

Continuous Integration of Infrastructure

Nick Cammorato, TERC
June 12, 2013

What Prompted This?

- As more and more things became managed via configuration management, infrastructure management became more and more like software.
- What prompted configuration management is that the bulk of my time was ultimately spent coping with change.

Sources of Change



Internal

(Development, Technical Debt Load, Management, etc.)



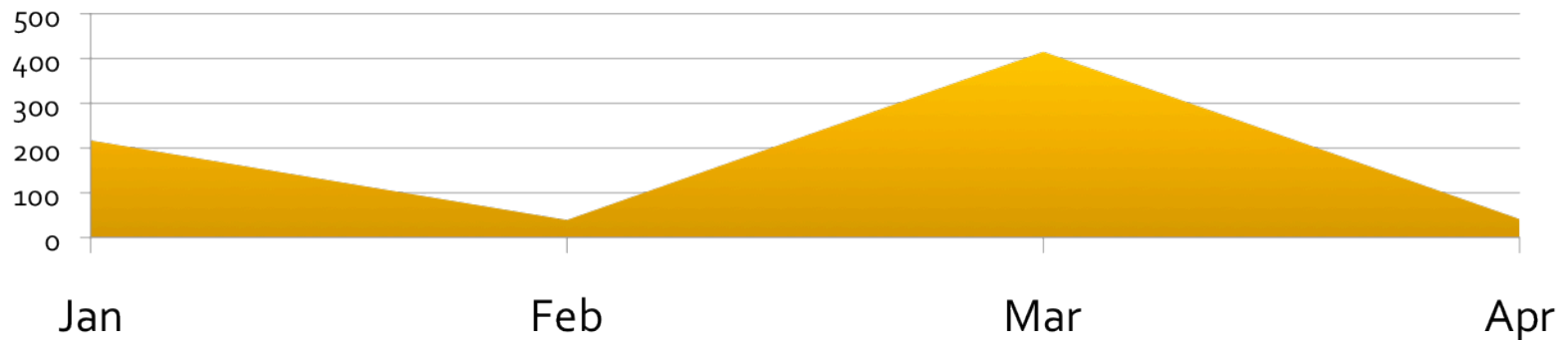
External

(Vendors, Security Issues, Regulatory Requirements, Hardware Failure)

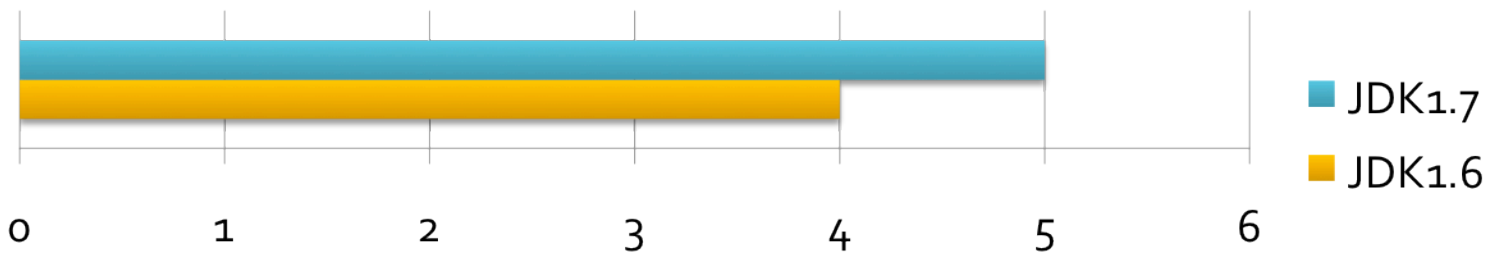
External Sources of Change

Q1, 2013

Enterprise Linux 5/6 Updates



JDK Updates



Coping with Change



- Each change should be tested
- Changes do not occur in isolation, each change likely triggers an integration problem.
- Manual testing is labor intensive
- Labor intensive means slow to do “the right way” which reduces the ability to respond quickly.
- Configuration management and change control processes help but we can do better.

What is an integration problem?

Imagine each of these is a jigsaw and the the nesting is a tower of hanoi puzzle
Where failure means explosions.



Continuous Integration

Update

- Developer Makes Changes
- Developer checks-in changes to source control

Integrate

- CI server triggers a job on commit, manually, or scheduled

Fast Build

- Builds can take a long time so sanity check them. (IE: Lint, build a subset, etc.)

Full Build

- Actually build the project for real.

Test

- Unit/regression/functional tests run.

