Puppet Fundamentals :
Session 2

Module Management
Classifying Nodes
Hiera
Design Patterns
Git
Module Development
Module Testing
Deploying Code
Master vs Masterless

Module Management : Index

- Introduction
- Structure
- Manual Creation
- The module command
- PuppetForge

Module Management : Introduction

Modules ...

- Contain all manifests (except for site.pp)
- Are single purpose
- Contain multiple functions
- Are self-contained (do not depend on other modules)
- Location is determined by the “*modulepath*” in puppet.conf

Module Creation methods :

- Manual creation
- Puppet module generate
- PuppetForge

Module Management : Structure

manifests	Contains all manifests
/init.pp	Contains the base class
/other_class.pp	Class my_module::other_class
/other_type	define mymodule::other_type
/subdir	A sub directory
/foo.pp	Class my_module::subdir::foo
files	Module filebucket
/my_file	A file for deployment
lib	Plugin directory, including custom facts
templates	Module templates
tests	Contains examples of declaring classes and types
spec	Spec tests for plug-ins

Module Management : Manual Creation

- Create your module directory under */etc/puppet/modules/*
- Create the minimum required files

  /manifests

  /metadata.json
- Create your manifests

Module Management : The module command

- “puppet module generate [name]-[module]”
  - Creates manifests and metadata.json file for you (and more)

- “puppet module list” : Show all installed modules
- “puppet module search” : Search PuppetForge for modules
- “puppet module install “ : Install a module from PuppetForge
- “puppet module uninstall” : Remove a module

Module Management : PuppetForge

- A library of public modules
- Variable quality
- Includes PuppetLabs own modules
- Includes other vendor modules
- Work-flow : search, install, modify, use

Workshop : Create a module

We are going to create the three types of module
and compare the differences

Create a module manually :

- Create the minimum module requirements : module name and manifests
- Check “puppet module list”

Create a module using “puppet module generate”

- “puppet module generate [user]-[modulename]”
- Check “puppet module list”

Create a module using “puppet module search / install”

- Search for an rsync module using “puppet module search”
- Install the module using “puppet module install”

- View the differences between the 3 types of file

Classifying Nodes

You've built your classes. Now lets decide where to use them.

Methods of Classification:

- site.pp (node statements)
- hiera_include
- External Node Classifiers (Foreman, Razor, Enterprise Dashboard)

Classifying Nodes : site.pp

Use Cases:

- Research clusters
- Testing / Proof of Concept

Modifying site.pp is:

- Easy to set-up
- Simple to test
- Inflexible

Classifying Nodes : site.pp

Absolute definitions :

```
node 'app-1': {
  include webserver
  If $operatingsystem == 'redhat' {
    include selinux
  }
}
```

Regex definitions :

```
node /.ttest.localdomain/ {
  include webserver-test
}
node /.pprod.localdomain/ {
  include webserver-prod
}
```

a default definition :

```
node default {
  include webserver-test
}
```

Workshop : Classify with “node”

We are going to limit which classes our nodes receives with the “node” function

Add a node definition

- Edit /etc/puppet/manifests/site.pp
- Wrap your class declaration in a node definition matching your nodes hostname
- Add a default clause
- Run the puppet agent (puppet agent -v -t) to check your setup

Classifying Nodes : hiera_include

/etc/puppet/manifests/site.pp

```
Exec { path => '/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin' }

File {
  owner => 'root',
  group => 'root',
  mode  => '0600',
}

hiera_include('classes')
```

/etc/puppet/hieradata/common.yaml

```
---
classes:
  - webserver
  - security
  - myusers
```

Workshop : Classify with “hiera_include”

We are going to limit which classes our nodes receives,
This time with the hiera_include function

Classify our nodes with hiera

- Edit /etc/puppet/manifests/site.pp
- Replace your node definitions with hiera_include('classes')
- Create an /etc/puppet/hieradata directory
- Copy the sample hiera.yaml file from the training repo
  - training-repo/hiera/hiera.yaml
  - To /etc/puppet/hiera.yaml
- link /etc/puppet/hiera.yaml to /etc/hiera.yaml
- Add your module-class name to */etc/puppet/hieradata/common.yaml*
- Run the puppet agent

