Change Management with Puppet

2011-04-13
PuppetNYC Users Group

Garrett Honeycutt
Professional Services Consultant
garrett@puppetlabs.com
http://linkedin.com/in/garretthoneycutt
We are hiring

• Professional Services
• Core Developer
• Front End Web Developer
• QA/Test Eng
• Web Marketing Manager
• Sales Account Manager
PICC

Professional IT Community Conference

- April 29th and 30th
- Hyatt Regency in New Brunswick, NJ
- Travel expenses provided
What?

Change – “an event that results in a new status of one or more configuration items”[1]

Why?

Environments are the same!

Dev == QA == Staging == ... == PROD
Why?

Compliance with Change Management policies

- CAB – Change Approval/Advisory Board
- Different environments have different criteria for passing to the next one
Different Environments

Puppet Test Area -> Dev -> QA -> Prod

Each environment has different teams and sometimes conflicting goals
Gate Examples

Puppet Test Area -> Dev
• Dev’s agree/know of change

Dev -> QA
• Dev’s have completed and self tested

QA -> Prod
• QA team has verified systems
• Ops is ready (has runbooks, monitoring setup, ... )
Documentation and Policies

Understand your environments

- What are they?
- What is their order of precedence?
- What are their SLA’s?
- Who owns them?
Documentation and Policies

Understand gating factors for change

• What are the gates between each environment?
• Who approves them?
• In what forum are they approved?
VCS Structure (SVN view)
VCS Structure (git view)

same as SVN except

- you do **not** have separate directories for
  - trunk
  - branches
  - tags
VCS Structure

**trunk / master**

- New code that is the best known *working code*
- but still not very well tested ...
VCS Structure

branches

- short lived
- use topical branches!
- associate branches with ticket numbers, so you can leverage your ticketing system to capture who is requesting changes and why
- avoid assigning branches to people as they tend to be long lived
VCS Structure

tags

- immutable (even if you can technically make changes)
- found that BIND style serials work quite well for naming tags
- 2011041300 would be the first tag on April 13th, 2011.
Flow

- Change request comes in (from your ticket system)
- You create a branch from trunk/master that corresponds with the request
- Make changes to the branch
- Merge the branch back into trunk/master
- test against trunk/master
- create a tag
- associate that tag with the next environment all the way through to Prod
Flow
Oops, we found a bug

- tags are immutable, remember?
- create a brand new tag off of trunk/master
- start the process from the beginning
- short-cuts are more expensive
Release Management

Multiple people making changes?

• You need a release manager to be responsible for merging from branches into trunk/master

• Potentially rotate who holds this position
Multiple teams exchanging code?

- Investigate using multiple module paths
- Communication!
- private github – can facilitate cooperation
Mailing List of changes

Create a mailing list for all changes

• You can always ignore it

• reach out to those writing poor code **before** they ask you to merge it into trunk

• svnmailer is great
Testing trunk/master

Create at least one representative system for each different type of system you model

- Run these systems off the code in trunk/master
- Before cutting a tag, rebuild all these systems from scratch
  - further tests that relationships between resources are working
  - proves you can actually provision a system from scratch
Approaches to testing branches

- Puppet’s understanding of environments is good for this
- Setup a different Puppet master per branch
- Do not rely on a puppet master at all -- use puppet apply and test locally