Help Your Developers Help Themselves

Scott Stancil
@hoverduck

AUTOMATTIC

CHAPTER 1

The Problem

Challenges

The project I work on wasn't built with end-to-end testing in mind. Here are a few of the challenges we face:

- The tests live in a separate GitHub repo from the production code
- They run in a separate CI instance, making feedback to the code authors difficult
- The full test suite is only run autonomously against production *after* the code goes live for millions of users.

This means the only way to get early feedback on changes before merge is to run the tests manually. Historically this has only been done by a member of the QA team.

Complaints

As the project matures, we're trying to consolidate the automated tests into the development process, and get a common series of complaints and reasons why the development team can't run the tests themselves:

- The e2e test environment is too hard to set up
- Tests running in live browser windows interrupt their other work
- It isn't clear what set of changes are actually being tested
- The tests take too long

Developers ***love*** the feedback from a good suite of automated tests, but it's generally just easier for them to ping us to run them. And developers hate inefficiency.

CHAPTER 2

The Solution

Make It Convenient

Eliminate the roadblocks between your developers and the tests and you'll find them excited to run them earlier in the development process. The earlier bugs are found, the easier they are to fix.

- The e2e test environment is too hard to set up
    - Eliminate the setup by packaging up your test environment
- Tests running in live browser windows interrupt their other work
    - Hide the browser window in a virtual environment
- It isn't clear what set of changes are actually being tested
    - Run the tests against a local server hosting a specific code branch
- The tests take too long
    - Speed up your tests through increased parallelization

How Do We Do it?

Docker is the magic that will enable you to make your developers' lives easier, which will in turn make your own lives easier

Docker? DevOps?

The concepts of DevOps and Docker containers are often intimidating to folks. But if you're capable of writing a script to automate a web browser, you're capable of building a test infrastructure and sharing it with the world.

The Selenium open source project has made this incredibly easy with pre-built Docker images that address the first two of our developers' complaints:

- The e2e test environment is too hard to set up
- Tests running in live browser windows interrupt their other work

Docker Compose

Docker compose lets you easily consolidate your entire test environment in a single place, a file named `docker-compose.yml`. Let's review this simple example:

```
version: "3"
services:
  selenium:
    image:
selenium/standalone-chrome-debug:3.3.0
    ports:
      - "4444:4444"
      - "5902:5900"
    volumes:
     - /dev/shm:/dev/shm
```

Docker Compose Explained

```
version: "3"
services:
  selenium:
    image: selenium/standalone-chrome-debug:3.3.0
    ports:
      - "4444:4444" <- Selenium port
      - "5902:5900" <- VNC port
    volumes:
     - /dev/shm:/dev/shm
```

^ the debug option enables VNC

^ Ensures Chrome won't crash for lack of shared memory

How Does It Work?

- Just run `*docker-compose up*`, and Docker will launch a miniature Linux VM behind the scenes.
- Set the environment variable *SELENIUM_REMOTE_URL=http://localhost:4444/wd/hub*
- Run your tests as normal.

Problem Solved?

- The e2e test environment is too hard to set up
  - Not anymore 😄

- Tests running in live browser windows interrupt their other work
  - Nope, they're contained in a single VNC window

And as a bonus, you know now that every test run will be against the exact same version of Chrome, Chromedriver, etc. No more "it only works on my machine" 😄

CHAPTER 3

Going Further

The Second Half

The remaining issues on our list are a little more complicated.

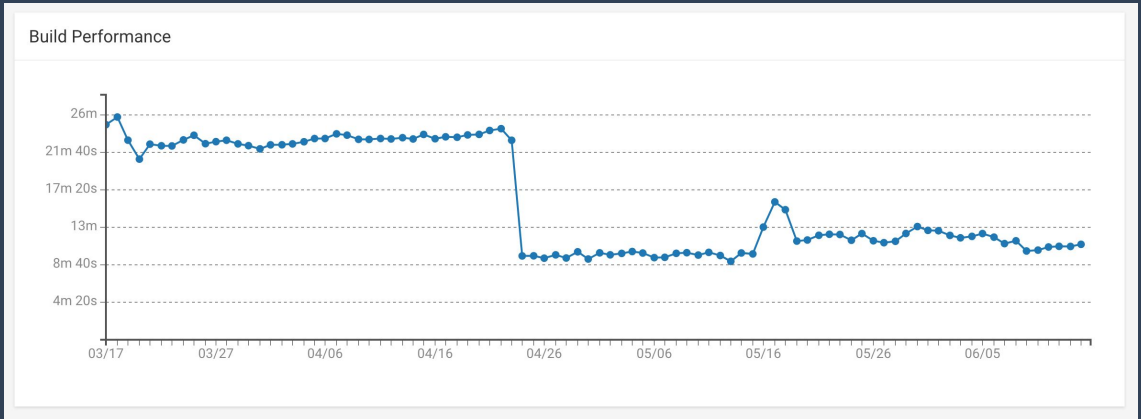
- It isn’t clear what set of changes are actually being tested
- The tests take too long

Let’s start by looking at the last one and work our way back

Parallelization

Test runtime is by far the easiest metric to track, and it's also one of the easiest to address with the least technical effort.

In our project we cut our test execution time in half by using the *Magellan* test runner-runner to increase parallelization without needing any extra hardware.



Testing Specific Changes

It’s complicated, and highly dependent on your specific application. But Docker is extremely flexible.

Add your application server(s) to the docker-compose config. Now your tests and app will run on their own private network, and the app will be using the developer’s own code base.

If your application isn’t currently Dockerized, find the developer in your organization who’s most enthusiastic about the idea and they’ll help!

Loading the app in a container is also useful for general development work and manual testing.

EPILOGUE

In Summary

Problem Solved!

- It isn’t clear what set of changes are actually being tested
    - The developer can be sure that the code being tested is direct from his or her current branch
- The tests take too long
    - It’s easier than you think to increase parallelization

With a Docker-enabled workflow for both testers and developers, everyone’s job is easier in the long run.

Thanks

Scott Stancil
@hoverduck

AUTOMATTIC