



Continuously deliver your puppet code with
jenkins, r10k and git

Toni Schmidbauer

October 13, 2014

whoami

- ▶ SysAdmin@s-itsolutions.at
- ▶ toni@stderr.at
- ▶ stderr@jabber.org
- ▶ <http://stderr.at>
- ▶ <http://github.com/tosmi>

Agenda

- ▶ A short story about configuration management
- ▶ What is continuous delivery
- ▶ Tools used to achieve continuous delivery
- ▶ DEMO
- ▶ Things to improve

A short story about configuration management (CM)

- ▶ We manage a very diverse environment of UNIX/Linux Systems (Solaris 10/11 sparc/i386, AIX, RHEL/CentOS 5/6/7)

A short story about configuration management (CM)

- ▶ We manage a very diverse environment of UNIX/Linux Systems (Solaris 10/11 sparc/i386, AIX, RHEL/CentOS 5/6/7)
- ▶ We've got around 1000 nodes in total

A short story about configuration management (CM)

- ▶ We manage a very diverse environment of UNIX/Linux Systems (Solaris 10/11 sparc/i386, AIX, RHEL/CentOS 5/6/7)
- ▶ We've got around 1000 nodes in total
- ▶ Before CM we had **strict** standards on how to manage these systems

A short story about configuration management (CM)

- ▶ We manage a very diverse environment of UNIX/Linux Systems (Solaris 10/11 sparc/i386, AIX, RHEL/CentOS 5/6/7)
- ▶ We've got around 1000 nodes in total
- ▶ Before CM we had **strict** standards on how to manage these systems
- ▶ The problem:
 $\text{count}(\text{teammembers}) == \text{count}(\text{standards})$

A short story about configuration management (CM)

- ▶ We manage a very diverse environment of UNIX/Linux Systems (Solaris 10/11 sparc/i386, AIX, RHEL/CentOS 5/6/7)
- ▶ We've got around 1000 nodes in total
- ▶ Before CM we had **strict** standards on how to manage these systems
- ▶ The problem:
count(teammembers) == count(standards)
- ▶ So configuration management is the solution to all our problems

The solution to all our problems

The solution to all our problems

- ▶ Broke our systems

WHY????

Problems with our old CM system

- ▶ Deployments sucked

Problems with our old CM system

- ▶ Deployments sucked
 - ▶ Deployment via manual tagging and checkout, so mistakes happened
 - ▶ Deployment in stages, but we always had to cross our fingers

Problems with our old CM system

- ▶ Deployments sucked
 - ▶ Deployment via manual tagging and checkout, so mistakes happened
 - ▶ Deployment in stages, but we always had to cross our fingers
- ▶ Testing sucked

Problems with our old CM system

- ▶ Deployments sucked
 - ▶ Deployment via manual tagging and checkout, so mistakes happened
 - ▶ Deployment in stages, but we always had to cross our fingers
- ▶ Testing sucked
 - ▶ No Unittest
 - ▶ No acceptance tests

Problems with our old CM system

- ▶ Deployments sucked
 - ▶ Deployment via manual tagging and checkout, so mistakes happened
 - ▶ Deployment in stages, but we always had to cross our fingers
- ▶ Testing sucked
 - ▶ No Unittest
 - ▶ No acceptance tests
- ▶ No immediate feedback if things where ok or **not**

Problems with our old CM system

- ▶ Deployments sucked
 - ▶ Deployment via manual tagging and checkout, so mistakes happened
 - ▶ Deployment in stages, but we always had to cross our fingers
- ▶ Testing sucked
 - ▶ No Unittest
 - ▶ No acceptance tests
- ▶ No immediate feedback if things where ok or **not**
- ▶ Systems installed without CM are hard to bring under CM control

Problems with our old CM system

- ▶ Deployments sucked
 - ▶ Deployment via manual tagging and checkout, so mistakes happened
 - ▶ Deployment in stages, but we always had to cross our fingers
- ▶ Testing sucked
 - ▶ No Unittest
 - ▶ No acceptance tests
- ▶ No immediate feedback if things were ok or **not**
- ▶ Systems installed without CM are hard to bring under CM control
- ▶ Every system was a special case

So whats our solution?