Classifying Nodes : External Node Classifiers

What are they

- Any external application can be used to provide a list of classes to your nodes.
- These are ENCs, or External Node Classifiers

Examples

- Foreman
- Razor
- Spacewalk
- Cobbler

Hiera : Index

- YAML
- Introduction
- hiera.yaml
- Workshop : Populate the data hierarchy
- Priority Search
- Array Merging
- Hash Merging
- Workshop : Exploring Hiera
- Using Hiera for Classes
- Using Hiera for data
- Walkthrough usermgmt
- Workshop : Getting it to work

Hiera : YAML

YAML ain't markup language

- A human-readable data-serialization format
- Easily mapped to common data types

Simple array

```
---
clamav::maxscansize: '2000M'
clamav::tcpsocket:  '3310'
clamav::tcpaddr: '127.0.0.1'
```

Text block

```
---
graphite::storage_schemas_content: |
  [carbon]
  pattern = ^carbon\..*
```

Array list

```
---
groups: ['admin','dev']
```

Array list

```
---
deploy_jenkins_jobs::jobs:
  - deploy_microservice
  - deploy_puppet
  - deploy_app_config
```

Hiera : Introduction

What is it

- Hierarchical value store.
- Default backends of json or yaml
- Enables data separation – This is a good thing

Define variables

- Store complex values
- Merge values from across the whole database
- Types: String, Boolean, List, Hash

Apply classes

- Determined by the data hierarchy (hiera.yaml)
- The hierarchy uses facter
- Class your nodes by Environment, Role or Module (or anything else)

Hiera : hiera.yaml

An example hiera.yaml

```
---
:backends:
  - yaml
:yaml:
  :datadir: “/etc/puppet/hieradata”
:hierarchy:
  - "%{::environment}/%{::hostname}"
  - "%{::environment}/role.%{::role}"
  - "%{::environment}/mod.%{module_name}"
  - "%{::environment}/common"

  - "role.%{::role}"
  - "mod.%{module_name}"
  - common

:merge_behavior: deeper
```

Workshop : Populate the data hierarchy

We need to set-up some data in Hiera in order to use it.

Install the following modules

- puppetlabs-motd, puppetlabs-rsync

Install the following ruby gem

- “gem install deep_merge”

Add classes to hieradata/common.yaml

- Add content A to common.yaml
- run puppet agent -v -t
- Run “hiera motd::content”

Add classes to hieradata/test/common.yaml

- Add content B to common.yaml
- Run “hiera motd::content ::environment=test”

Add classes to hieradata/node/host-10-23-2-xx..yaml

- Add content C to host-10-23-1-xx.yaml

A : common.yaml

```
---
classes:
  - motd

motd::content: 'Content from common'
```

B : test/common.yaml

```
---
classes:
  - motd

motd::content: 'Content from test'
```

C : node/host-10-23-2-XX.yaml

```
---
classes:
  - motd

motd::content: 'Content from production'
```

Hiera : Priority Search

Using variables in your manifest: The Priority search

Variables are set to the **first** value they find in the hierarchy

hieradata/role_webcontent.yaml:

```
webcontent::ensure_page: 'present'
```

hieradata/webserver1.yaml:

```
webcontent::ensure_page: 'absent'
```

init.pp

```
class shuttering {
  file { '/usr/share/nginx/html/shutter_page.html':
    ensure => $ensure_page,
    owner => 'haproxy',
...
```

Hiera : Array merge

hiera_array ($myvalue)

An array is constructed from every matching value in the hierarchy.

