704-918-1799 | www.quoininc.com

Test Automation at Quoin

Boston

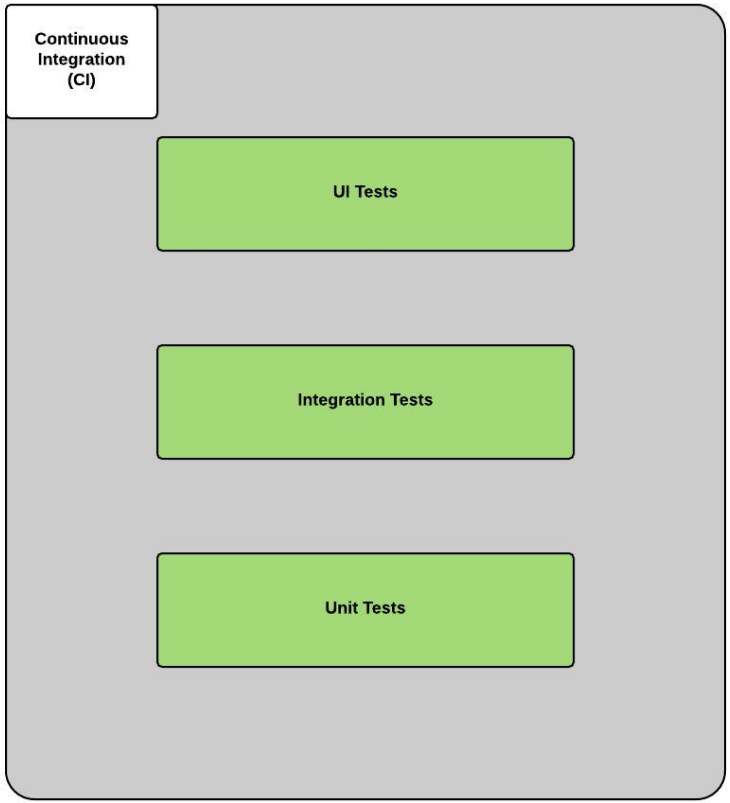
New York

Charlotte

Washington, D.C.

Nicaragua

209 Delburg Street, Suite 207
Davidson, North Carolina 28036



Automated Testing Framework

Unit Tests

- Used to test the lowest level components of a program
  - Test individual functions / methods
- Can use stubs to provide supporting functionality / data needed by the function being tested, but whose execution is not critical to what is being tested.
- Should be created and run by developers as they code
  - Developers can manually execute tests
  - Manually, you can execute a single test, a group of tests, or the entire suite of tests
  - All unit tests should pass clean before checking in / submitting code
- Can be executed automatically by Continuous Integration
  - Runs against main development branch (checked in code)
  - Used to limit errors introduced by each code merge

Unit Tests - test frameworks

- cxxtest - C/C++
- rspec - Ruby on Rails (also see: Better Specs)
- JUnit - Java
- mocha - JavaScript test framework running on Node.js
- jasmine - A competing JS test framework

```ruby
require 'rails_helper'

RSpec.describe Appointment, type: :model do

  # Appointments have validations to ensure scheduled time is not in the past
  # So to ensure our tests behave consistently and predictably, fake the current date and time
  # The scheduled_date in the factory is set to 1 day after this
  before :each do
    DateTime.stub(:now).and_return(Time.utc(2016, "feb", 2, 19, 5, 0))
  end


  it "has a valid factory" do
    expect(create(:appointment)).to be_valid
  end

  it "is invalid without an appointment type" do
    expect(build(:appointment, appointment_type: nil)).not_to be_valid
  end

  it "is invalid without an appointment status" do
    expect(build(:appointment, appointment_status: nil)).not_to be_valid
  end

  it "is invalid without a nurse patient" do
    expect(build(:appointment, nurse_patient: nil, nurse_id:nil)).not_to be_valid
  end

  it "is invalid without a date scheduled" do
    expect(build(:appointment, date_scheduled: nil)).not_to be_valid
  end

  context "when dated scheduled is in the past" do
    it "is invalid" do
      expect(build(:appointment, date_scheduled: '2016-02-01 19:00:00')).not_to be_valid
    end
  end

  context "when updating an existing appointment" do
    before :all do
      @appointment = create(:appointment, date_scheduled: '2016-02-20 19:00:00')
      @complete_status = create(:appointment_status, description: 'completed')
    end

    context "after the scheduled date" do
      before :each do
        # Travel in time to after the appointment
        DateTime.stub(:now).and_return(Time.utc(2016, "feb", 2, 21, 5, 0))
      end

      it "should be valid" do
        @appointment.appointment_status = @complete_status
        expect(@appointment).to be_valid
      end
    end
  end
end
```

