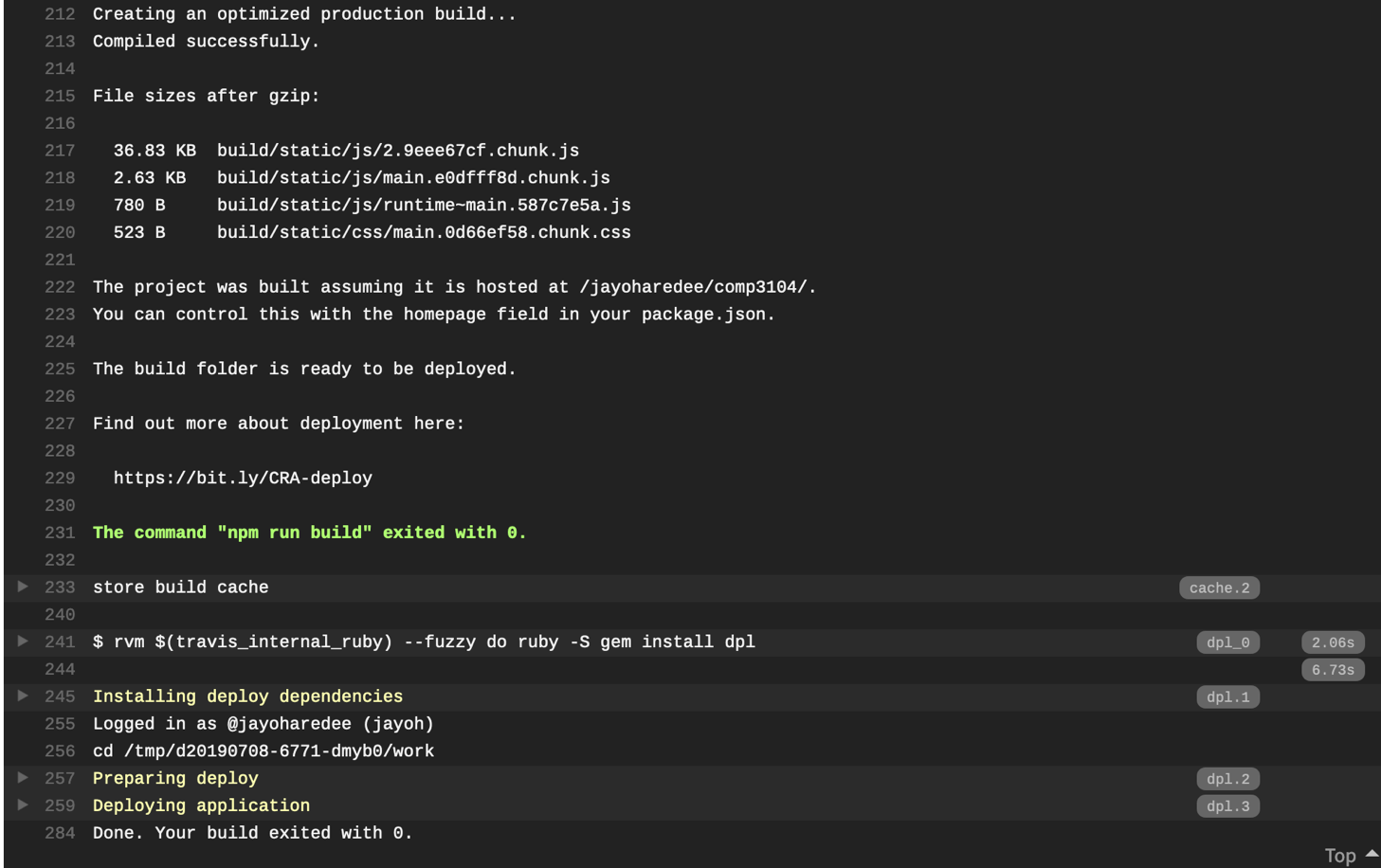


Once the PR is created, we then see that Travis queues up a build and it will also reflect if we take a look in our Travis CI web admin:



These stats will also be available to us in GitHub after approving the pull request.

If we take a look at our build log in Travis we should see something similar to what we’ve seen before when building locally:



npm install

- npm install -g yarn