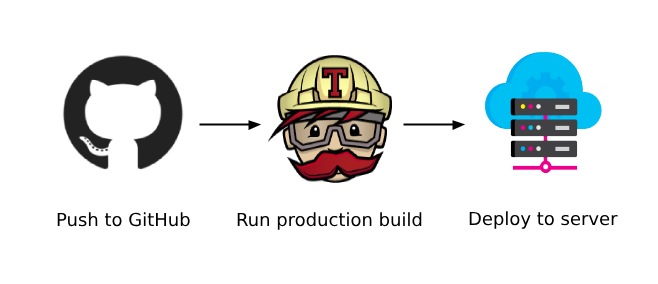
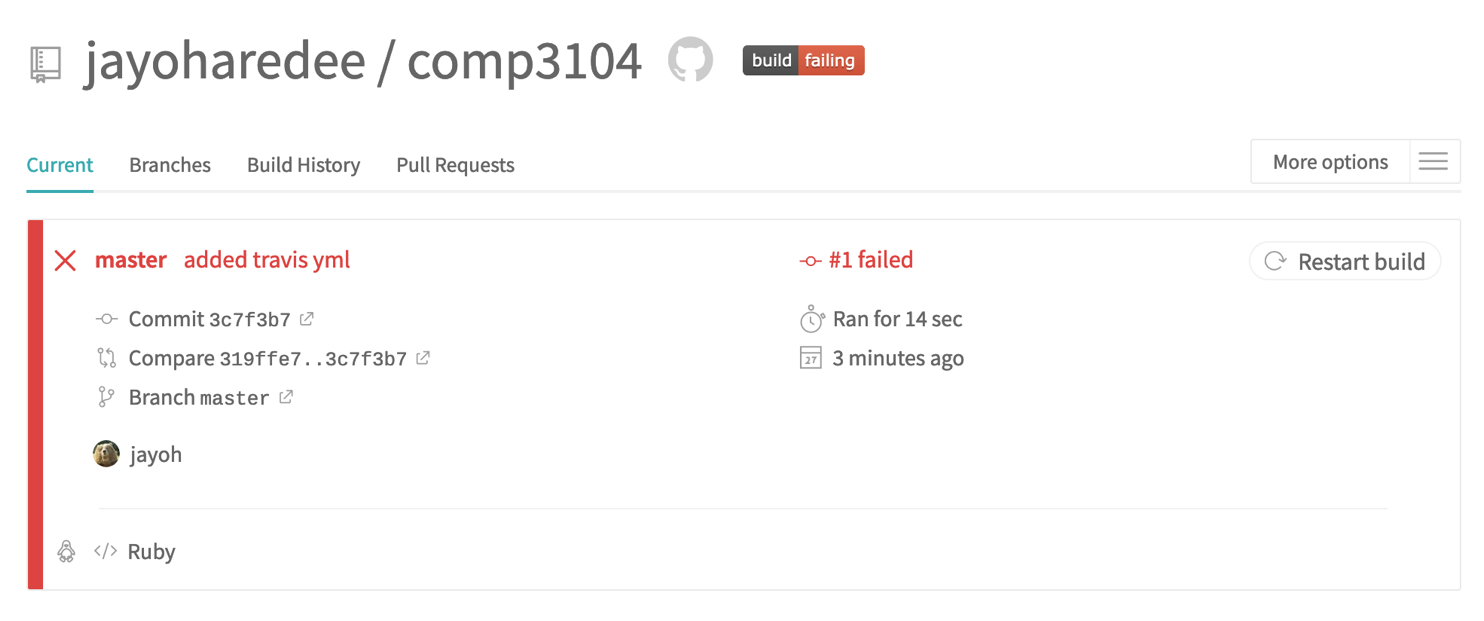
# Lab 04 - Configuring Travis



# In last week's exercise we signed up for a Travis account using GitHub for the account creation process. We also activated our comp3104 repository with Travis by toggling the green button. If you have not yet done this, please take care of it in this lab class. If you require any assistance, please let me know.

If you’ve been following along, you should see an output like mine in your Travis CI account:



This is an indication that we have a failing build in case the UI elements have not made the status apparent. The reason for our failed build is due to us asking Travis to watch our repo and it finding an empty configuration file. Let’s talk a little bit about what a Travis config file is, how it works and then work on getting our build to pass.

## Working with Travis

In order for us to work with Travis, two steps must be taken:

1. Telling Travis to watch our repository
2. Adding a **`.travis.yml**` file to our repository

When Travis CI has been activated for a given repository, GitHub will notify Travis whenever new commits are pushed to the master/main branch, or as best practice dictates, when a PR is approved, and code is merged into the master/main branch.

Travis will be acting as our Continuous Integration suite and perform any steps we instruct it to through the configuration file in our repository - `.travis.yml`. It can complete anything from running tests, building the application or running deployment scripts.

**We know that Continuous Integration is the act of committing changes to the source code and with each commit made, we trigger a build of our application**. This practice allows teams to detect problems early and also include other sequences to take place like a static code analysis or the execution of a test suite.

