E2E Tests and Continuous Integration

Recap: Unit Tests (component level)

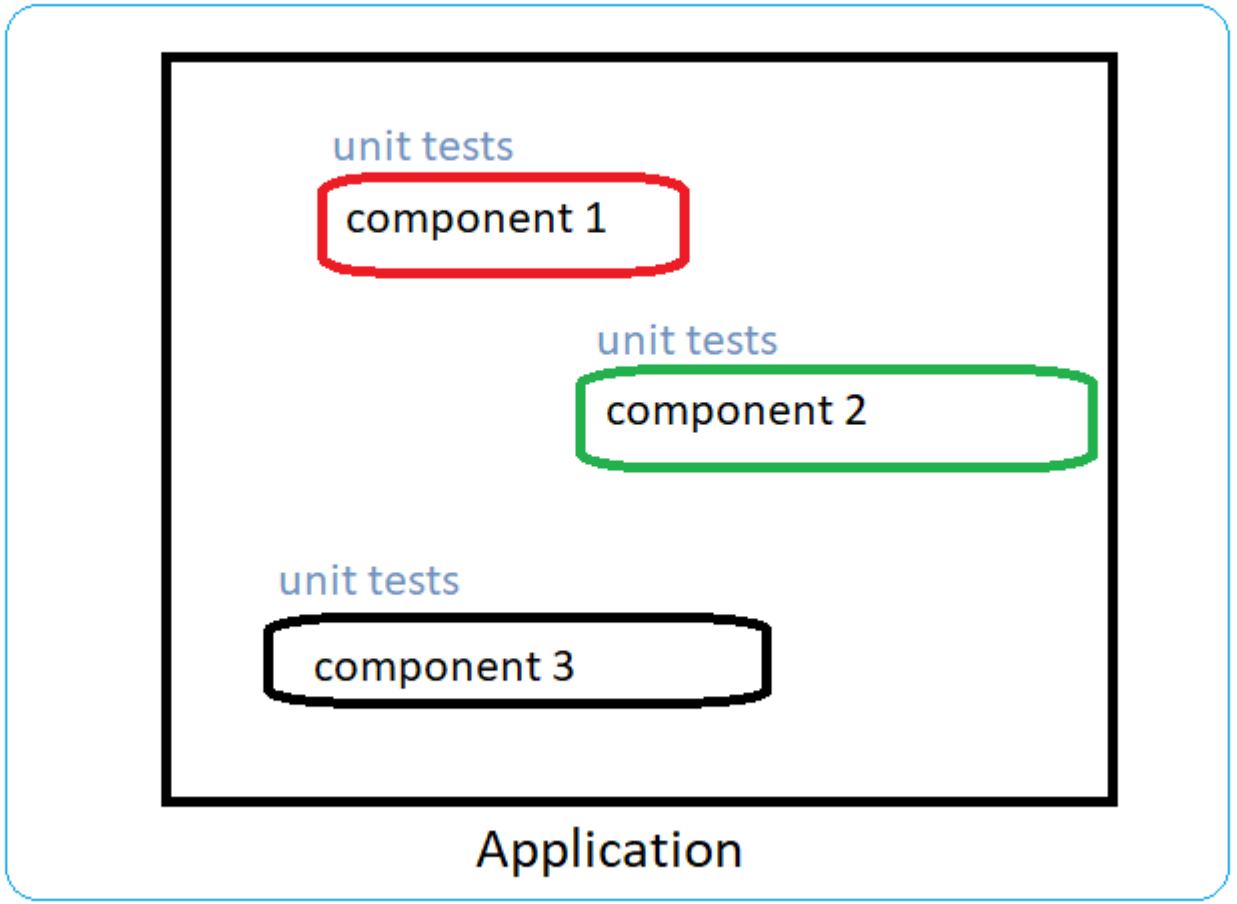
- Tests a unit in isolation. In our case, this is an individual Angular component most of the times.
- Involves mocks + spies on the function calls a lot
- The simplest form of tests, easy to write and understand
- First line of defense against unwanted modification of the source code

E2E (System level)

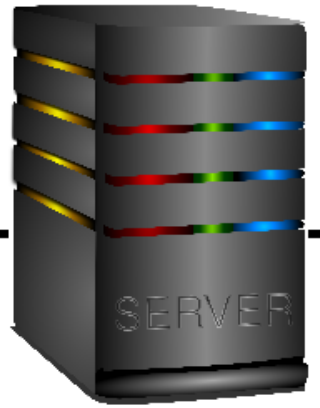
- Stands for end-to-end testing, at a level above of integration test
- Normally, we do no mocking in e2e, as we are interested in testing the actual functionality of all units together
- Usually slower to run, as it tries to test whole compiled application

E2E

- + Great for high-level validation of the entire system.
- + Can spot bugs that unit tests fail to capture
- - Can't give you the comprehensive test coverage that you'd expect from unit tests.
- - Difficult to write and perform poorly compared to unit tests.
- - Break easily, often due to changes or misbehavior far removed from the site of breakage.


unit tests
component 1
unit tests
component 2
unit tests
component 3
Application
e2e tests

Hosted Angular application (on localhost)



Runs tests on

E2E tests (written with Jasmine)

Demo of E2E

- Setup: Protractor (wrapper of Selenium Webdriver) + Jasmine
- Way 1: protractor + npm start
- Selenium and Webdriver interaction
- Way 2: npm run e2e

Continuous Integration

- The underlying idea is to commit small chunks of code at a time
- test gradually, to avoid putting together a lot of code at the same time and having to face hell debugging.
- In our case, we wish to maintain CI for unit testing and possibly E2E, without someone having to manually execute them every single time.

