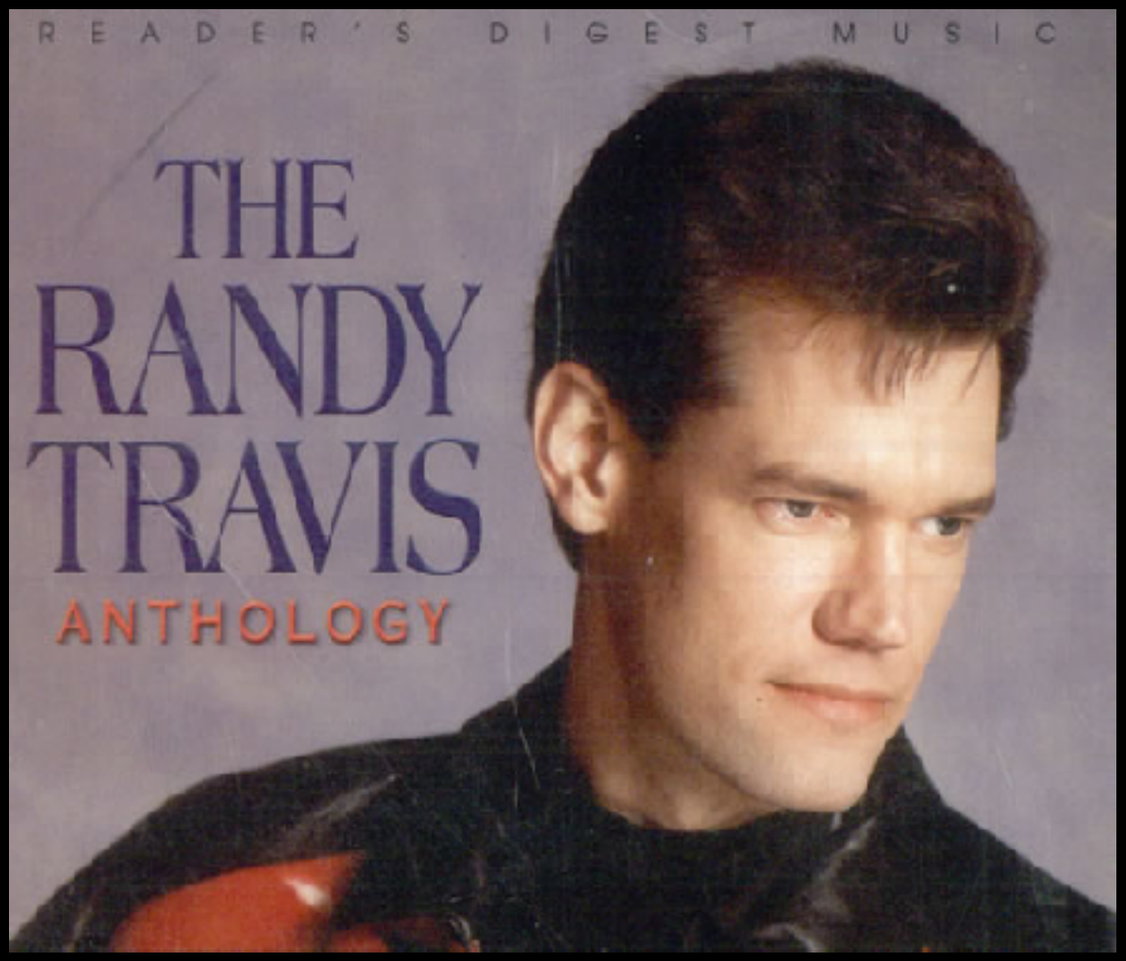
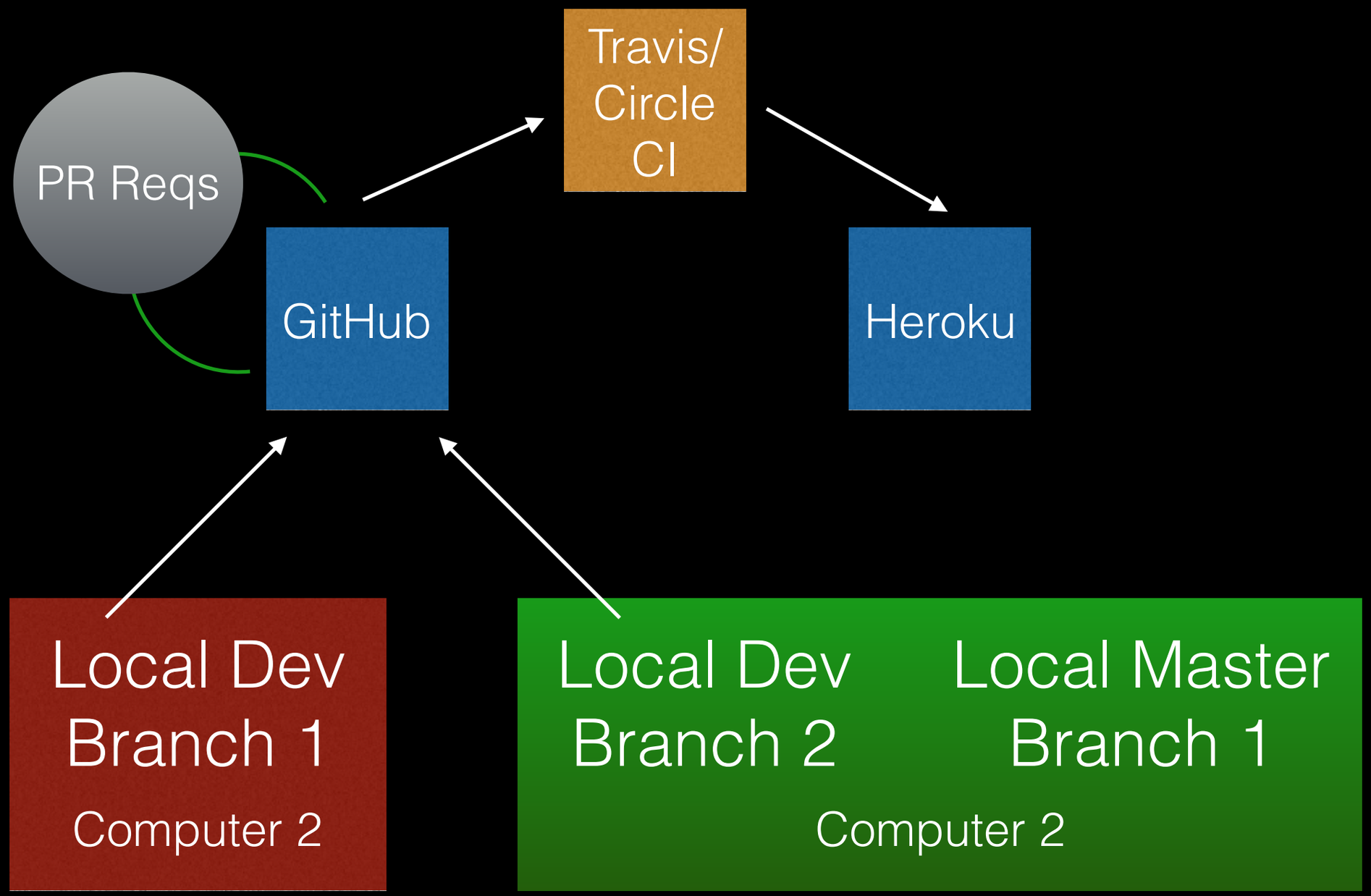