Let’s test this theory out by adding a change to our Travis config file and pushing the change to our master/main branch. **Please use the minimal configuration found below and add the following to your .travis.yml file in your comp3104 project directory.**

`

**language: node\_js**

**node\_js:**

**- "stable"**

**deploy:**

**provider: pages**

**skip\_cleanup: true**

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

**local\_dir: build**

**on:**

**branch: main**

`

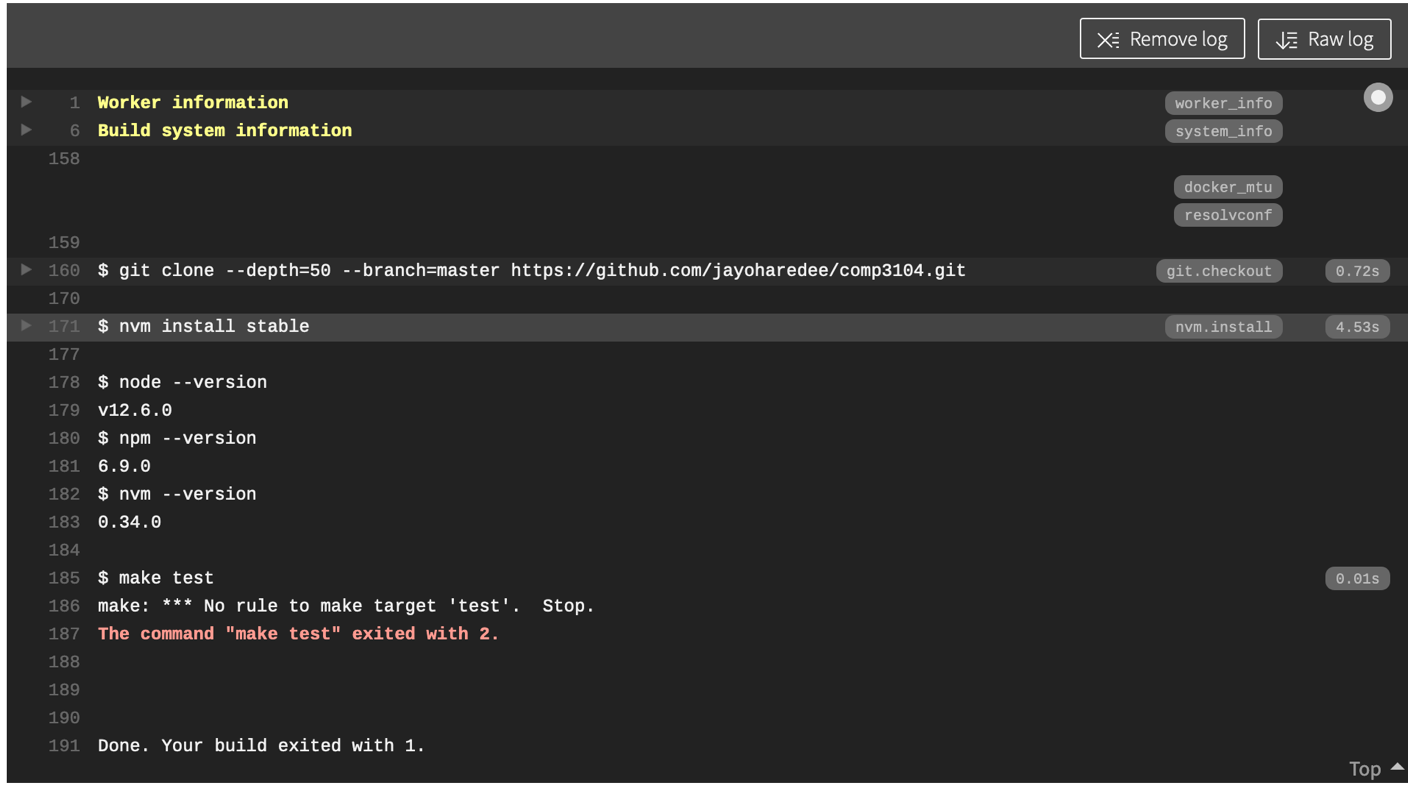
The above text is only intended for ease of access to copy and paste but please be aware of any spacing as we are dealing with YML format. YML stands for “Yet Another Markup Language” for more information on this standard, please reference the labs supplementary material section. One thing to know is that it relies on block scoping similar to Python and relies on nesting of nodes so please pay attention to the spacing found in the image below.

A screenshot of a computer

Description automatically generated with medium confidence

After committing the configuration change and pushing to our master/main branch, we should run into a couple more failures, but let’s work through them together.

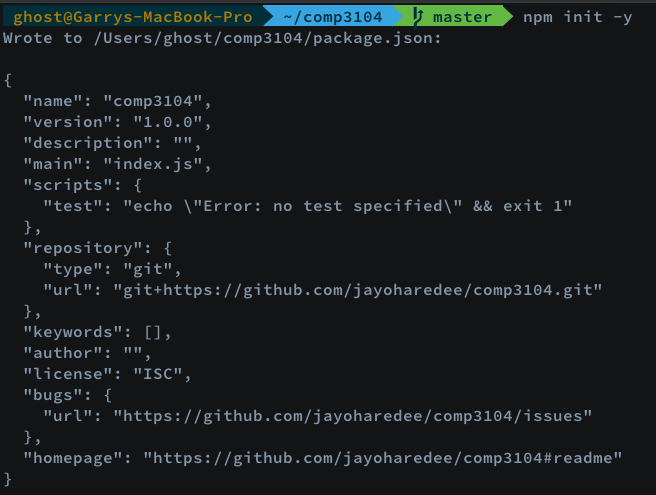
If we take a look at the job that was triggered after pushing our recent change in our `.travis.yml` config file, we should the following failure:

The exact error thrown was due to Travis expecting a test case, but we did not provide one, thus the job failed when no test was found. Let’s use npm to initialize our comp3104 project with Node. Doing so will create a package.json file in our project directory which will allow us to write both test and build scripts.