Integration Tests

- Test multiple components of a program to ensure they work together properly
- Test build automation with Docker, Chef, Ansible, etc..
- Black-box testing:
  - From the outside in using browser automation tools like Selenium
  - Service endpoints: usually the server-side framework provides tools for testing these

Acceptance Tests

- Slight variation on Integration Tests in that they are approached from a user's perspective
- Cucumber Behavior Driven Development (BDD)
  - JavaScript version: Cucumber-js
  - Tests are written as Features and Scenarios that look similar to user stories
  - They use Gherkin syntax (Given - When - Then)
  - Tests can be read and understood by business users
  - 'Step definitions' use regular expressions to connect the lines of the Features and scenarios to code that drives the test
  - Can be integrated with Selenium to execute the test steps in a browser
  - Overlap of UI and Integration tests

```
Feature: Display Help Text View Empty Section
  As a User, I want to see a help text on empty forms sections

  Scenario: I should see a help text on "Photos and Audio" and "Other Documents" forms when is empty for cases
    Given I am logged in as an admin with username "primero_cp" and password "primero"
    When I access "cases page"
    And I press the "New Case" button
    And I press "Save"
    Then I should see a success message for new Case
    And I press the "Photos and Audio" button
    And I should see "Click the EDIT button to add Photos and Audio details" on the page
    And I press the "Other Documents" button
    And I should see "Click the EDIT button to add Other Documents" on the page

  Scenario: I should see a help text on "Photos and Audio" and "Other Documents" forms when is empty for tracing requests
    Given I am logged in as an admin with username "primero_cp" and password "primero"
    When I access "tracing requests page"
    And I press the "New Tracing Request" button
    And I press "Save"
    Then I should see a success message for new Tracing Request
    And I press the "Photos and Audio" button
    And I should see "Click the EDIT button to add Photos and Audio details" on the page

  Scenario: I should see a help text on "Other Documents" forms when is empty for incidents
    Given I am logged in as an admin with username "primero_mrm" and password "primero"
    When I access "incidents page"
    And I press the "New Incident" button
    And I press "Save"
    Then I should see a success message for new Incident
    And I press the "Other Documents" button
    And I should see "Click the EDIT button to add Other Documents" on the page
```

```ruby
Then /^I should see a success message for (new|updated) (Case|Incident|Tracing Request)$/ do |action, model|
  page.should have_selector(:css, "p.notice")
  message_text = page.find(:css, "p.notice").text
  if action == "new"
    message_text.match(/^#{model} record (.*) successfully created.$/).nil?.should eq(false)
  elsif action == "updated"
    message_text.match(/^#{model} (.*) was successfully updated.$/).nil?.should eq(false)
  end
end

Then /^I should see "([^\"]*)" on the page$/ do |text|
  page.has_content?(text).should == true
end

Then /^I should not see "([^\"]*)" on the page$/ do |text|
  expect(page).to have_no_content(text)
end

Then /^I should see (a|an) "([^\"]*)" (button|span button) on the page$/ do |grammar, label, button_type|
  if button_type == "span button"
    expect(page).to have_selector("//span[contains(@class, 'span_button')]", :text => label)
  else
    expect(page).to have_selector(:link_or_button, label)
  end
end

Then /^I should not see (a|an) "([^\"]*)" button on the page$/ do |grammar, label|
  expect(page).to_not have_selector(:link_or_button, label)
end

Then /^I should see a "([^\"]*)" link on the page$/ do |label|
  expect(page).to have_selector(:link_or_button, label)
end

Then /^I should see "(.*)" logo|logos in the header$/ do |count|
  page.should have_css("ul.agency_logos li img", :count => count)
end
```

UI Tests

- Selenium - a tool to automate web browser interaction
  - Supports cross OS and browser testing via Selenium Grid: IE, Chrome, Firefox, Safari, Opera
  - Multiple programming languages are supported to launch the tests
  - Can be integrated with various testing frameworks
  - CAUTION
    - Can be susceptible to race conditions.
    - Much slower than unit tests
- PhantomJS - A headless, in-memory JS browser
  - Faster execution
  - No reliable cross browser/platform testing