Travis CI

AKA Travis Continuous Integration

AKA How to make testing and deployment to Heroku easier

Wat CI Duz 4U


PR Reqs
Travis/
Circle
CI
GitHub
Heroku
Local Dev
Branch 1
Computer 2
Local Dev
Branch 2
Local Master
Branch 1
Computer 2

Why Use CI?

- What is continuous integration (CI)?
  - “[T]he practice of merging all developer working copies to a shared mainline several times a day”
  - So, as teams push branches to GitHub, they merge to master frequently.
  - Purpose is to prevent integration issues from divergent code causing merge conflicts
  - Dependent on automated test-driven development for rapid re-verification (automated regression testing)
  - Travis CI extends the concept of continuous integration to continuous deployment as well
  - Not just for testing Ruby projects! Look! https://docs.travis-ci.com/user/language-specific

How to do CI

(https://www.thoughtworks.com/continuous-integration)
(my words in [])

- Maintain a single source repository *[on GitHub]*
- Automate the build *[using a CI software like Travis or Circle CI]*
- Make your build self-testing *[using a CI software like Travis or Circle CI]*
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment *[using a CI software like Travis or Circle CI]*
- Make it easy for anyone to get the latest executable version *[from GitHub]*
- Everyone can see what's happening
- Automate deployment *[using a CI software like Travis or Circle CI]*

How to do CI (like specifically)
(https://www.thoughtworks.com/continuous-integration)
(my words in [])

- Developers check out code into their private workspaces
- When done, commit the changes to the repository
- The CI server monitors the repository and checks out changes when they occur
- The CI server builds the system and runs unit and integration tests
- The CI server releases deployable artifacts for testing
- The CI server assigns a build label to the version of the code it just built
- The CI server informs the team of the successful build
- If the build or tests fail, the CI server alerts the team
- The team fixes the issue at the earliest opportunity
- Continue to continually integrate and test throughout the project

Team Responsibilities
(https://www.thoughtworks.com/continuous-integration)
(my words in [])

- Check in frequently
- Don’t check in broken code
- Don’t check in untested code
- Don’t check in when the build is broken
- Don’t go home after checking in until the system builds