Package

- Package built into a deliverable (rpm, dpkg, war, etc.)

Feedback

- Show the status of the build. Notify if necessary.

Deploy

- Operations problem now!

Components to integrate

- Configuration Data
 - Segmented from code in puppet via hiera or as of version 3.0 parameterized classes and an ENC
- Code which applies configuration data (IE: Puppet)
 - Puppet modules or Chef Recipes
- Upstream packages (Operating System, Internal)
 - Your yum or dpkg repositories

What do we use to integrate?

- The same thing development probably uses:
Jenkins - www.jenkins-ci.com
 - Jenkins has a rake plugin available
 - Rake can do anything within the ruby ecosystem and is task based.
 - This includes things like vagrant, rspec, etc.
 - Jenkins is trivially installed and managed (there are packages available for pretty much everything)

Prepping the Build

- Jenkins executes rake
- A rake task does a fast build/sanity check on a complete new config.
 - This should ignore package and configuration changes but trigger a full build.
- If this succeeds another rake task runs to spin up a VM environment via vagrant.
 - A full, full build is impractical unless you can have a complete duplicate of your environment. Our full build has to be a subset.

Dependency Resolution

- What can potentially impact what?
- If I update, say the apache package, I then obviously need to test websites which rely on apache.
 - To do this I may also need to spin up a database
 - And install additional software(IE: PHP, Tomcat, Java)
 - And a proxy
 - And...
- Most current options are bad for this, requiring much manual configuration(IE: ivy) and are basically solely dedicated to dependency resolution.

The Build

- With dependencies known we can build out a subset of infrastructure to test the specific changes in an intelligent way.
- The build itself is really just two config management runs
 - Applying the catalog we currently have
 - Introduces a requirement that all catalogs can cleanly apply to a base system
 - Then applying the new catalog

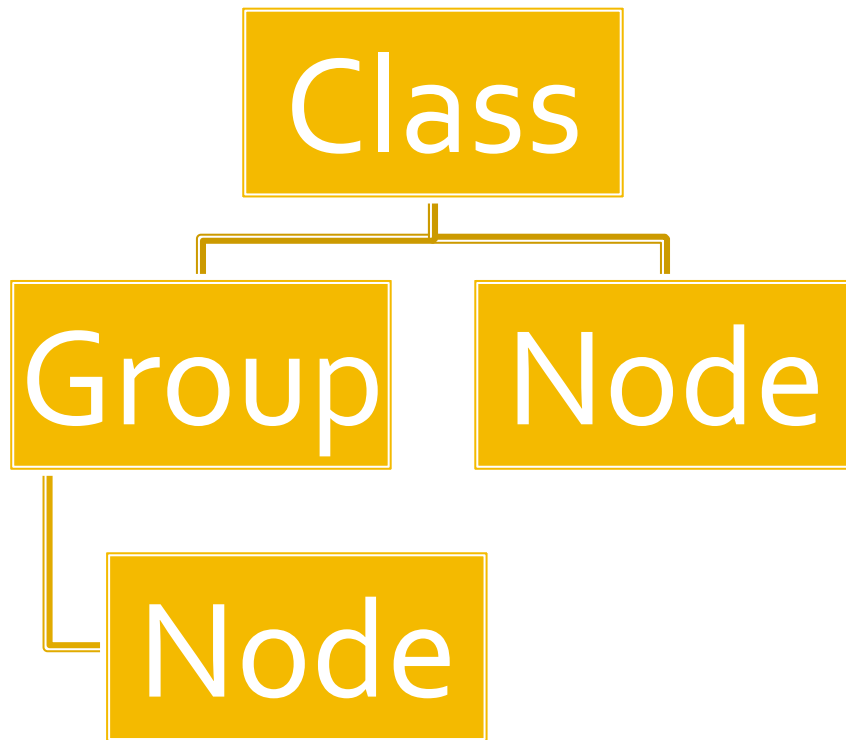
Testing

- Monitoring is effectively functional testing
 - Apache is supposed to be running, is it?
 - Is it responding on this port?
 - Do I get the right responses?
- As long as we deal with monitoring services in the service itself will be configured when the catalog is applied
 - So all we really need to do is invoke it via a rake task and parse the results

Requirements Met

- Configuration Management
- Continuous Integration Server
- The capability to virtualize test systems and provision them automatically
- Monitoring Software tied into configuration management
- Version Control
- Promotion Method

