Engineering in Focus

the Fandor engineering blog

Automated Testing Clinic follow-up: capybara-webkit vs. poltergeist/PhantomJS

with 2 comments

In my presentation at the February Automated Testing SF meetup I (Dave Schweisguth) noted some problems with Fandor's testing setup and that we were working to fix them. Here's an update on our progress.

The root cause of several of our problems was that some of the almost 100 @javascript scenarios in our Cucumber test suite weren't running reliably. They failed occasionally regardless of environment, they failed more on slower CPUs (e.g. MacBook Pros only a couple of years old), when they failed they sometimes hung forever, and when we killed them they left behind webkit-server processes (we were using the capybara-webkit driver) which, if not cleaned up, would poison subsequent runs.

Although we've gotten pretty good at fixing flaky Cucumber scenarios, we'd been stumped on this little handful. We gave up, tagged them @non_ci and excluded them from our build. But they were important scenarios, so we had to run them manually before deploying. (We weren't going to just not run them: some of those scenarios tested our subscription process, and we would be fools to deploy a build that for all we knew wouldn't allow new users to subscribe to Fandor!) That made our release process slower and more error-prone.

It occurred to me that I could patch the patch and change our deployment process to require that the @non_ci scenarios had been run (by adding a git tag when those scenarios were run and checking for it when deploying), but before I could put that in to play a new problem appeared. When a couple of new engineers joined Fandor and got shiny new laptops running the latest MacOS, all of our @javascript tests failed intermittently for them. It was time to tackle the problem head on.

Step one was to upgrade Capybara, capybara-webkit and Qt to the latest versions. We'd been putting off the Capybara upgrade because Capybara had changed to be more strict. For one, it now requires elements to be styled visible to be found in the DOM unless one specifically includes hidden elements in the search. Revising all of the Cucumber steps in our entire suite (about 800 scenarios) to handle these and other changes in Capybara took several developer-days. But when we were done we were in worse shape than when we'd started. Every @javascript scenario failed intermittently, in every environment, including continuous integration. If this had been a feature change bringing down our business metrics we'd have just rolled it back, but Ruby is a harsh mistress: very few gems maintain backward compatibility with their dependencies for very long, so we knew we needed to stick with the upgrade

and make it work.

Step two was to try a different @javascript driver, an idea we'd had for a while but hadn't yet gotten to. We'd tried the poltergeist gem, which connects Capybara to the PhantomJS headless browser, back when we first set up our @javascript scenarios, but we had an easier time getting capybara-webkit to work so that's what we ran with. But we'd heard good things about PhantomJS lately so it was worth another try.

That first try was pretty impressive: just switching from capybara-webkit to poltergeist made most of our scenarios pass most of the time, and poltergeist's much better error messages allowed us to fix some problem scenarios right away. (capybara-webkit would usually simply fail after it couldn't find the element it was looking for in the alotted time, or an assertion step would fail with no indication that the real problem had occurred several steps earlier. poltergeist helpfully indicates when there are Javascript errors on a page, when a page doesn't load in time or when an element you're looking for is behind another element.) Best of all, poltergeist never hung, but timed out cleanly when it had to.

PhantomJS is also a lot easier to install, both on MacOS and Ubuntu, than webkit and Qt. Previously we'd had to install many dependencies and fight to get specific releases of Qt — capybara-webkit seems to be sensitive to the exact Qt version — on each machine. PhantomJS includes its dependencies in a single binary which didn't require any additional dependencies in any of our test environments.

Nonetheless, we weren't done. Many @javascript scenarios still failed occasionally, and a handful of scenarios failed every time. But poltergeist's specific error messages gave us a clue as to what to do next.

Since, when scenarios failed intermittently, poltergeist complained that the page hadn't loaded in time, **step three** was to see why not. poltergeist provides a handy method, page.driver.network_traffic, that one can call to see the network requests that were made as a page loaded. What we saw was exactly what you'd expect from watching the same page load in your browser debugger: each page loaded dozens of external scripts and other resources, most of them (supporting analytics and advertising) not very relevant to most scenarios. So we just changed our app to not include those script tags in the test environment. The result: all but that last handful of problem scenarios passed. The timeouts must have been due to intermittent slowness of loading those third-party scripts.

Life isn't perfect yet. We haven't figured out those last few scenarios yet, so we're back to our @non_ci solution to keep the build relatively useful. We also need to write a few scenarios that do include the third-party scripts so we cover those lines of template. But we have many fewer @non_ci scenarios than we did with capybara-webkit, and we have all the other advantages of poltergeist/PhantomJS:

- it's more deterministic, in that problem scenarios are more likely to fail consistently than to be flaky
- it's less machine-dependent; our suite now behaves pretty much the same in all of our test environments
- it gives better error messages
- it fails when there are Javascript errors, even if the test would otherwise pass
- it doesn't hang, and
- it's easier to install.

Stay tuned for the eventual elimination of those last few @non_ci scenarios and that whole process antipattern.

Written by fandave

March 2, 2014 at 12:53

Posted in Agile development, Rails, Ruby, Testing

2 Responses

Subscribe to comments with RSS.

[…] Automated Testing Clinic follow-up: capybara-webkit vs. poltergeist/PhantomJS […]

How I spent my working vacation | Dave Schweisguth in a Bottle

March 2, 2014 at 13:54

Reply

Nice post, Dave! Glad to see that PhantomJS is working out for you and you've been able to evolve your test setup there! :)

Leo Cheng

March 3, 2014 at 15:58

Reply