Continuous Deployment

(https://www.thoughtworks.com/continuous-integration)

- Continuous Deployment is closely related to Continuous Integration and refers to the release into production of software that passes the automated tests
- So, for you… CI followed by automatic deployment to Heroic

What Travis CI Does

- Monitors GitHub for public repositories users have flagged for Travis CI builds (private repos too for $$$)

- Creates a temporary Linux environment to run spec tests defined in your project

- If tests pass, then developers can configure Travis CI to optionally deploy to a web-hosting service (e.g. Heroku)

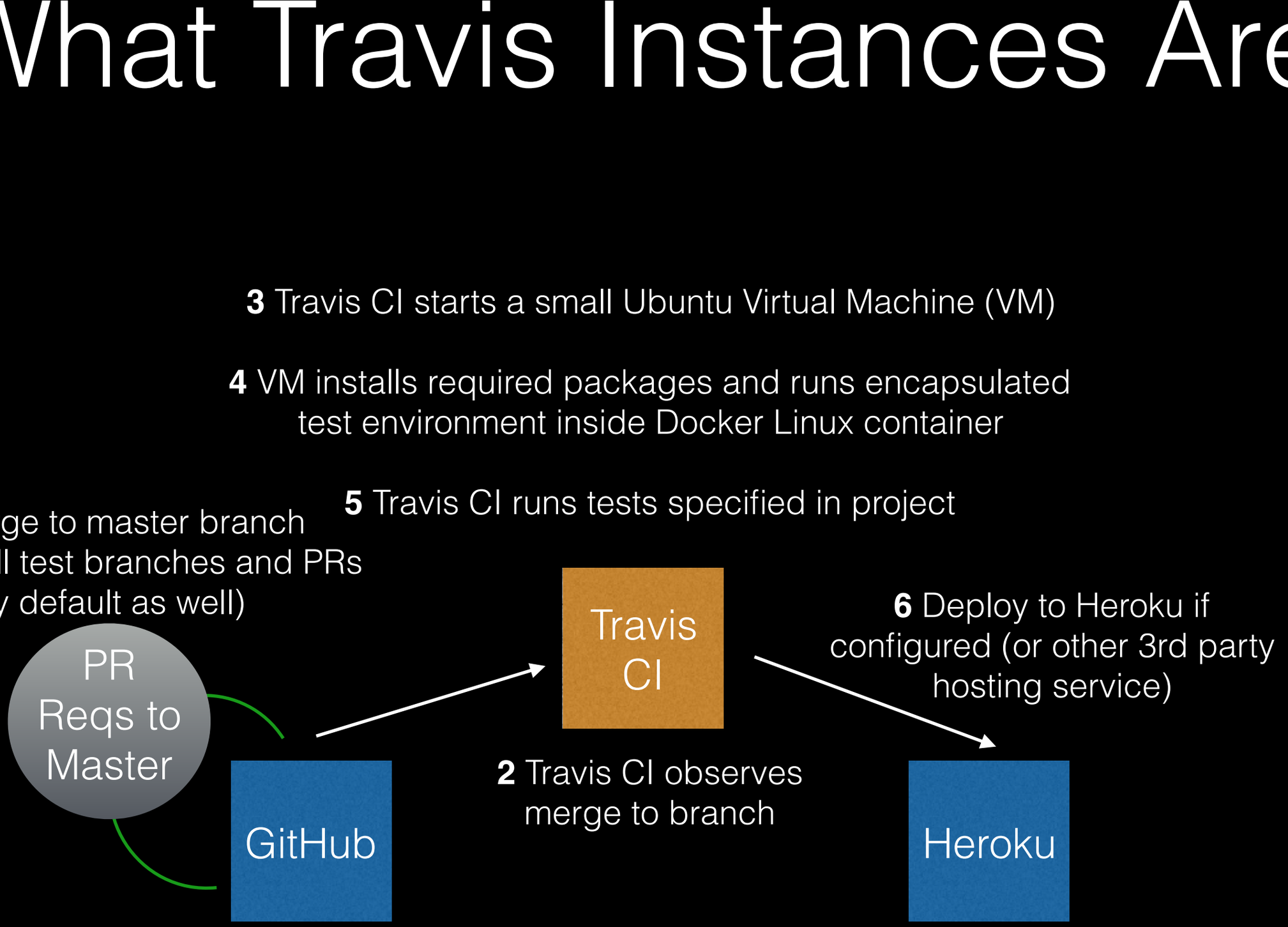
What Travis Instances Are

3 Travis CI starts a small Ubuntu Virtual Machine (VM)

4 VM installs required packages and runs encapsulated test environment inside Docker Linux container

5 Travis CI runs tests specified in project

1 Merge to master branch (Travis will test branches and PRs by default as well)



2 Travis CI observes merge to branch

6 Deploy to Heroku if configured (or other 3rd party hosting service)

How to configure test/deployment in Travis (prior to messing with Travis)