It retrieves every string or array for a given key, then flattens them into a single array of values.

hieradata/role_webcontent.yaml:

```
webcontent::require_page: 'webcontent1'
```

hieradata/webserver1.yaml:

```
webcontent::require_page: 'webcontent2'
```

init.pp

```
...
file { '/usr/share/nginx/html/shutter_page.html':
  ensure => file
  owner => 'haproxy',
  group => 'haproxy',
  source => 'puppet:///modules/webcontent/shutter_page.html',
  require => File[$webcontent::require_page],
}
```

Hiera : Hash Merge

hiera_hash($myvalue)

A hash is constructed from every matching value in the hierarchy.

It retrieves every value for a given hash key.

/role_haproxy.yaml:

```
$haproxy::proxy_config:
  configa: “My general haproxy config goes here”
  configb: “More haproxy config goes here”
```

/production/role_haproxy.yaml:

```
$haproxy::proxy_config:
  configb: 'Additional config for production only goes here'
  configc: 'Additional config for production only goes here'
```

init.pp

```
file { '/etc/haproxy/haproxy.conf':
  ensure => present,
  content => $haproxy::proxy_config,
}
...
```

Hiera : Using Hiera for Classes

```
hiera_include('classes')
```

- hiera_include uses an array merge
- Every class found in the data hierarchy is used
- Decisions are based on Facts, and which areas of the data hierarchy they point to

Hiera : Using Hiera for data

- Variables specific to a module (e.g. apache::vhost: 'my-vhost')
  are made available to the relevant class
- Modern Paramaterized Classes are based on hieradata
- Priority search is used by default
- Specific tools provide the other styles of lookup

Workshop : usermgmt

A run-through of how the user module works

The usermgmt module...

- Is Parametrized
- uses Defined Types
- uses Templates

Workshop : usermgmt- Making it work

Create the correct hieradata that will allow two users to be created on your node

Install some required packages:

- yum install gcc-c++

Copy in the usermgmt module

- Copy the training-repo usermgmt module
  cp -r training-repo/examples/usermgmt /etc/puppet/modules/

Copy in the hieradata snippet

- cat usermgmt/hieradata/common-snippet.yaml >> /etc/puppet/hieradata/common.yaml
- Edit the common.yaml and check it for errors

- Run “hiera --hash usermgmt::userlist”

Design Pattern : Roles

hiera.yaml: - *"%{::environment}/role.%{::role}"*

A role is a discrete collection of Classes representing a distinct piece of server functionality.

It is a popular pattern due to its flexibility, and easy application.

/etc/facter/facts.d/base.yaml

```
environment: production
role: webserver
```

Assigning a role

- Custom or External facts

hieradata/qa/role-webserver.yaml

Components of a role:

- Either include your classes and variables
- Or include further subdivisions

Hieradata/role-webserver.yaml

```
---
Classes:
  - nginx
  - staticcontent
  - dnsmasq
  - dmz_server
```

Module Development : Using Git

Git is a distributed code-management tool following in the footsteps of Subversion and CVS.

- Fully Distributed
- Capable of integrating large chunks of independent code
- Well supported
- Use Web Services such as github or bitbucket
- Or use private repo's