Head over to your terminal and please ensure you are in your project directory. Once you’re in the comp3104 project, you will want to enter the following command:

`**npm init -y**`

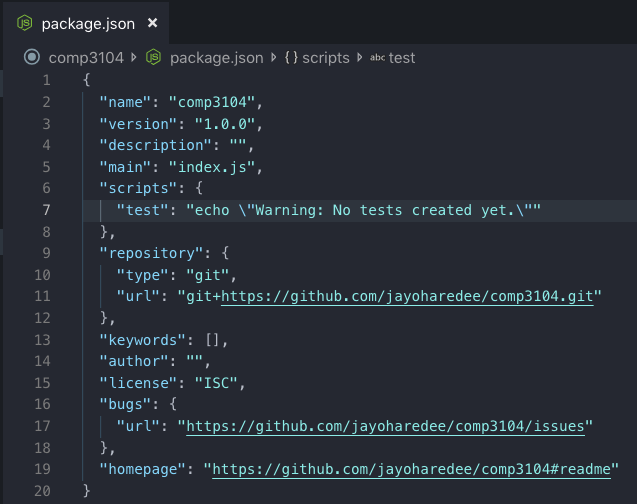
The resulting output should produce something like what’s found in my terminal after executing the above command.



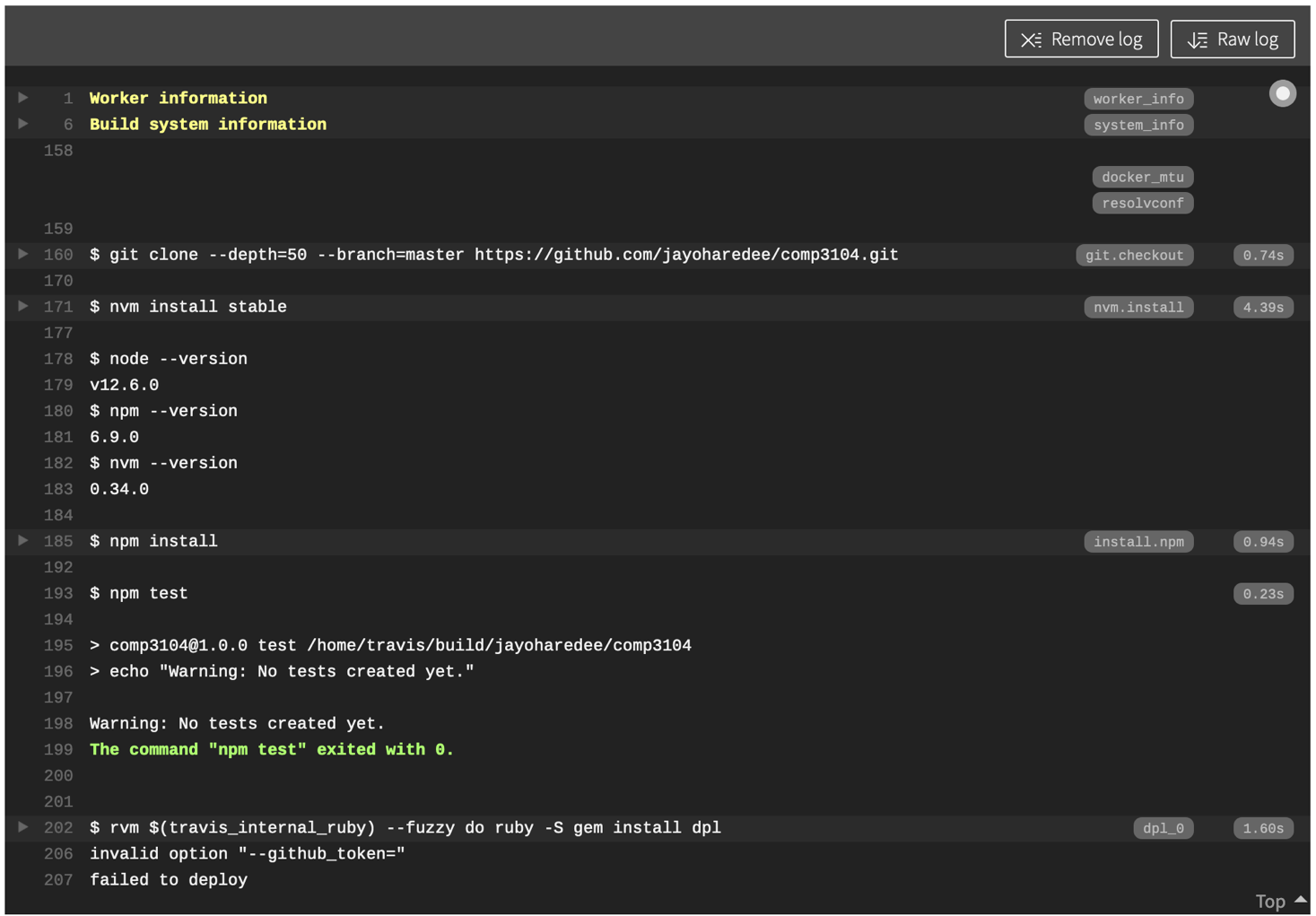
It’s just a little bit of JSON, something you may have seen before consisting of key/value pairs. Please take a look at the 5th key entitled “scripts”. Notice the object assignment containing a key called “test”. We’re going to want to open the newly generated package.json and replace the value for “test” with the following:

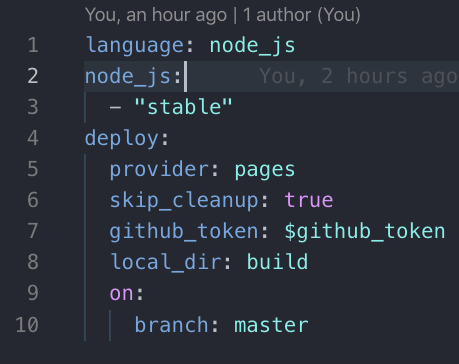
**`"echo \"Warning: No tests created yet.\""`**

I’m using VS Code for an editor now, for yourself, please use whatever your most comfortable with to edit the file. After replacing the value for the “test” key, my JSON now looks like this:

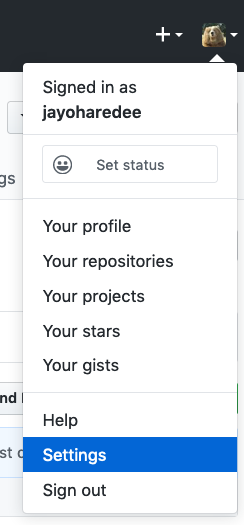


Please note line 7, I am simply echoing out the fact that we have not yet created any test cases. This should now echo out in our Travis log when we commit this new change to our master/main branch in the comp3104 repo. Let’s commit this change and push it up to our master/main branch.

Alright! Seeing some green with an exit of 0 is good but we still have one last error to deal with. Notice how it mentions something about an invalid option for ”**--github\_token=”** ? Let’s take care of this by going back to our travis config file for a moment.



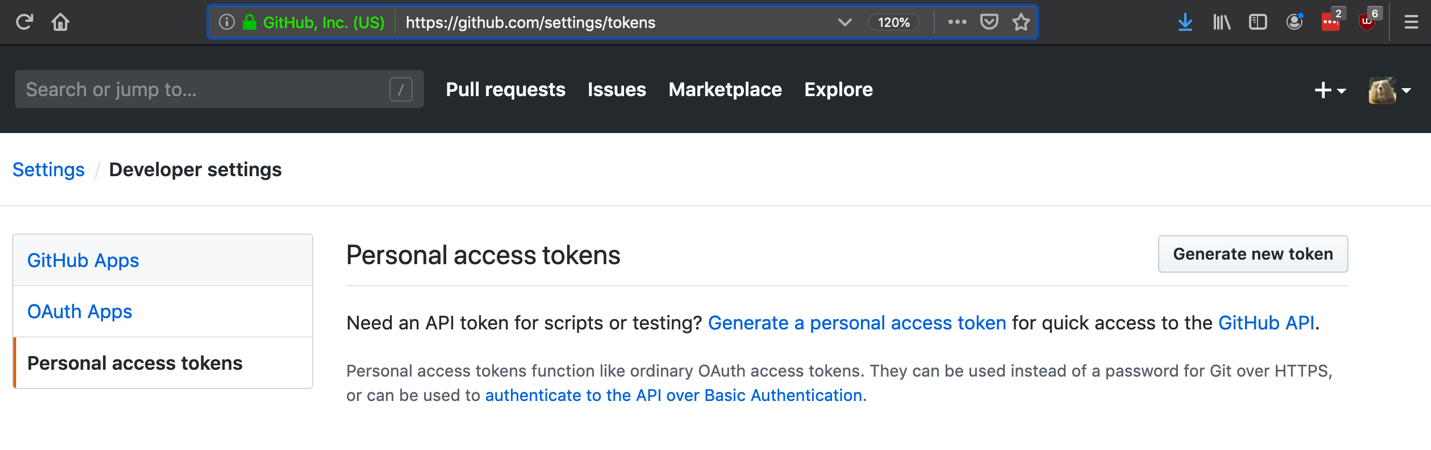
Looking at line 7, we can see a node called `github\_token:` with a value of `$github\_token`. This value is intended to be an environment variable containing a personal access token which GitHub will provide us for the comp3104 repository. This token grants Travis permission to interact with our repository. Head to the same settings area we used when assigning our public SSH key.

