PUPPET FUNDAMENTALS

RICH REMINGTON

rich.remington@gmail.com

LOGISTICS + ABOUT YOU (AND ME)

- Class time: 9-5
  Lunch ~11:45-12:45
  5-10 min. break per hour
- About you
  - Have you used Puppet before?
  - What is your role?
  - What do you want to get out of the course?
- About me

DAY 1 AGENDA

- Logistics/Agenda/Intro
- Why use configuration management?
- Why use Puppet?
- What is Puppet?
- Declarative vs. Imperative
- Resource Types
- Writing and applying manifests
- Resource Abstraction Layer (RAL)
- Classes
- Modules

DAY 2 AGENDA

- Variables
- Facts and **`facter`**
- Templates
- Master/agent configuration
- Node matching
- Puppet data flow
  - How Puppet works under the hood
- Functions
- Conditionals
- Class parameters

DAY 3 AGENDA

- Scope/Inheritance
- **`hiera`**
- Reporting
- Puppet Forge
- Roles and Profiles
- R10k (and Environments)
- External node classifiers
- RSpec
- Rouster

INTRODUCTION

HOW DID WE GET HERE?

- Typical office conversation
  CEO: "Faster!"
  CTO: "Cloud"
- "cloud" used to carry with it a positive, happy vibe because it promised to lift so many chores from our shoulders
- problem is that we're now responsible for dozens, hundreds, even thousands of machines
- the cloud made it easy to deploy them, but now we have to take care of them–configuration management

POSSIBLY FAMILIAR IT STORY

- you start with a single instance (e.g., a LAMP Stack)
- you decide to move the database to a separate server
- traffic starts to increase, so you add web servers
- IT tells you that since your application is so successful, you have to add an IT-accessible user to every server
  - imagine doing this with 3 servers
  - now imagine doing this with 1000+ servers

WHY USE CONFIGURATION MANAGEMENT?

- speed
- consistency
- repeatability
- reduction in human error
- increased effectiveness of IT personnel
- reduced cost in managing infrastructure
- makes your life easier!

WHY USE PUPPET?

- customize policies for each machine based on local data
- can handle OS-specific differences
- can be invoked w/specific tags
  - allows a filtered run that only performs operations matching those tags during a given invocation
- agents report back success, failure, and specific return codes for each resource, and entire run
- decentralized approach
  - each machine executes its own Puppet catalog separately
  - no machine is waiting for another to complete (cf. DIY)

FAVORITE THINGS ABOUT PUPPET

- can handle infinitely complex enforcement order
  - saves humans from having to worry about that cost unless it's necessary
  - other tools require a list of what to run in what order, which
    - imposes ordering on tasks that aren't necessarily ordered
    - litters source code with unnecessary info
- large (and active) community of users
  - IRC channel/mailing list = get help 24/7
  - puppetforge

OK, SO WHAT IS PUPPET?

- open-source IT automation tool
- Ruby-based programming language that provides a precise and adaptable way to describe a desired state for each machine
- you describe final state, Puppet does the work to get your machine(s) in that state, e.g.,
  - which users you want on your system
  - which packages you want installed
  - which services you want running
- it's declarative…

DECLARATIVE VS. IMPERATIVE

THINKING DECLARATIVELY

- shell scripts and typical programming languages are imperative
  - ...you tell them what to do, and HOW to do it
- in a declarative language you specify final states, not steps required to get there
- a common mistake is to attempt to use Puppet to make changes in an imperative manner
  - ...easier to handle change when you cast aside imperative programming

HANDLING CHANGE

- when you write imperative code to perform a sequence of actions, it works once, but second time…FAIL
- imagine creating a user on a Linux system…

```
[root@ip-192-168-168-33 ~]# useradd -u 2000 -g 10 -c 'Joe User' -m joe
[root@ip-192-168-168-33 ~]# useradd -u 2000 -g 10 -c 'Joe User' -m joe
useradd: user 'joe' already exists
```

- we could write some code to check if the user exists...

```
[root@ip-192-168-168-33 ~]# cat adduser-check
#!/bin/bash

getent passwd joe > /dev/null 2>&1

if test $? -ne 0; then
    sudo useradd -u 2000 -g 10 -c "Joe User" -m joe
else
    echo User joe already exists.
fi
[root@ip-192-168-168-33 ~]# ./adduser-check
[root@ip-192-168-168-33 ~]# ./adduser-check
User joe already exists.
```

but...this becomes unwieldy very quickly!

