Testing in Node.JS for Continuous Deployment Win

Niall O'Higgins

niallo@beyondfog.com
@niallohiggins

What is Continuous Deployment?

- Various definitions
- Starting at most naïve - “push on green”
  - Run tests on each commit
  - If tests pass, deploy to production
- Key implications of CD:
  - Excellent test coverage
  - Automated deployments
  - Small batches
  - Monitoring

Testing 101

- How are tests useful?
  - Implied specification or contract
  - Catch syntax errors in dynamic languages
  - Enable refactoring
  - Greatly decrease developer stress levels
  - Greatly increase confidence in software

Different kinds of tests

- Terminology varies but here goes:
- Unit tests – function/object level. Very fast to run. Limited scope, can use dependency injection.
- Integration tests – Working with some external resource e.g. MongoDB, Github API, etc. Great coverage, typically a lot slower.
- Functional tests - Full stack browser/HTTP-level tests. Slow, good coverage, can be brittle (e.g. Selenium).

Testing in Node.JS

- Not built into frameworks “out-of-the-box” a la Rails, Django, Pyramid etc.
- How do you do mocking and dependency injection?
- Tools exist but hard to know where to start.
- Many different options. Which should I use?

Choose a Test Runner

- Nodeunit – simple, lightweight, setup teardown supported.
- Mocha – Similar to Nodeunit, more bells and whistles (e.g. test creation API, pluggable output formatters)
- Vows – More opinionated BDD framework.

Choose an Assertion Lib

- Node stdlib 'assert' module a good place to start
- Should.js – patches Object prototype and gives you code like:

  user.should.have.property('pets').with.lengthOf(4);
- Chai – Can work like should.js, or in a more TDD style

Mocking/Stubbing/Dependency Injection

- Test code depending on an external resource (e.g. database) without actually talking to that external database.
- Fake the behaviour for testing purposes.
- Sinon.JS has tons of support for this.
- Node-sandboxed-module by @felixge enables this to work across modules. Run module in new V8 context, monkey-patch require return values.

Node-Sandboxed-Module

```
// database.js
exports.query = function(key, cb) {
 // ... implementation ...
}
// user.js - functions to manage users of the system
var database = require('./database.js');
exports.get_user = function(username, cb) {
  database.query("user_" + username, function(err, user) {
      if (err) throw err;
      var res = {};
      // ... do some additional processing then call cb() ...
 });
}
```

NPM Test

- NPM has wonderful support for “test this package”
- npm test
- Just provide a test script in package.json
- Please support this simple interface :-)

Deploying Node.JS

- Node.JS is fast – often acceptable to run “stand-alone” or behind a load balancer
- Few load balancers support Websockets :(
- Node-http-proxy does :)
- Terminate SSL somewhere else (e.g. node-http-proxy, stud, Nginx/ELB if no Websockets)
- Fabric/Capistrano or similar for code pushes
- Puppet/Chef or similar for system management

Deploying Node.JS

- Or just use a PaaS!
- Heroku, dotCloud and nodejitsu all have Node.JS support
- Heroku most mature, but doesn't support Websockets
- DotCloud supports Websockets and great for polyglot applications
- Nodejitsu in private beta, tons of open source contributions, ask in IRC for invite

Runtime management

- Auto-restarts - “forever” a useful tool for keeping node.js processes up in case of crashes.
- Forever also harvests standard output and standard error and logs to files.
- Recommend using a wrapper like forever.
- Tmux/screen also nice option to re-attach later.

Logging

- Winston – supports various log levels and outputs such as console, rotated on-disk file, Loggly, etc.
- Bunyan – log levels, JSON format, pretty printing-tools
- Forever or similar useful for capturing any STDIO streams.

Continuous Deployment

- Each commit, code goes to production
  - Very scary at first
  - Better have good tests!
  - Better have solid deploy automation!
  - Better have monitoring!
- Do hard things all the time. Many deploys per day.
- Small batch size. No jumbo releases.

Checks and Balances

- Tests can have bugs too
- Code review process.
- Quickly detect problems
  - Monitoring for error rates (ratio for 5xx vs 2xx)
  - Cloudwatch/Nagios alerts (Massive CPU spike after deploy, uh oh)
- Stagger release (deploy to 10% of users first)
- Rollback easily (keep previous version on disk, flip a symlink)

Results

- No fear of deployment.
- Features pushed whenever.
- Enforces good test coverage.
- Enforces solid monitoring and collection of very useful metrics.
- Decrease stress levels.

- Current state of the art in process (IMHO)

Thanks

- We are building a hosted solution for Continuously Deploying Node.JS & Python apps
- Private Beta - Looking for testers http://striderapp.com
- Talk to me if interested in trying it!
- niallo@beyondfog.com
- @niallohiggins