Continuous delivery

- ▶ is a pattern for getting software from development to release

1

Continuous delivery

- ▶ is a pattern for getting software from development to release
- ▶ this pattern is called **the deployment pipeline**

1

The deployment pipeline



Why are we using continuous delivery

- ▶ When you automate your deployment,

Why are we using continuous delivery

- ▶ When you automate your deployment,
 - ▶ less mistakes will happen

Why are we using continuous delivery

- ▶ When you automate your deployment,
 - ▶ less mistakes will happen
 - ▶ the same mistake will not happen twice

Why are we using continuous delivery

- ▶ When you automate your deployment,
 - ▶ less mistakes will happen
 - ▶ the same mistake will not happen twice
 - ▶ you need to test everything

Why are we using continuous delivery

- ▶ When you automate your deployment,
 - ▶ less mistakes will happen
 - ▶ the same mistake will not happen twice
 - ▶ you need to test everything
 - ▶ you deploy more often

Why are we using continuous delivery

- ▶ When you automate your deployment,
 - ▶ less mistakes will happen
 - ▶ the same mistake will not happen twice
 - ▶ you need to test everything
 - ▶ you deploy more often
- ▶ The more you deploy, the more confident you get in your deployment

Copyrighted Material

The Addison-Wesley Signature Series



A MARTIN FOWLER SIGNATURE
BOOK
Martin

CONTINUOUS DELIVERY

RELIABLE SOFTWARE RELEASES THROUGH BUILD,
TEST, AND DEPLOYMENT AUTOMATION

JEZ HUMBLE
DAVID FARLEY



Foreword by Martin Fowler

Copyrighted Material

Tools to build a deployment pipeline

Jenkins

- ▶ Jenkins is an Open Source continuous integration server
- ▶ It's purpose is to execute and monitor jobs
- ▶ Jobs are shell scripts or any other thing that's executable and returns 0 on success
- ▶ Many plugins available to extend Jenkins (e.g. git, build-pipeline, monitor)
- ▶ You can **link jobs** together, thats our **pipeline**

Jenkins II

Build Pipeline



Monitoring with Jenkins



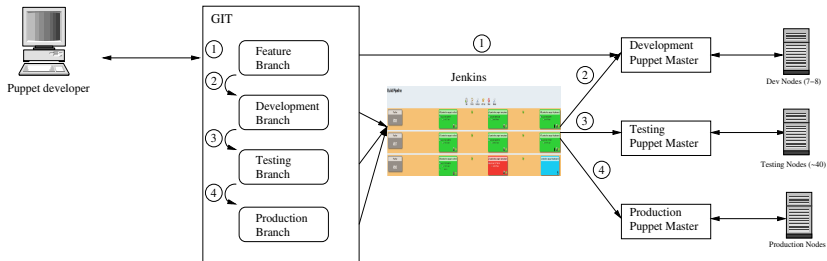
GIT

- ▶ **git push** triggers the deployment pipeline
- ▶ one central repository for internal modules
- ▶ gitolite for access control
- ▶ 3 main branches
 - ▶ development
 - ▶ testing
 - ▶ production
- ▶ feature branches for new site local modules
- ▶ hiera data is in the same repository

GIT repository layout

- ▶ `modules/`:
 - ▶ where r10k stores external (forge, github) modules
- ▶ `site/`:
 - ▶ site local modules, we do not want to share
- ▶ `hiera/`:
 - ▶ our hiera yaml files
- ▶ `Puppetfile`:
 - ▶ config file for r10k that specifies which external modules we need
- ▶ `Vagrantfile`:
 - ▶ To boot a development puppet environment on your local workstation

GIT workflow



- ① Features Branches get automatically created on Puppet Master (Dynamic Environments)
- ② Development Branch gets deployed on commit via Jenkins
- ③ Testing Branch gets deployed via GIT tag
pushing to testing triggers a deployment
- ④ Production Branch gets deployed via GIT tag
pushing to production triggers a deployment

It's all the same for Hiera yaml files!

