Figure 10-2. GitHub pull request showing automated test results from CircleCI.

Merge and Release

Your team members should review your code changes, looking for potential bugs, enforcing coding guidelines (more on this later in the chapter), checking that the existing tests passed, and ensuring that you've added tests for any new behavior. If everything looks good, your code can be merged into the `main` branch.

The next step is to release the code. If you're using immutable infrastructure practices (as discussed in "Server Templating Tools"), releasing application code means packaging that code into a new, immutable, versioned artifact. Depending on how you want to package and deploy your application, the artifact can be a new Docker image, a new virtual machine image (e.g., new AMI), a new *.jar* file, a new *.tar* file, etc. Whatever format you pick, make sure the artifact is immutable (i.e., you never change it) and that it has a unique version number (so you can distinguish this artifact from all of the others).

For example, if you are packaging your application using Docker, you can store the version number in a Docker tag. You could use the ID of the commit (the sha1 hash) as the tag so that you can map the Docker image you're deploying back to the exact code it contains:

```
$ commit_id=$(git rev-parse HEAD)
$ docker build -t brikis98/ruby-web-server:$commit_id .
```

The preceding code will build a new Docker image called `brikis98/ruby-web-server` and tag it with the ID of the most recent commit, which will look something like `92e3c6380ba6d1e8c9134452ab6e26154e6ad849`. Later on, if you're debugging an issue in a Docker image, you can see the exact code it contains by checking out the commit ID the Docker image has as a tag:

```
$ git checkout 92e3c6380ba6d1e8c9134452ab6e26154e6ad849
HEAD is now at 92e3c63 Updated Hello, World text
```