- Set up pubic GitHub repo for your project
- Push your project to GitHub
- Create a Heroku project and deploy to Heroku
- Test your application on Heroku

How to configure test/deployment in Travis (Travis First Steps)

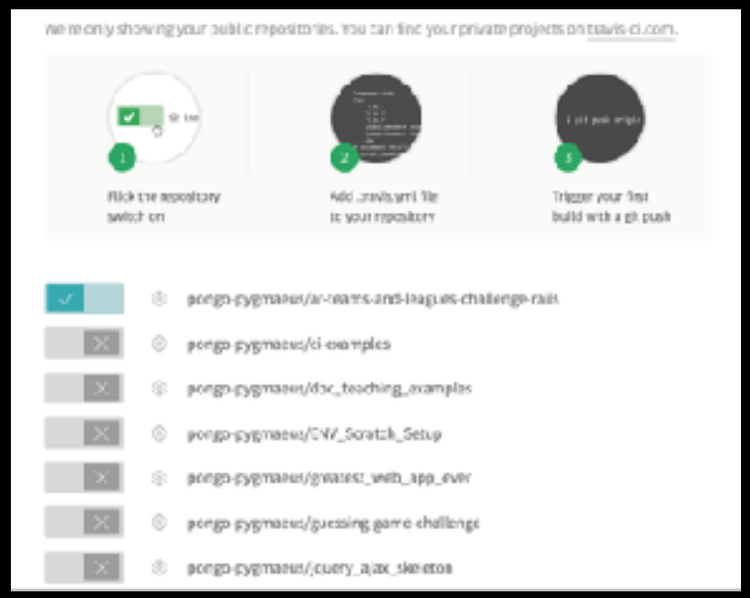
- Visit https://travis-ci.org and click on the “Sign Up” button (you can do private repo stuff from https://travis-ci.com)

- If you’re logged into GitHub in the current browser session, Travis should automatically log in. If not, SORRY.

- Once you’re in, navigate to https://travis-ci.org/profile/[GitHub User Name]

How to configure test/deployment in Travis (Travis First Steps)

- If your GitHub project is public, you should see it in a list like this:



- Select the slider bar on the left to activate your project to integrate with Travis

How to configure test/deployment in Travis (shove Travis in the middle)

- Add a file to the root directory of your project. Call it ***.travis.yml***

- Git commit and watch the magic (fail) on Travis CI

- After your project fails for the first time, add some things to your ***.travis.yml*** file to actually make things work

How to configure test/deployment in Travis (shove Travis in the middle)

- In your project directory (or anywhere on your system run ***gem install travis*** to install the Travis CI CLI on your computer
- Run ***travis encrypt $(heroku auth:token) --add deploy.api_key*** in your project directory to generate the following lines in your ***.travis.yml*** file

```
deploy:
  provider: heroku
  api_key:
    secure: Lxa……
```

- These lines setup Travis to auto deploy to Heroku after completing tests
- Add the following line after ***provider*** to make it abundantly clear what Heroku project you want Travis to deploy to:

```
app: frightening-moonlight-10587
```

How to configure test/deployment in Travis (shove Travis in the middle)

- If you need to run tests that require a database add the following lines to configure the test database to be generated inside the Travis CI VM instance

UNDERSCORES ONLY IN DB NAME

```
services:
- postgresql
- env:
- DB=postgresql
before_script:
- psql -c 'create database test_database_name;' -U postgres
script:
- RAILS_ENV=test bundle exec rake db:migrate --trace
- bundle exec rake db:test:prepare
- bundle exec rake db:seed RAILS_ENV=test
- bundle exec rspec spec/
```

Use PG

Create test DB

Setup and Seed DB

Run tests

How to configure test/deployment in Travis (Importannnnttttt)

- Specify which version(s) of Ruby you want to test under:

```
language: ruby
rvm:
- 2.3.1
```

- Why use multiple versions?
    - Your code might support a legacy project that has to run under multiple production environments. So you want to configure your Travis CI VM to run with whatever libraries you need to pass tests in preparation for deployment to production systems.

How to configure test/ deployment in Travis CI

- git commit that stuff
- git push origin master
- Watch the magic (hopefully work) on Travis CI
- Then watch your project show up on Heroku!!
- Let’s look at a live example

Common Pitfalls

- Assuming a Travis CI Linux instance knows anything about the environment your project needs
- Anything you need to control about Travis CI needs to be set up inside the .travis.yml file

Other options with Travis

- ***Everything*** you need: https://docs.travis-ci.com/
- Other resources: https://www.martinfowler.com/articles/continuousIntegration.html
- Circle CI Resources: https://circleci.com/docs/1.0/