Travis CI

- Paid service, unless open source project
- When a new commit is detected, Travis clones the Github Repo being watched, and runs the new build remotely.
- It will then execute the scripts passed to it via its conf file, and it our case, it executes the tests. We will be notified of the outcome
- Setup is done with a *.travis.yml* file at the root of the project

local workspace

commit some changes

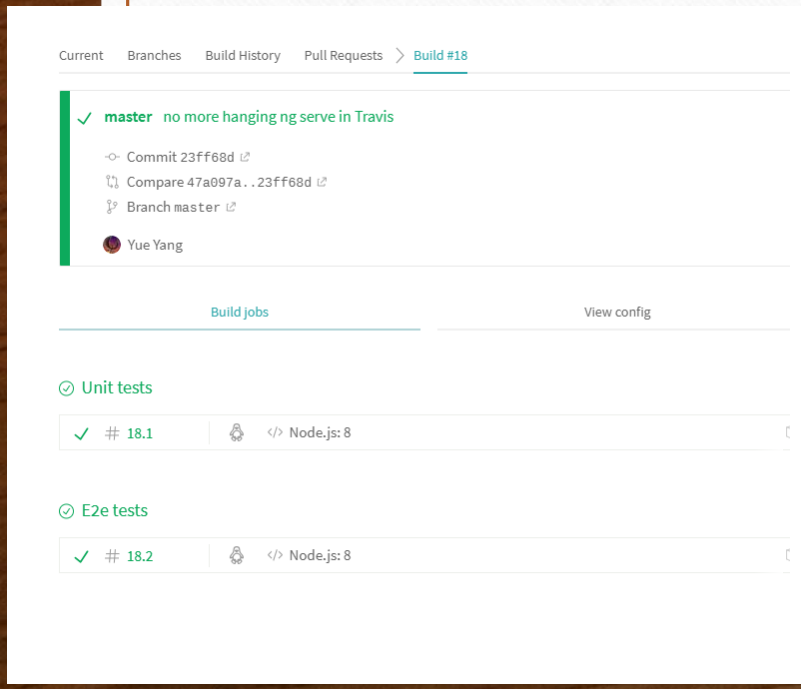
angularUnitTesting
GitHub
Repo

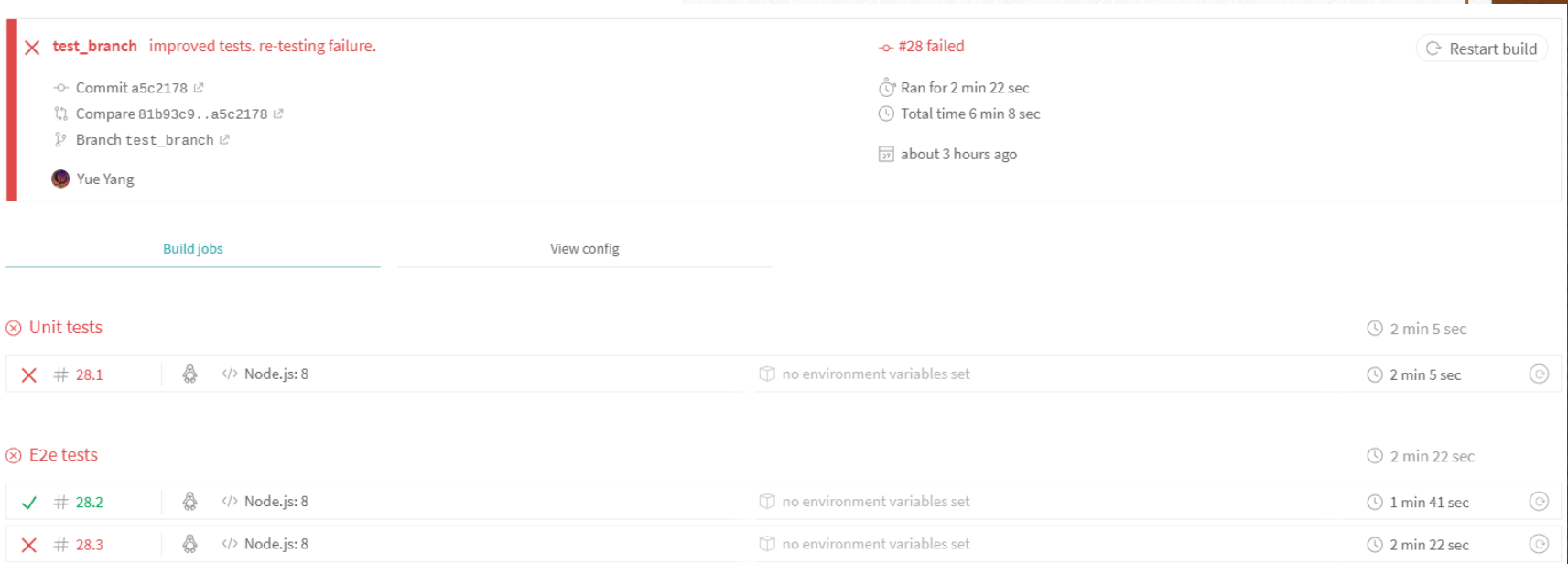
Travis CI

clones repo and creates a new build with the committed changes + runs configured tests on build and notifies us of the outcome

Travis CI Demo

- Successful and failed builds





What if the tests fail?

- Notify concerned users
- Refuse commits/check-ins ? Up to debate
- Another possible setup: Jenkins + Gerrit (requires 10GB+ of hard disk space ☹)

Resources

- https://angular.io/guide/testing
- http://softwaretestingfundamentals.com/system-testing/
- https://docs.travis-ci.com/user/customizing-the-build/ (Travis CI)
- https://www.protractortest.org/#/ (Protractor)
- https://developers.google.com/web/updates/2017/04/headless-chrome