Current ENC Data Structure

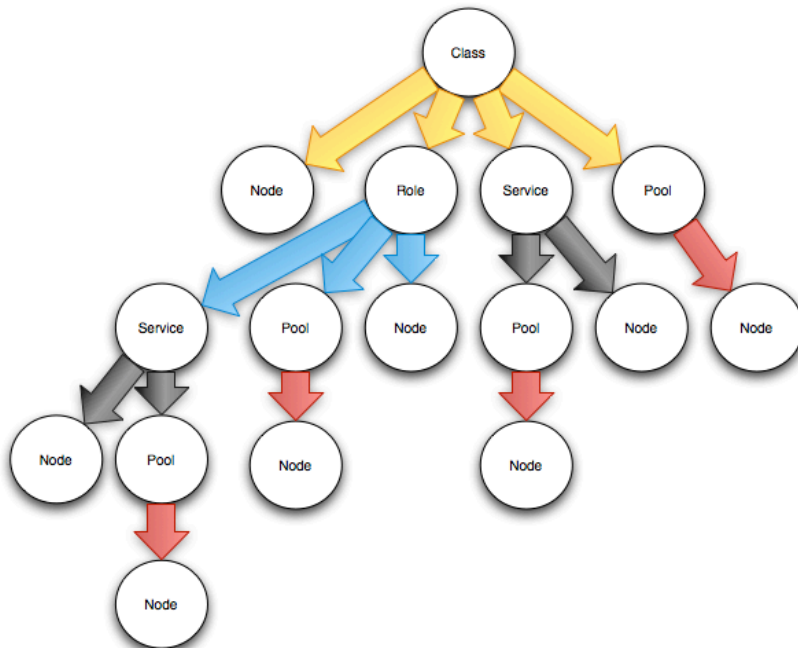


- Parameters are defined at each level
- Parameters inherit
- Parameters are sourced

Dependency Resolution

- Getting a list of nodes that classes or groups of classes have been applied to is trivial.
 - But an entire “service” rarely lives on single server or even a pool of servers.
 - We need a higher level object that links these things for dependency resolution.
- Simultaneously under-eager and over-eager
 - We probably don’t need to build out all 20 servers in an app pool – but they’ll share classes

An ENC to be Named Later



Class

- Assignable to all objects
- Corresponds to a puppet module

Node

- Assigned to service or pool
- Corresponds to a server

Pool

- Collection of nodes assignable to a service

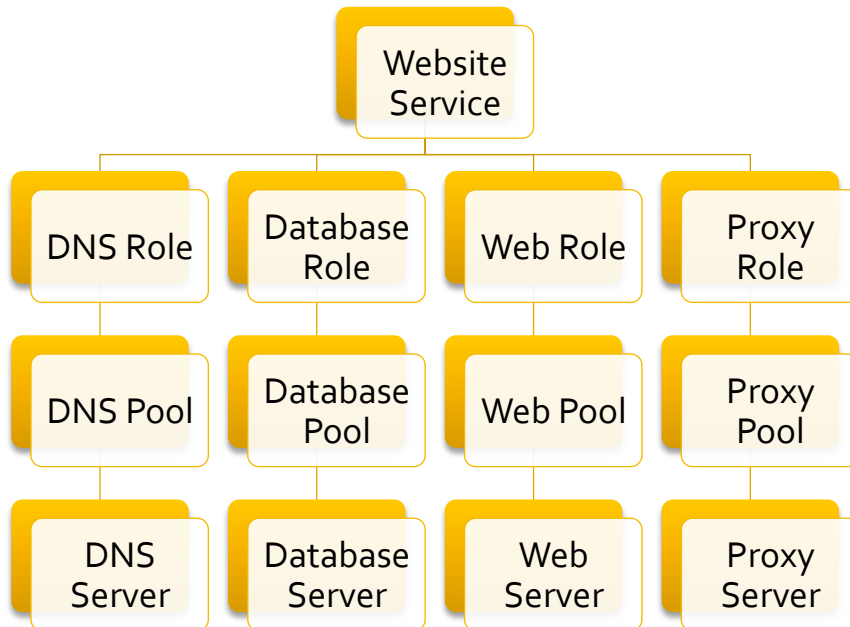
Role

- Collections of classes which may extend or override class parameters
- Assigned to nodes and services

Service

- Specific configuration linking roles/pools and classes/nodes

Example Service



- Service object consists of configuration specific to that service split by role (more on this later)
- Roles are collections of classes
- Pools are collections of servers with largely identical configuration.
- Pools or nodes are implicitly assigned to a Service