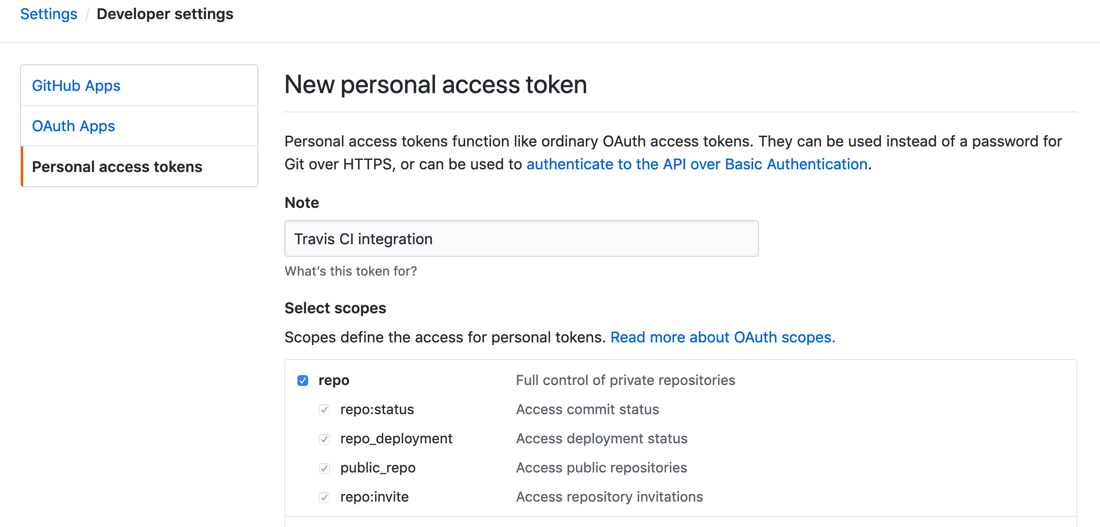


After accessing the settings area, you should see an item in the side menu that says ‘Developer settings. We need to access the Developer Settings followed by another side menu navigation click on ‘Personal access tokens’. If you’re logged in, you should be able to access this page with the following URL:

<https://github.com/settings/tokens>

Once on the personal access tokens page, we’re going to need to generate a new token by clicking the grey button titled “Generate new token”. See the below image for more details.

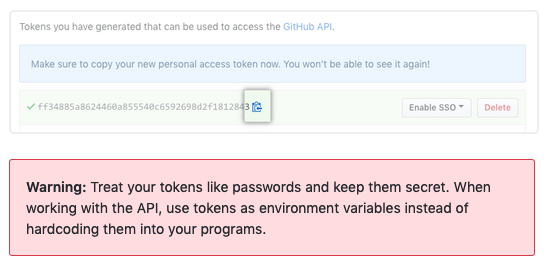
You’ll then want to select the repo scope to grant permissions for use.