Code Quality Tools

- Code Coverage
- Static analyzers
- Code Metrics
- Code profilers

Code Coverage

- Shows which functions/methods are and are not being tested
- Within those functions/methods, which lines are and are not being tested
- Gives percentages of the code being tested
- Tools
  - Bullseye - code coverage for C/C++
  - Simplecov - code coverage ruby gem for Ruby on Rails
  - Istanbul - code coverage for JavaScript

Static Analyzers

- Help detect bugs and quality issues not caught by the compiler
- Detects potential bugs without executing the program
- Built into some IDE's such as Rubymine
- Tools
  - ESLint - Syntax code analyzer for JavaScript, points out poor code construction
  - FlexeLint - Syntax code analyzer for C/C++
  - Rubymine - IDE for Ruby on Rails that has built in analysis tools

Code Metrics

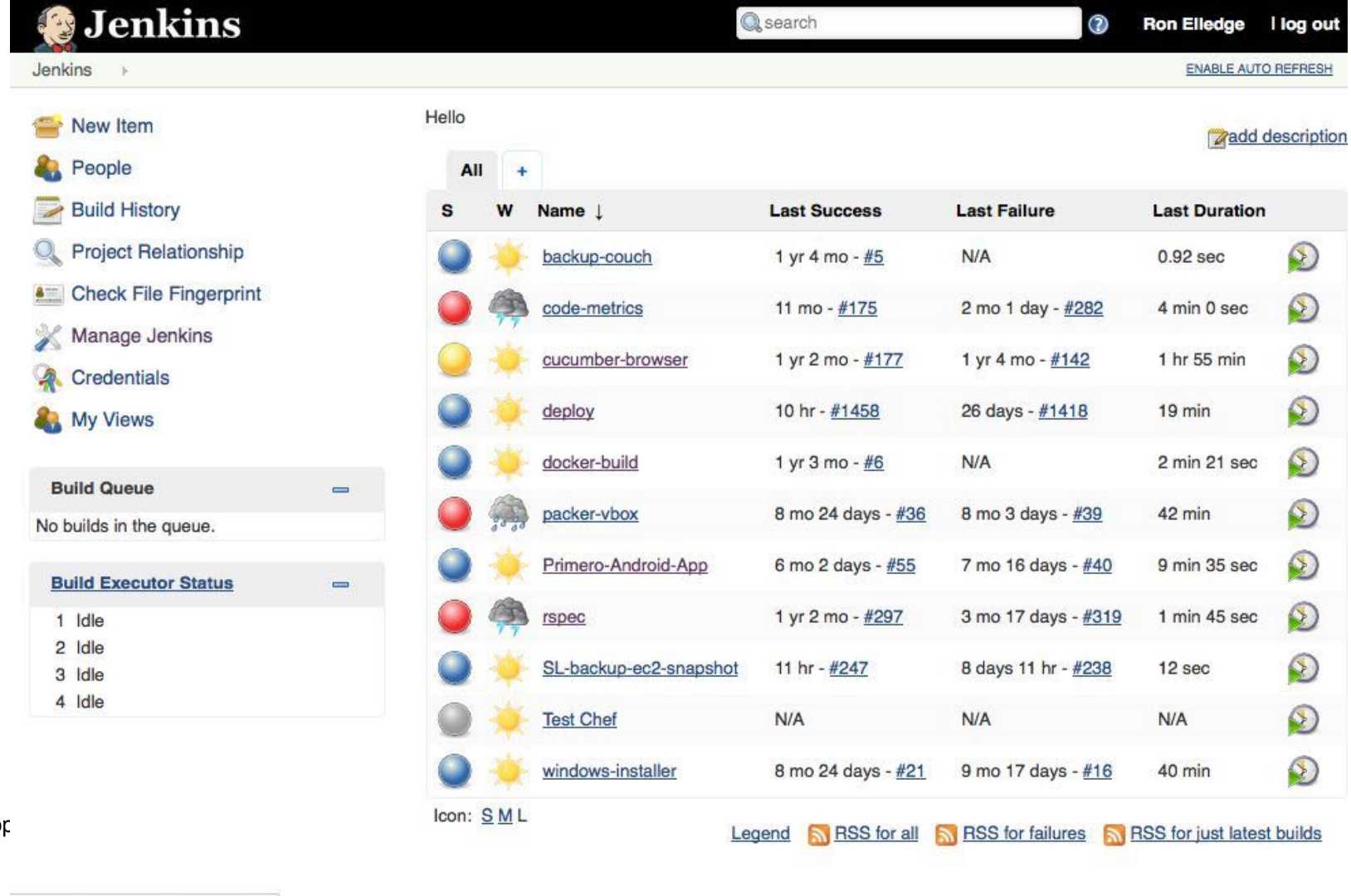
- Help determine complexity
- Help determine non-standard practices
- Tools
  - metric_fu - ruby gem containing suite of multiple code metrics tools
  - CCCCC - code metrics tool for C/C++
  - ESLint - static analysis tool for JavaScript. Also has complexity measurement capabilities