- ▶ a tool to deploy puppet environments and modules
- ▶ every git branch gets deployed to a corresponding puppet environment
- ▶ in the current version (1.3.2) dependencies have to be managed manually

Example Puppetfile

```
1  forge 'forge.puppetlabs.com'

3  mod 'puppetlabs/ntp', '3.1.2'
   mod 'puppetlabs/postgresql', '3.4.2'
5  mod 'puppetlabs/stdlib', '4.3.2'
   mod 'puppetlabs/firewall', '1.1.3'
7  mod 'puppetlabs/apache', '1.1.1'
   mod 'puppetlabs/lvm', '0.3.2'
9  mod 'nosolutions/tsm', '0.2.2'
   mod 'saz/sudo', '3.0.6'
11 mod 'spiette/selinux', '0.5.4'

13 mod 'concat',
    :git => 'https://github.com/puppetlabs/puppetlabs-concat',
15   :commit => 'feba3096c99502219043b8161bde299ba65e7b8a'
```

You are able to pin to a git tag / branch / commit hash

a word on testing

- ▶ you must have unit tests for your puppet code: **rspec-puppet**
- ▶ for acceptance tests there's **puppetlabs/beaker**
- ▶ you need to test everything to get most out of the build pipeline
- ▶ we test
 - ▶ internal puppet modules
 - ▶ hiera data
 - ▶ puppet configuration
- ▶ all internal modules are required to have rspec tests

rspec-puppet example

samplemodule/manifests/init.pp

```
1 class samplemodule ( $message = 'defaultmessage' ) {  
    notify { 'samplemessage':  
3      message => "This is the sample module, my message is: $message",  
        }  
5 }
```

samplemodule/spec/classes/samplemodules_spec.rb

```
1 require 'spec_helper'  
  
3 describe 'samplemodule', :type => :class do  
    context 'with default parameters' do  
5      it { should contain_notify('samplemessage') }  
        end  
7    end
```


beaker example

```
1 require 'spec_helper_acceptance'
3 describe 'profiles::ossbase class' do
4   it 'should work with no errors' do
5     apply_manifest('include profiles::ossbase', :catch_failures => true)
6   end
7 end
```

DEMO

Do try this at home

- ▶ You need:
 - ▶ Vagrant from <http://vagrantup.com>
 - ▶ Virtualbox
 - ▶ Git client
- ▶ You have to run:
 - ▶ `git clone`
`https://github.com/tosmi/puppetcamp2014.git`
 - ▶ `cd puppetcamp2014`
 - ▶ `vagrant up`
 - ▶ `vagrant ssh`

Things we need to improve

- ▶ We need more test Systems (CentOS/RHEL/Solaris/AIX?)
- ▶ We need more acceptance tests
- ▶ Once again use stages in production
- ▶ Puppetlabs should package beaker as a rpm/deb whatever, gems suck in production

Summary

- ▶ Continuous Delivery is implemented via a deployment pipeline

Summary

- ▶ Continuous Delivery is implemented via a deployment pipeline
- ▶ Within the pipeline **everything** is automated

Summary

- ▶ Continuous Delivery is implemented via a deployment pipeline
- ▶ Within the pipeline **everything** is automated
- ▶ When everything is automated you need to test everything

Summary

- ▶ Continuous Delivery is implemented via a deployment pipeline
- ▶ Within the pipeline **everything** is automated
- ▶ When everything is automated you need to test everything
- ▶ Tools we are using to implement a deployment pipeline
 - ▶ Jenkins (CI)
 - ▶ GIT
 - ▶ Gitolite
 - ▶ r10k
 - ▶ rspec-puppet
 - ▶ puppetlabs/beaker

Thanks for your attention!