Advantages of new model

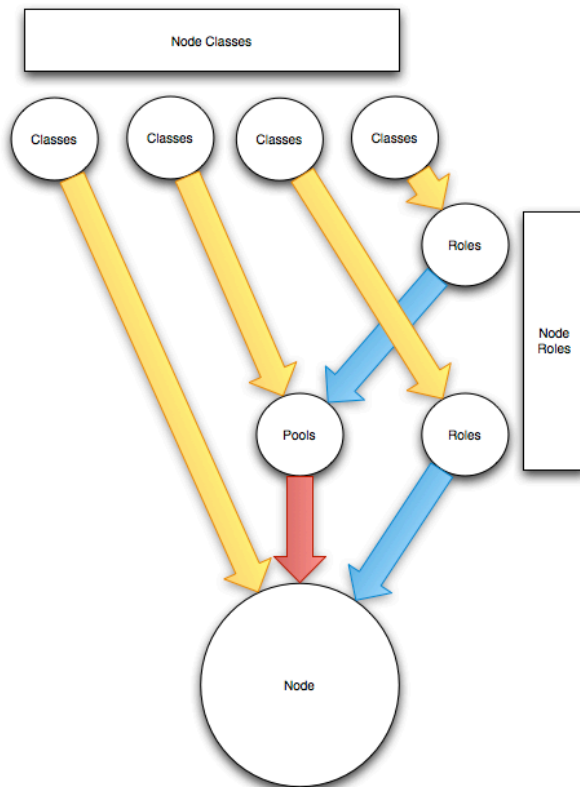
- Can retrieve dependent nodes at any level.
 - Through directly assigned roles or classes as in the current ENC
 - And indirectly through linked services
- Can deduplicate via pools
 - IE: If we have 12 database servers in a pool we can define that we only need 1 or 2.

A bit about Hieradata

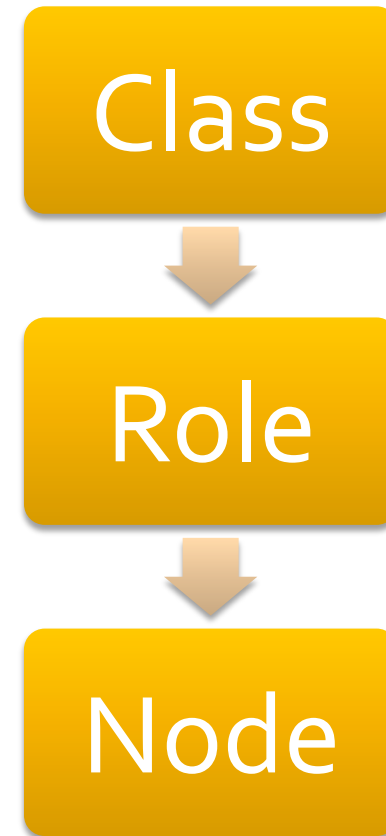
- The goal of hieradata is to keep site-specific data out of manifests.
 - Multiple levels of overrides
 - You define the overrides in sequence
 - Array joins and Hash merges
 - Reduces the need to edit code
 - Eases ability to share modules
- Avoids repetition