Now it’s time to generate the token by clicking the big green button below

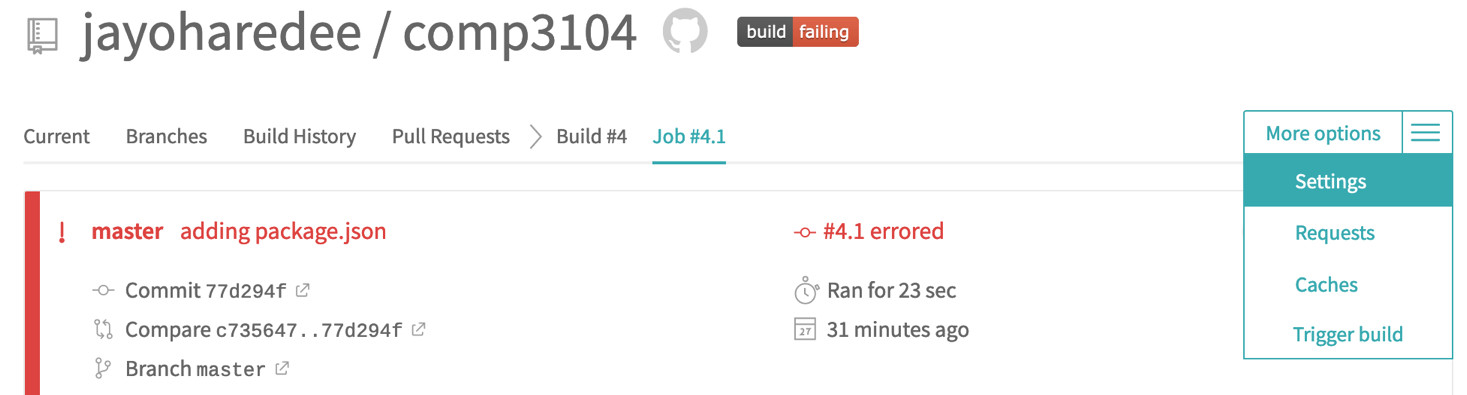


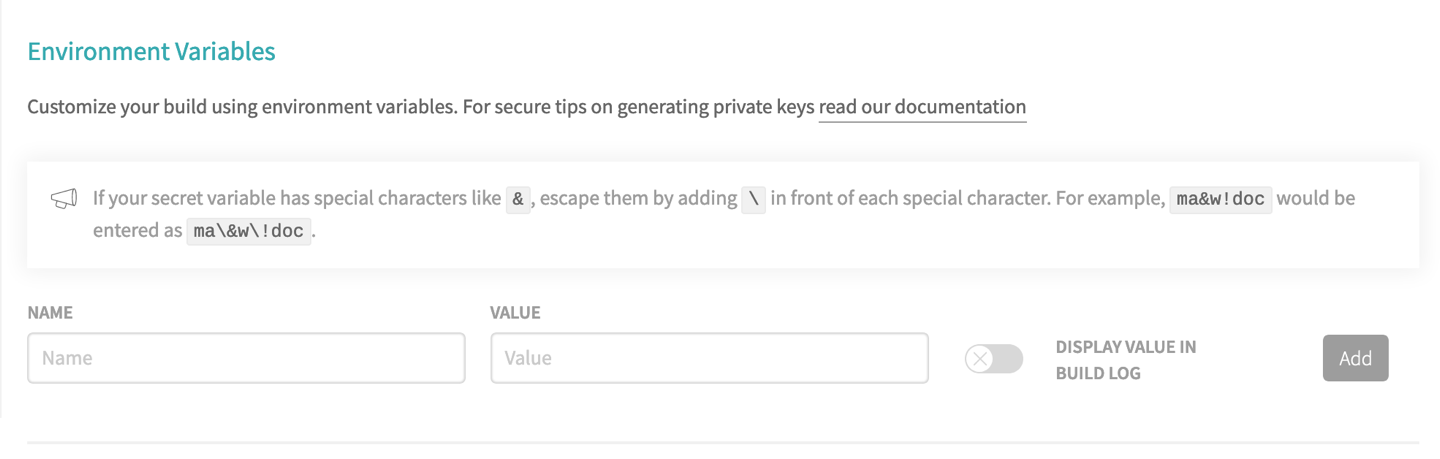
After generating the token, we’ll want to keep this private so don’t share your token with others. Click the clipboard as demonstrated in the below image.

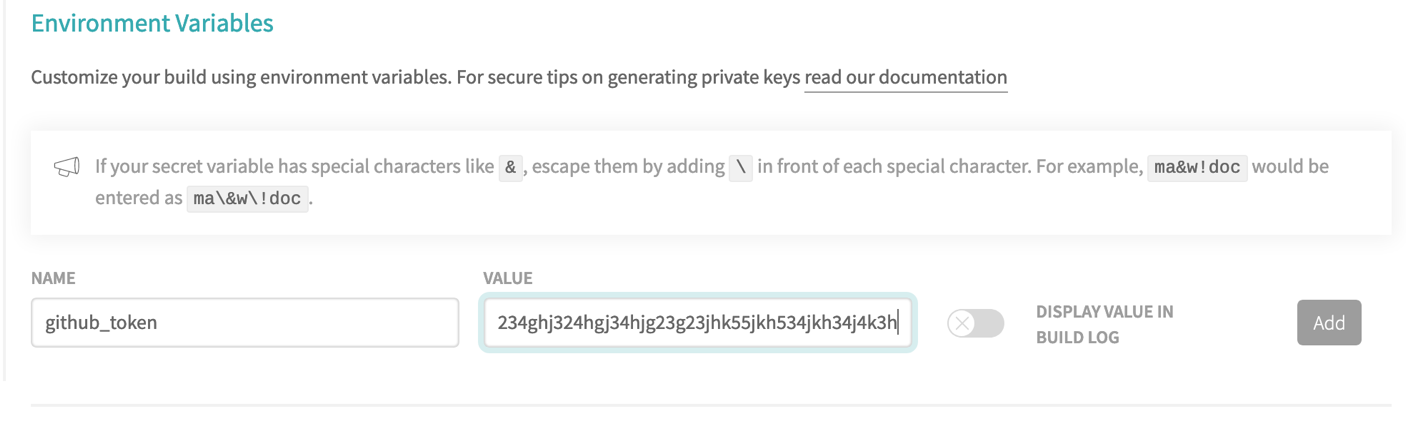


Once we have the copied token, we’ll want to head back over to our Travis console and find our settings page to create the $github\_token environment variable. I’ve gone ahead and documented the process for this so please follow the images below.

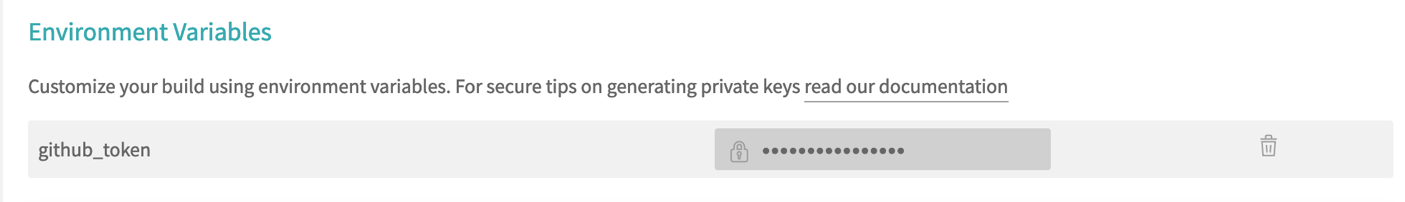
First step; head back to the Travis CI console where our build log was produced and look in the top right corner for a menu that says ‘More Options’ and click on the ‘Settings’ link in the drop down menu.

After being presented with the Settings page, if you scroll down about half way you should be presented with a section entitled ”Environment Variables”. Here we’ll be entering our key which is github\_token and our value is the copied token from GitHub taken from our previous step.

Here’s what mine looks like after filling it out. Ensure that the toggle is not active just like mine is below:



We should now have a newly created environment variable that we can use in our `.travis.yml` config file. After clicking the “Add” button, you should be presented with something like this:

 Now, to test the above out, we’re going to create a new folder in our project directory called build. To do this, we’ll be on the command line and in our project directory using the `mkdir` command we used in our first lab.