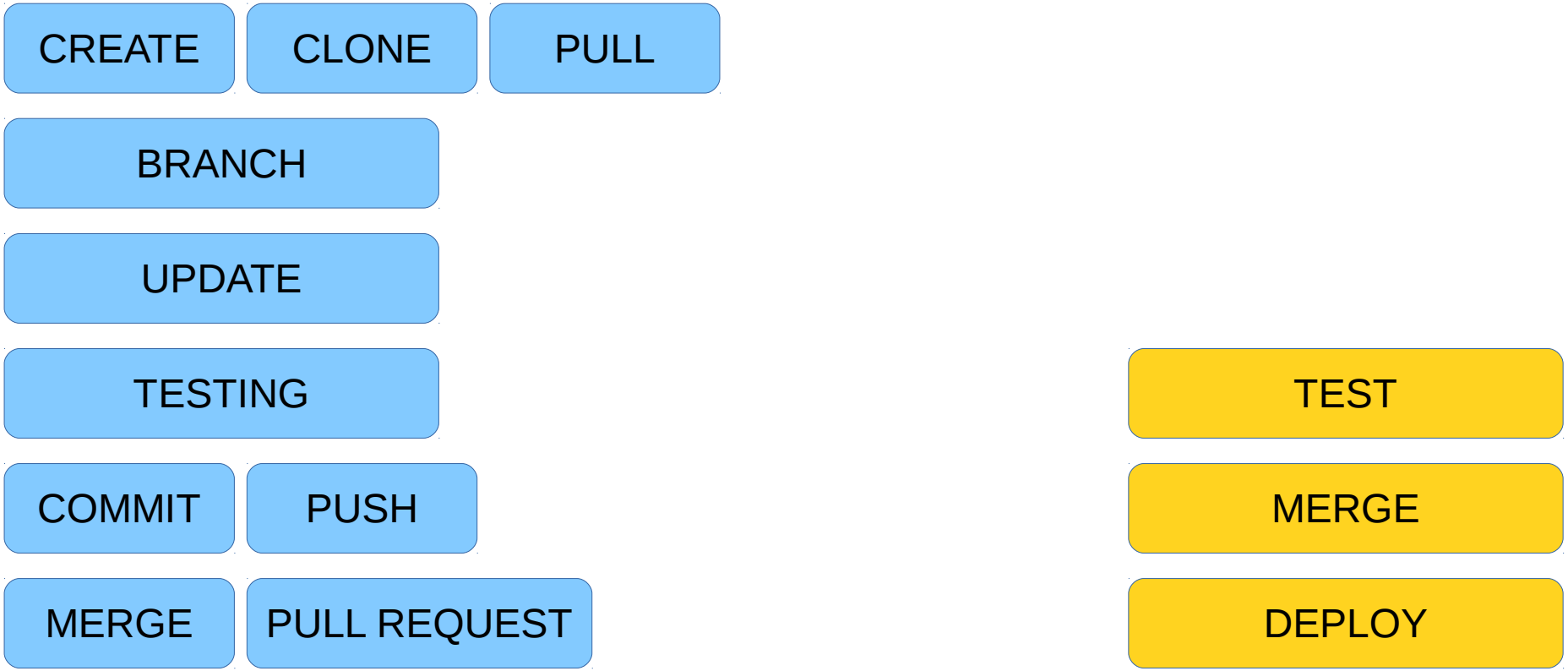
Module Development : Using Git

Commands

- git init — Create a blank git repo
- git clone — Clone an existing repo to your drive
- git pull — Refresh a repo from its origin repo
- git add — Stage changes for commit
- git commit — Commit changes, ready for a push
- git push — Push changes to the origin repo
- git merge — Merge branches
- git branch — Manage code branches
- git .....

Module Development : Using Git

Workflow :



Module Development : Testing

because there is no roll-back

As you code :

- puppet-lint
- puppet parser validate
- puppet apply [your.pp] -e “include [some_class]” --noop
- vagrant

On Commit (git hooks)

- .git/hooks/pre-push.sh

  “bundle exec rake”

Post-Commit :

- Non-production puppet-master
- Puppet environments
- --noop

Module Development : Deployment

Manually :

- Git pull
- Deployment engines (Jenkins, Travis)
- Deployment hooks

Puppet : Best Practise

Development :

- Parametrize your classes
- Keep data and modules separate
- Keep modules single purpose
- Use rspec and puppet-lint
- Don't re-invent the wheel
- Have a Testing methodology
- Use Pull Requests (Merging to Master is a Baconable offense)

Architecture :

- Commit
- High Availability : Multiple puppet-masters
- Puppet is not a file-server

Architecture : Master vs Master-less

Puppet-master :

- Provides a central point of control (and failure)
- Suitable for long-lived machines where configuration changes over time

Master-less :

- “puppet apply” is executed on every node according to
  - A deployment job
  - Cron
  - On boot
- Good for massively distributed or disposable environments