Node Inheritance

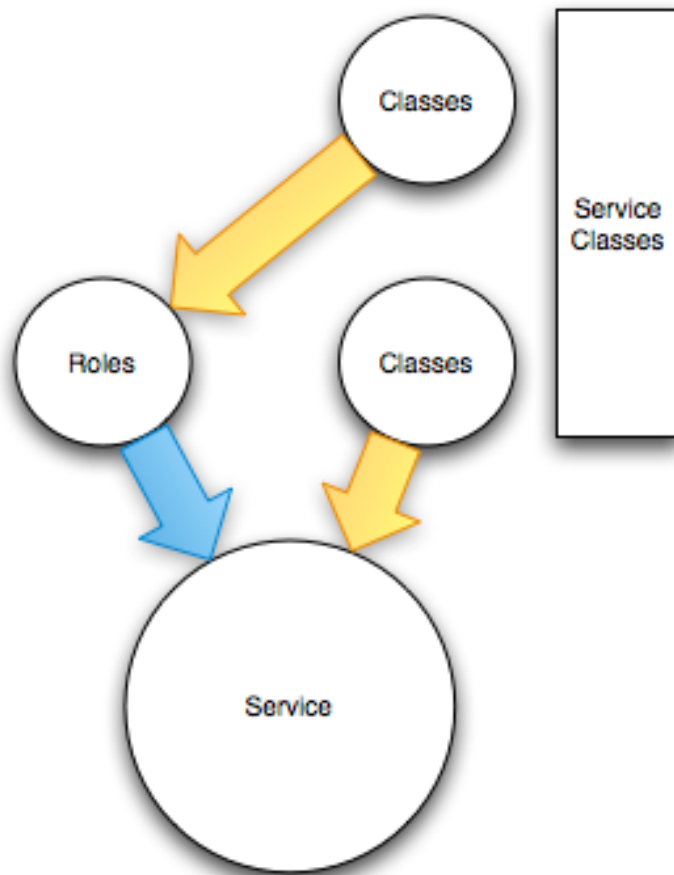
ASSOCIATION INHERITANCE



PARAMETER HIERARCHY



The Service Object



- Nodes and pools available for assignment are determined by service classes
- Configuration elements correspond to things like site configs, fleshares, dns entries, etc.
- It does maintain the node hierarchy as it does not inherit parameters.
- The point of a service from the perspective of hiera is to ADD data not override it.

The Service Object(continued)

Class

- configuration = {}

Class

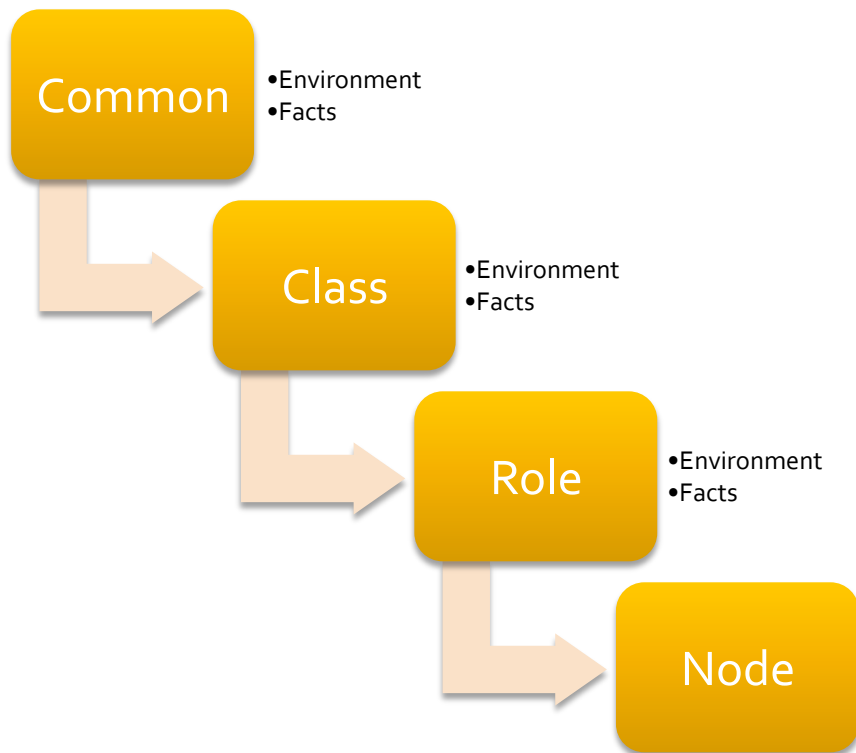
- configuration = {}

Class

- configuration = {}

- Configuration for each class that is assigned or inherited from assigned roles.
- Is namespaced to `${class}:service`

Configuration Hierarchy



- Services are collected last and joined as namespaced arrays of hashes.
 - `${class}:services => [{}]`

Continued Problems (the TODO list)

- Finish development of the ENC/Hiera backend
- The network equipment issue AKA we hit a brick wall in layer 3.
 - Lack of virtualization options means that we need another way to test networking configs
 - Idea currently under consideration is:
 - Translation table for the config and then realizing the config via virtual routers.

Further Reading

- Paul M. Duval, with Steve Matyas and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Boston: Addison-Wesley, 2007
- Jez Humble and David Farley. *Continuous Delivery: Reliable Software Release through Build, Test and Deployment Automation*. Boston: Addison-Wesley, 2010
- David Chelimsky, with Zach Dennis, Aslak Helleoy, Bryan Helmkamp and Dan North. *The RSpec Book: Behavior Driven Development with Rspec, Cucumber and Friends*. Raleigh: The Pragmatic Bookshelf, 2010
- James Turnbull and Jeffrey McCune. *Pro Puppet*. New York: Apress, 2011