Let’s hop over to our terminal and enter the following commands to create a folder along with an index.html file.

**`mkdir -p build`**

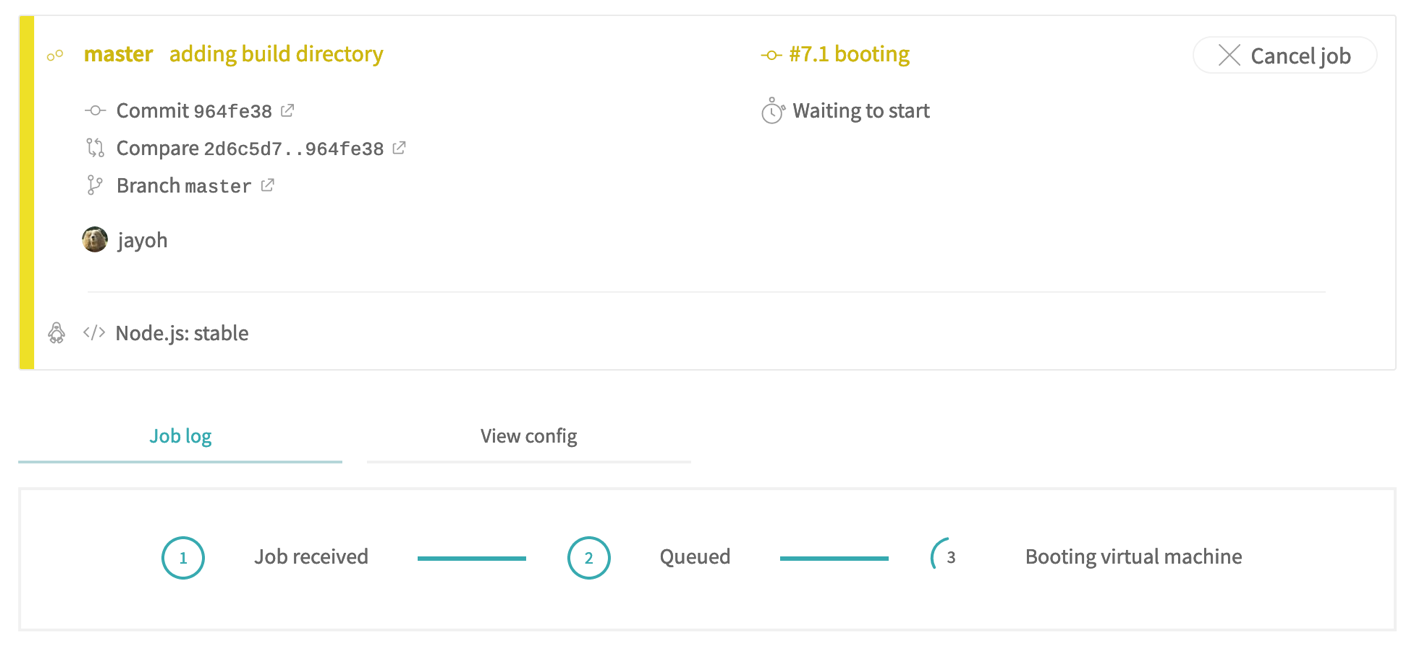
**`cd build`**

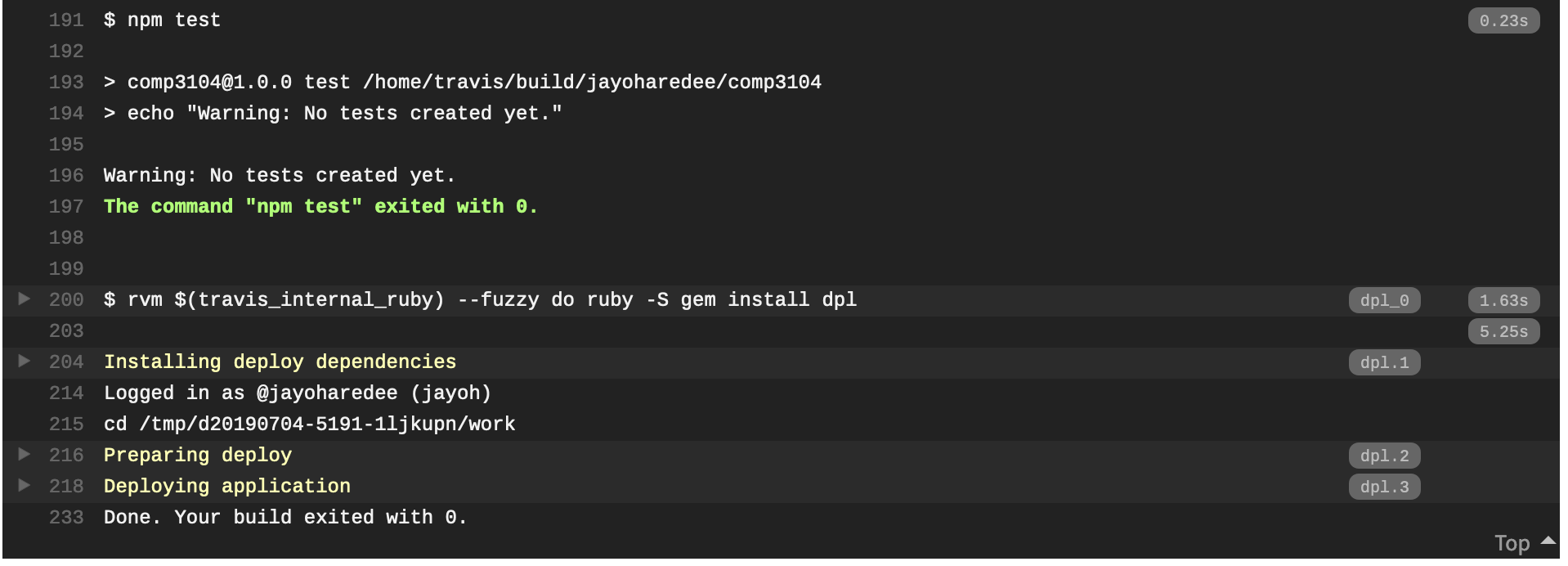
**`touch index.html`**

Let’s now push this newly created directory and file upstream and see what happens.

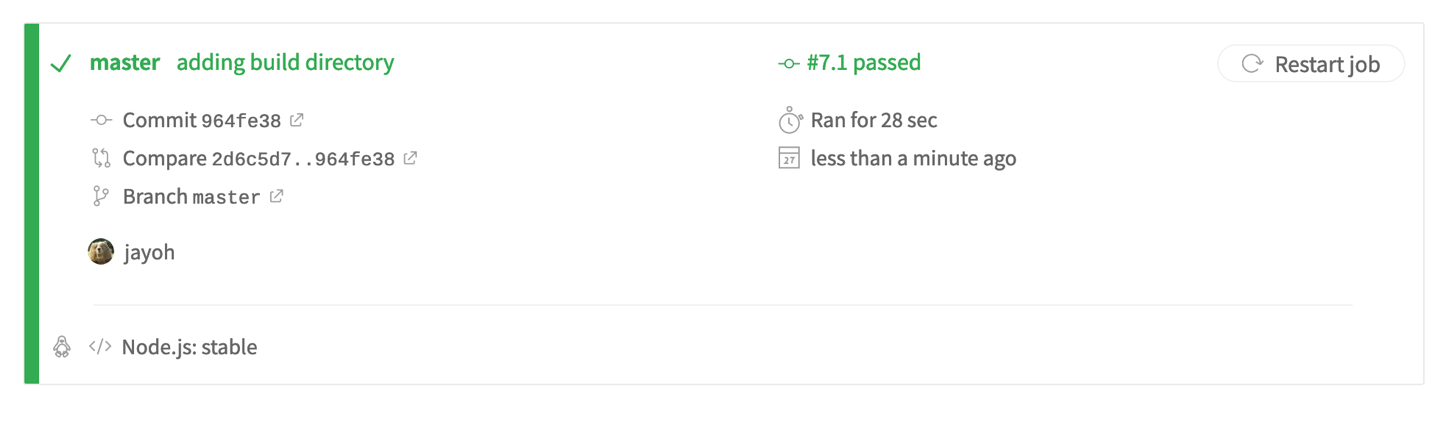
If everything went as planned, here is the process we should have seen take place:

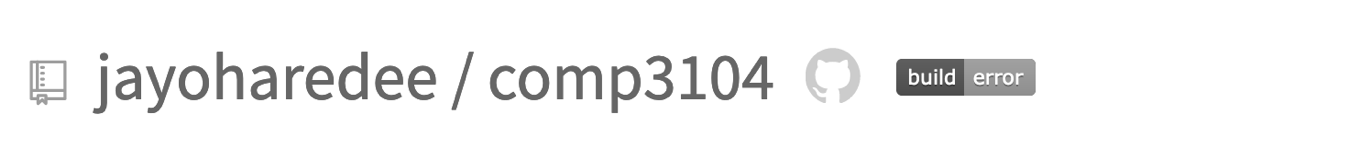
First, our CI suite boots up a container to run our process in:

After this completes, we should start seeing some output in our log, the end of which looks like this:

Alright, so judging by the output above it looks like our application was successfully deployed as there is no errors. Please note on line 214 above that Travis actually performs a login process with the token we provided earlier. We can see my username displayed that’s associated with the token.

At the beginning of this lab, we’ve seen what a failed build looks like. Hopefully now, your build is passing too:

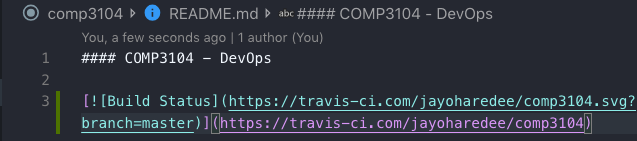
To ensure that everything is truly copacetic, we’re going to perform one last step. We’re going to add a badge to our README.md file to indicate the status of our latest build.



Looking at the image above, we can see a badge that says, ‘build error’. Let’s click on this badge in order to get the following pop-up:



You’ll want to select the second select element and choose the MARKDOWN option just as I’ve done above. Take the value from the text field and copy that bit of markdown. We’ll then want to go and paste it into our README.md file as shown below.



Let’s push this upstream and see what change has been made to our projects readme.

The resulting push should now display the following in our projects readme file:

