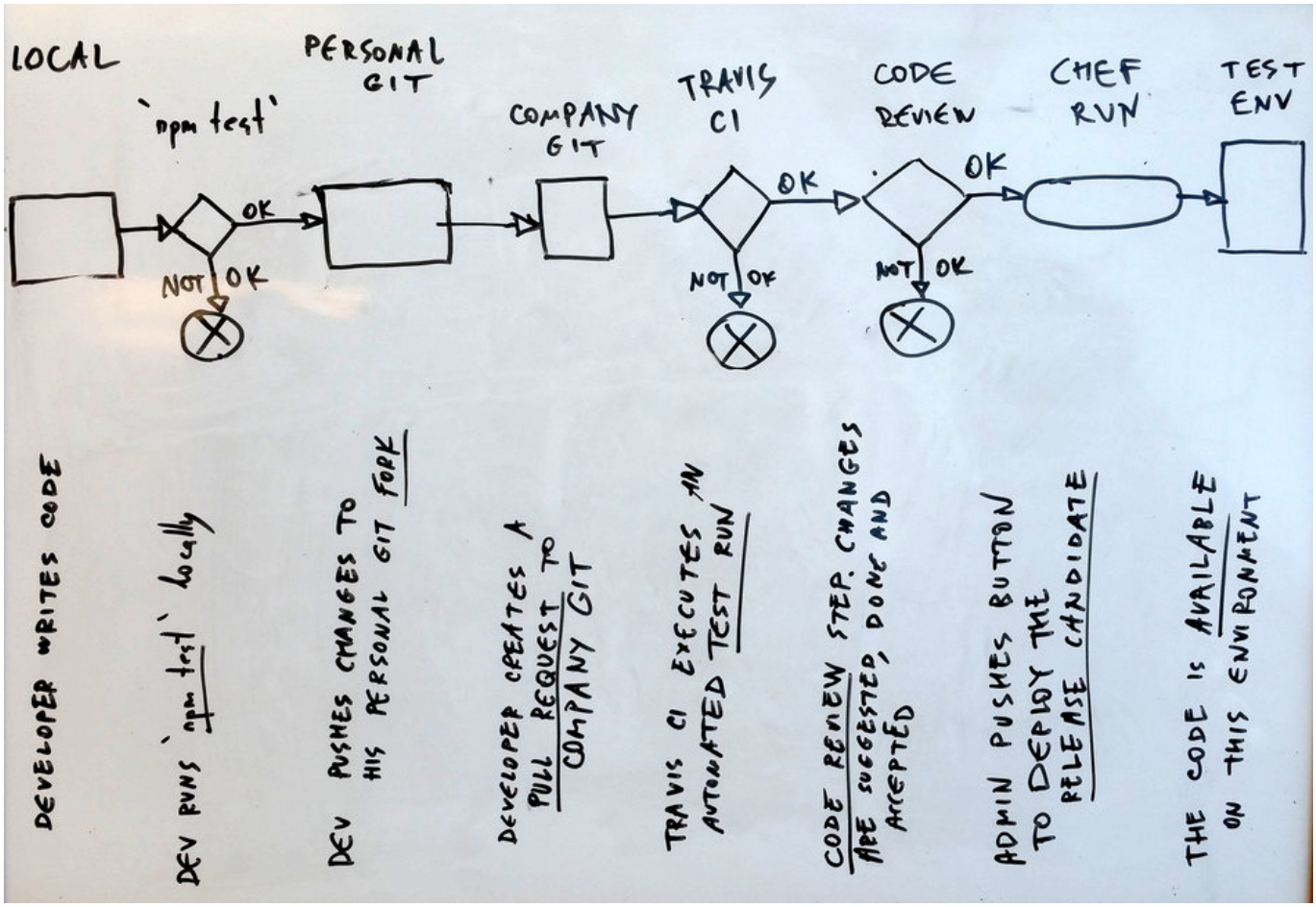
2016Q1 Sturfee Build Pipeline

This document describes some steps for enabling an automated build/deploy process at Sturfee. It gives some recommendations on tools and processes to use. It introduces some basic actionable concepts of Continuous Integration.



Environments

Let's agree on the **naming convention** that the subdomain name is equal to the environment name. Therefore, the environment demo1 is also demo1.sturfee.com, and the environment demo_1, if exists, would be at demo_1.sturfee.com . Here are the suggested environment names:

- demo1
- demo2
- dev
- production = sturfee.com
- staging
- test

Staging is equivalent to production, only without the users hitting the site, and possibly without the full database.

dev environment is used for manual testing.

test environment is a part of the continuous integration pipeline whereas the code goes through several verification steps to end up deployed in production. So the **release candidate** (a git reference such as a branch HEAD or a commit hash) gets deployed to `test` first, and eventually to `production` or `staging`.

Environments will be created manually and used automatically. Each environment definition (setup) will be static, meaning that characteristics of an environment will not change. Use multiple environments if the characteristics of one do not answer your needs.

Tools

git

We are already using git, which is a very good thing. We will also start using Pull Requests, which in turn necessitates that copies of the same codebase are forks. * git forks * git pull requests

npm

For angular.js code, we can use npm as our build automation tool. Run `npm test` to run a collection of tests on the codebase. * npm test

travis CI

Travis (travis-ci.org) will be seamlessly integrated and will provide us with automated test runs on every commit and every pull request.

Chef

To deploy the code to an environment, run `sudo chef-client` on that environment. * sudo chef-run

Testing

Testing is important. We will have several test steps: * eslint * unit tests (jasmine or Karma) * functional tests * integration tests * Selenium tests of the UI

New functionality whould be written along with automatic tests to check it. Every new function created should have a corresponding test.

JS testing frameworks

Take a look at these two possible test frameworks. * Karma (https://karma-runner.github.io/0.13/index.html) * Jasmine (http://jasmine.github.io/2.4/introduction.html)

Go testing

Lets see if one of these frameworks answers our needs: * ginkgo https://github.com/onsi/ginkgo * convey https://github.com/smartystreets/goconvey

Actionable Steps

- personal copies of sturfee_server should be forks of sturfee/sturfee_server. I notice that Peter's and Sturfee's versions aren't forks of one another.
- Please create Pull Requests (PRs) rather than commiting code to the main branch directly. This will allow the team to conduct code reviews.
- Allow us to build sturfee_server on some new environment, say on demo1.
- We can aggregate all configuration parameters in a single place, and allow Chef runs to configure the future environments based on these parameters.