Code Profilers

- Used for optimization
- Analyze memory usage
- Show frequency and duration of function calls
- Tools
  - rack-mini-profiler - Ruby gem to profile Ruby on Rails apps
  - Valgrind - Profiling tools for C / C++

Continuous Integration - Jenkins

- Jenkins is a continuous test, build, and delivery system
- Test
  - Manually run unit, integration, and UI tests from the Jenkins interface
  - Schedule tests to run at specified times / intervals
  - Trigger tests to be run when code is checked in
- Build
  - Has plugins to support build of multiple languages
- Delivery
  - Can be used to deploy a project to test and production servers
- Integrates with repositories
  - Git, CVS, Subversion
  - GitHub, Bitbucket
- Integrates with ticketing systems such as Jira
- Can run builds and tests on remote servers using ssh


Jenkins
search
Ron Elledge
log out
Jenkins
ENABLE AUTO REFRESH
New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
Credentials
My Views
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
3 Idle
4 Idle
Hello
add description
All
+
S
W
Name ↓
Last Success
Last Failure
Last Duration
backup-couch
1 yr 4 mo - #5
N/A
0.92 sec
code-metrics
11 mo - #175
2 mo 1 day - #282
4 min 0 sec
cucumber-browser
1 yr 2 mo - #177
1 yr 4 mo - #142
1 hr 55 min
deploy
10 hr - #1458
26 days - #1418
19 min
docker-build
1 yr 3 mo - #6
N/A
2 min 21 sec
packer-vbox
8 mo 24 days - #36
8 mo 3 days - #39
42 min
Primero-Android-App
6 mo 2 days - #55
7 mo 16 days - #40
9 min 35 sec
rspec
1 yr 2 mo - #297
3 mo 17 days - #319
1 min 45 sec
SL-backup-ec2-snapshot
11 hr - #247
8 days 11 hr - #238
12 sec
Test Chef
N/A
N/A
N/A
windows-installer
8 mo 24 days - #21
9 mo 17 days - #16
40 min
Icon: S M L
Legend
RSS for all
RSS for failures
RSS for just latest builds

Process Tips...

Testing with a database

- When you execute tests that update or rely on data in a database, you should always begin and end the test with the database in the same state. This is to ensure each run of the tests performs consistently with the same results.
- There are multiple strategies for handling this
  - Docker: Using Docker, you can have a container for your app and a container for your database. Then set up the test suite to build a container for your app in test mode, build a container for the test database, execute the tests, then tear down the containers. Each run of the tests starts with a fresh app and a fresh database.
  - Transactions: Wrap the tests in transactions which roll back the data at the end of the test run so each run of the tests starts with the same data.

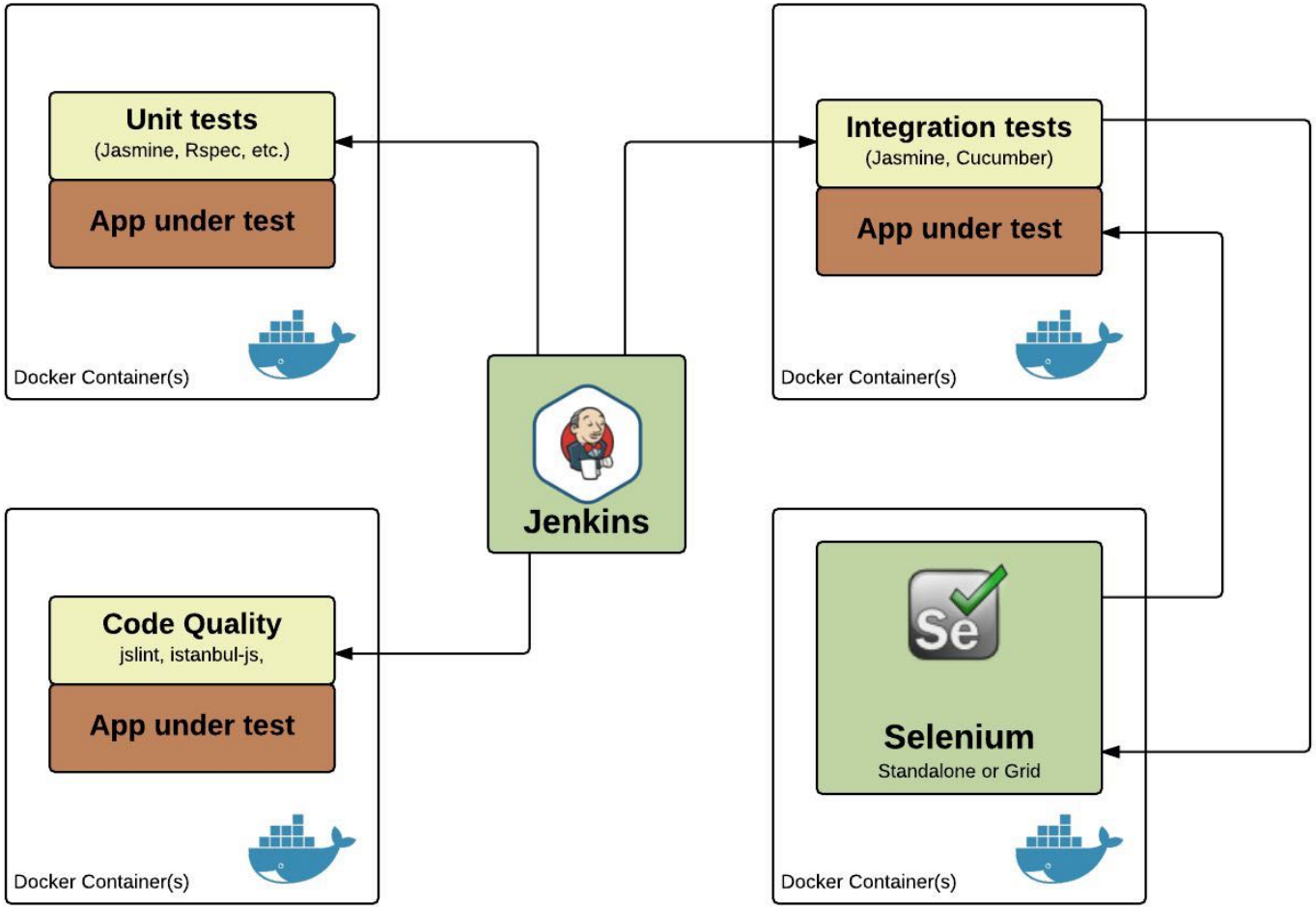
Test all paths

- You want your tests to exercise the edge cases, not just the “happy path”
- You should test error conditions
- You should test with valid and invalid input data
- Code coverage tools help to ensure all paths through your code have been covered by the test cases

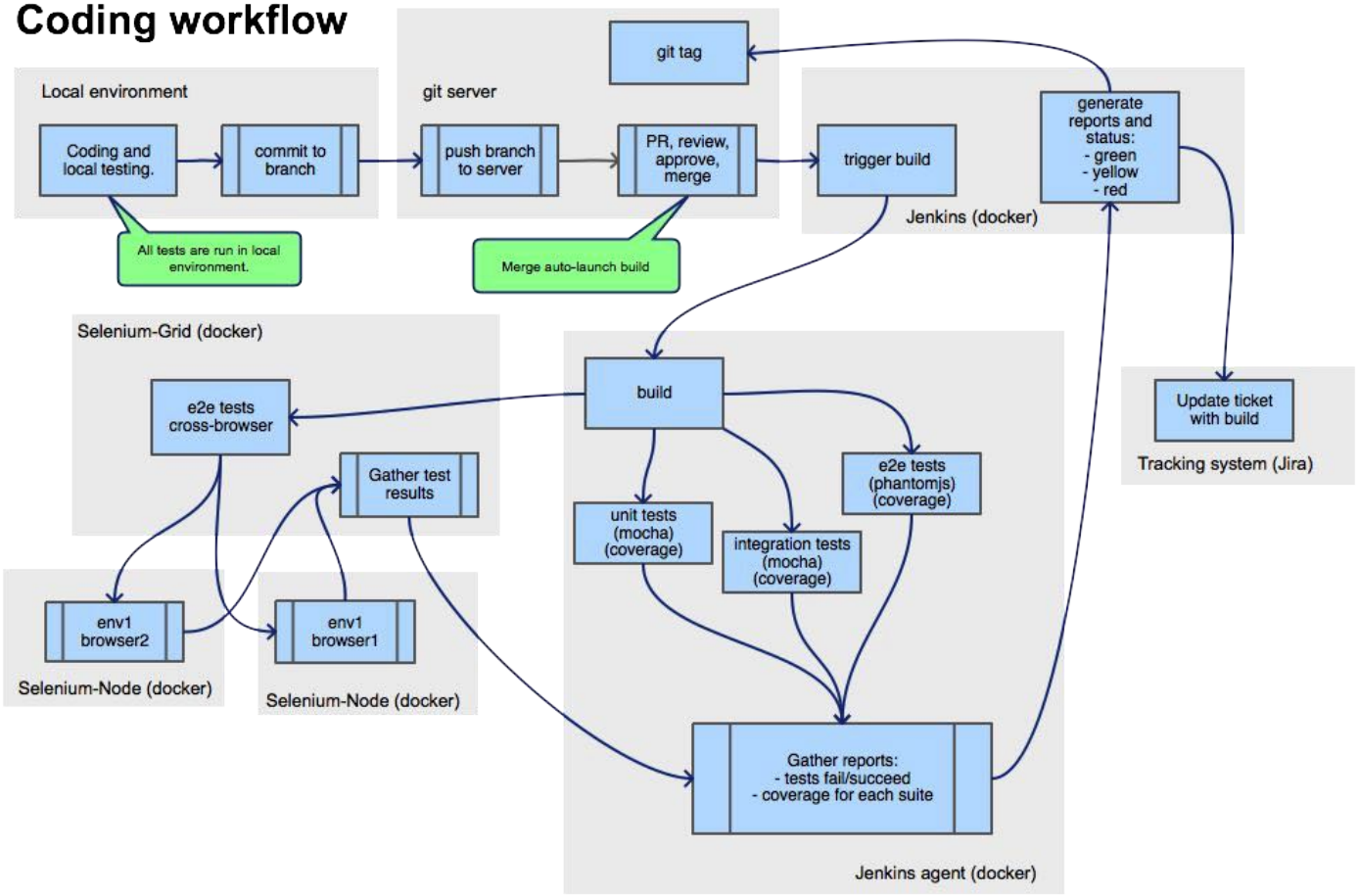
Putting it all together...

Implementation Strategy

- The testing tools shown in these slides can be executed manually as well as be triggered from a Continuous Integration system.
- Continuous integration is not a requirement for the tools. They can all be implemented individually.
- You can easily begin by implementing one or two of the tools then incrementally add other tools until you have a fully integrated test system.
- There is no requirement to begin by having tests run on remote servers. The developers can manually execute the tests in their development environment.


Unit tests
(Jasmine, Rspec, etc.)
App under test
Docker Container(s)
Integration tests
(Jasmine, Cucumber)
App under test
Docker Container(s)
Jenkins
Code Quality
jslint, istanbul-js,
App under test
Docker Container(s)
Selenium
Standalone or Grid
Docker Container(s)

Coding workflow



Lessons Learned

- Writing some of the tests may seem tedious at first, but get easier as you become familiar with the tool.
- Once the tests are written, they are very excellent at catching errors during future iterations.
- If you have good code coverage, regression testing is easy.
- There are initial setup costs. Setting up some of the pieces like a Selenium Grid and Jenkins requires a certain level of expertise in those tools.
- Setup cost of executing these test tools manually is minimal
- Once set up, executing tests and deployments are easy.
- Cucumber and Selenium tests are a lot slower than unit tests.
- Selenium is prone to race condition issues. It is tempting to put sleep / timer statements to address. In the case of Cucumber, we found it was better to use commands that wait for an expected event to occur before proceeding

Lessons learned - Project Management

- There is a time investment to creating automated tests.
- This additional development cost should be more than offset by time saved in testing and bug remediation.
- Everyone on the team has to buy in. In some experiences, a team created automated unit tests, but then other teams working on the same code base resisted following the process. Unless all teams maintain the existing tests and create new tests for new development, the tests become unreliable and much harder to maintain.