IDEMPOTENCE

- We want operations to be idempotent, i.e., the operation achieves the same results every time it executes

```
admin_user = 'admin'
echo hello > filename
```

vs.

```
admin_user = get_user()
echo hello >> filename
```

- in order for a configuration state to be achieved no matter the conditions, it is essential that the configuration language avoid describing the actions involved to achieve the desired state.
- instead, the configuration language describes the desired state, and leaves the actions up to the interpreter

DECLARATIVE, NOT IMPERATIVE

```
user { 'joe':
  ensure     => present,
  uid        => '2000',
  gid        => '10',
  comment    => 'Joe User',
  managehome => true,
}
```

- the above is a Puppet resource declaration for the user **joe**
- ensure user joe is present, has **uid = 2000**, **gid = 10**, name "Joe User" and that home dir should be created…but it doesn't say HOW to do these things
- in a moment we'll use Puppet to try this out

WHY USE PUPPET? (REDUX)

- why not just write a shell script?
  - if you have only a few commands and a few machines, shell scripts are probably the way to go
  - …but shell scripts don't scale
    - what if you have 1000 machines? 10,000?
  - shell scripts require a lot of error checking/handling because you are writing imperative code (do this, check that) as opposed to declarative (the final state must be this)

LET'S TRY PUPPET!

LOG IN TO YOUR SANDBOX

- get IP address from your instructor
- from your Mac, you will ssh into your AWS instance
  - launch Terminal (from Finder, use the Go menu and choose Utilities)
  - once you've launched Terminal, you can type

    `ssh student@<ip-address>`
- once logged in, type

  `sudo useradd -u 2000 -g 10 -c "joe user" -m joe`

  (hit up arrow key and try again)

  `sudo userdel joe`

  (hit up arrow key twice and try the useradd again)

INSTALLING PUPPET AGENT

- On your AWS instance, type

```
yum install -y puppet-agent
puppet --version
  (should report 3.8.7)
which puppet
```

RESOURCES

RESOURCES

- fundamental unit for modeling system configurations
- smallest building block of Puppet language
  - singular element you wish to evaluate/create/remove
- Puppet comes w/many builtin resources to manipulate system components you are already familiar with, e.g.,
  - users
  - groups
  - files
  - packages
- block of Puppet code that describes a resource is called a resource declaration

RESOURCES CAN BE COMBINED...

- ...to represent the desired state of your system

RESOURCES (CONT'D)

```
user { 'dws':
     ensure => present,
     uid    => '1005',
     gid    => '10',
     shell  => '/bin/bash',
     home   => '/home/dws',
}
```

- curly braces define the resource block
- title separated from the body with a colon
- body consists of attribute/value pairs

RESOURCES (CONT'D)

```
user { 'dws':
     ensure => present,
     uid    => '1005',
     gid    => '10',
     shell  => '/bin/bash',
     home   => '/home/dws',
}
```

- type plus title must be unique for a given node
- quote numeric values (not required in Puppet 3)
- Best practice: include comma after last attribute

HELLO, WORLD!

```
notify { 'greeting':
    message => 'Hello, world!',
}
```

- title = greeting
- message = attribute
- key/value pairs (like Python, Perl, etc.)
- => "hashrocket" (also called "fat comma")
- a fully functional manifest (a file containing Puppet code)
- let's use Puppet to implement this manifest
  (note: we don't "run" the manifest)

LAB: WRITING MANIFESTS

- create a directory to hold our manifests and work from:

```
mkdir manifests
cd manifests
```

- use an editor to create a manifest named **hello.pp** with the following contents (note: vim, nano, and emacs are installed)

```
notify { 'greeting':
    message => 'Hello, world!',
}
```

- implement/apply the manifest by typing

```
puppet apply hello.pp
```

LAB: WRITING MANIFESTS

- use an editor to create a manifest named joe.pp

```
user { 'joe':
  ensure     => present,
  uid        => '2000',
  gid        => '10',
  comment    => 'Joe User',
  managehome => true,
}
```

- implement/apply your manifest (as root…)
- check the syntax of your manifest

```
puppet parser validate joe.pp
```

THERE CAN BE ONLY ONE

- within a manifest or set of manifests being applied together (the catalog for a node) a resource of a given type can only be declared with a given title once

```
user { 'rocco':
  ensure => present,
  uid        => '1002',
  managehome => true,
}

user { 'rocco':
  ensure => present,
  uid        => '1003',
  managehome => false,
}
```

```
notify { 'rocco':
  message => "User rocco present",
}
```

Can't have two user resources with same name

This one's OK because it's a different resource type

RESOURCES (CONT'D)

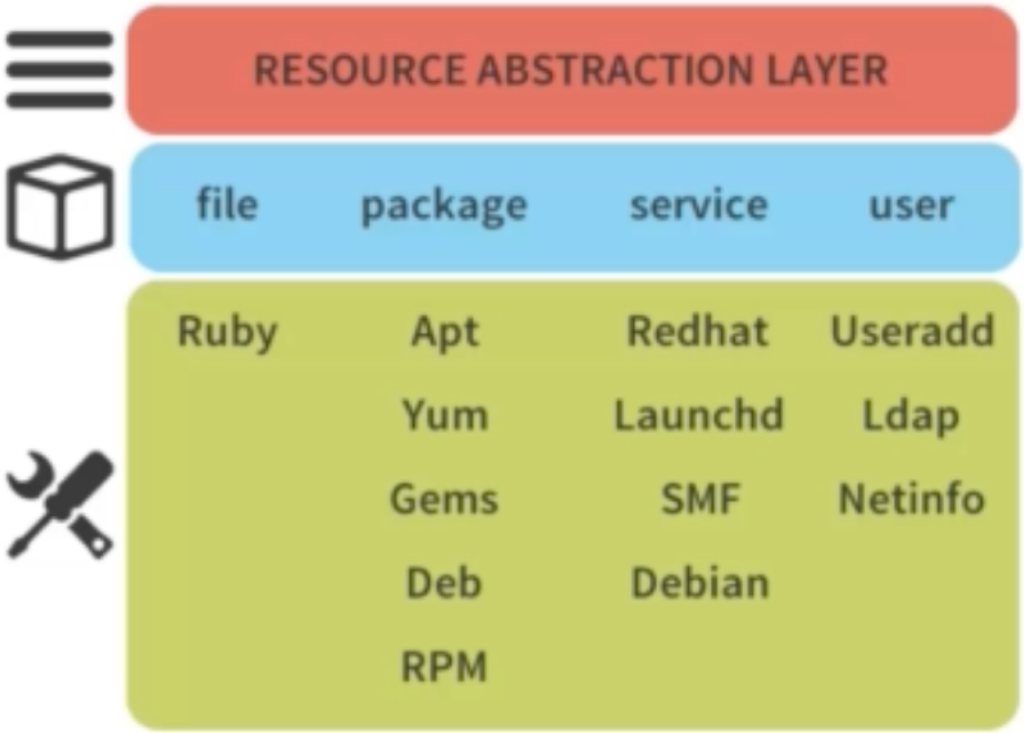
- resources are conceptually identical across platforms
- description of resource can be abstracted away from its implementation

- these two insights form Puppet's Resource Abstraction Layer…

RESOURCE ABSTRACTION LAYER

RESOURCE ABSTRACTION LAYER (RAL)

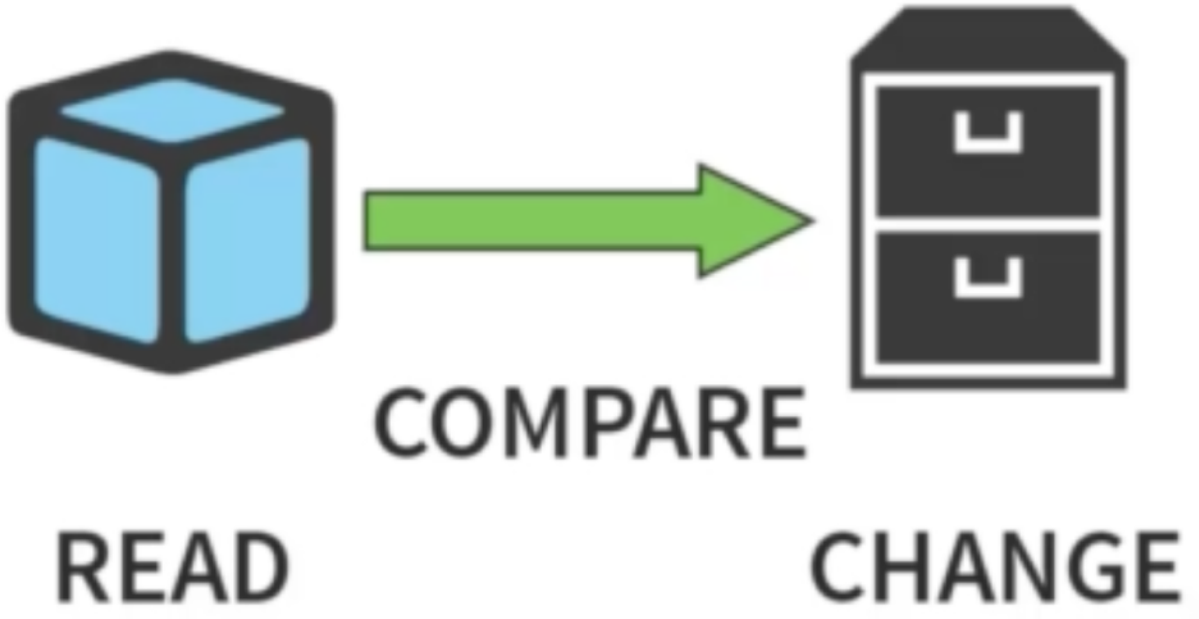
- RAL splits resources into types and providers



- providers are the interface between the underlying OS and the resource types

RESOURCE ABSTRACTION LAYER (CONT'D)

- Puppet uses the RAL to read/modify the state of resources on the system
- to sync a resource, Puppet uses RAL to compare the "is value" to the "should value"



PUPPET RESOURCE

- command-line tool for inspecting resources on your system
- interacts directly with RAL to get the job done

  `puppet resource type title` (query a specific resource)

  `puppet resource type` (query all resources of specified type)

```
[root@ip-192-168-168-33 ~]# puppet resource user root
user { 'root':
  ensure           => 'present',
  comment          => 'root',
  gid              => '0',
  home             => '/root',
  password         => '$1$duYHAQx9$4cEeW1GisGKZGi7SY9Ipw/',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '0',
}
```

LAB: PUPPET RESOURCE

- use **`puppet resource mailalias postmaster`** to get Puppet code for the postmaster mail alias
- put that code into a manifest and change the recipient to your user (**`student`**)
- apply your manifest
- take a look at **`/etc/aliases`** to be sure your manifest was applied properly (**`less /etc/aliases`**)
- note that you must use **`sudo`** or already be running as **`root`** if you want to make changes to system resources (or view them)

IMPLEMENTING A MANIFEST

- What does it mean to implement a manifest? Puppet will…
  1. use dependency information (if supplied) to determine which resources should be handled first
  2. compile manifest into a Puppet catalog (JSON)
  3. evaluate each resource to determine if changes are necessary
  4. create, modify, or remove the resource, if need be
  5. provide verbose feedback about each resource, i.e., whether it changed and what was done to change it
  6. log/text/slack/IRC/Twitter/Splunk/etc. results

MANAGING FILES

- the **`file`** resource allows us to manage files

```
file { '/tmp/testfile.txt':
  ensure  => present,
  mode    => '0644',
  replace => true,
  content => 'wow cow!',
}
```

- file should exist and have specific contents
- we do not need to concern ourselves with how, or when to make changes to the file
- what is the purpose of the **`replace`** attribute?

MANAGING FILES (CONT'D)

- let's take a look at that file…

```
less /tmp/testfile.txt
cat /tmp/testfile.txt
od -a /tmp/testfile.txt
```

- let's add a trailing newline character and take a look…

```
file { '/tmp/testfile.txt':
  ensure  => present,
  mode    => '0644',
  replace => true,
  content => 'wow cow!\n',
}
```

MANAGING FILES (CONT'D)

- the newline (\n) is interpreted, so we need to use double quotes
- single quoted strings prevent any interpretation of what's inside the quotes, so we get a literal '\' followed by 'n', which is of course not what we want

```
file { '/tmp/testfile.txt':
  ensure  => present,
  mode    => '0644',
  replace => true,
  content => "wow cow!\n",
}
```

LAB: MANAGING FILES

- use **`puppet resource`** to create a new manifest from the file **`/tmp/testfile.txt`**
- change the file's permissions (mode) to be group writable (either **`0664`** or **`ug=rw,o=r`**)
- change content
- **`puppet apply`**
- confirm changes using **`ls -la`** and **`cat/more/less`**
- **`puppet apply`** again and see what happens

LAB: HOST RESOURCES

- look up **`host`** resources in the Puppet docs
- create a host resource for a host **`foobar.com`** at IP address **`1.2.3.4`**, with an alias of **`fooserver`**
- at a minimum your host resource should contain the following attributes: **`ip`**, **`host_aliases`**, and **`comment`**
- apply your manifest and verify changes to **`/etc/hosts`**

SOME CORE RESOURCE TYPES

- **`notify`**
- **`file`**
- **`user`**
- **`group`**
- **`package`**
- **`service`**
- **`exec`**
- **`cron/scheduled_task`**
- vs. resource types that are not core types
  - some won't work on all machines, e.g., **`mysql_user`**

PUPPET CHEAT SHEET

puppet labs

Core Types Cheat Sheet

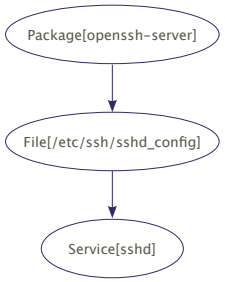
THE TRIFECTA

Package/file/service: Learn it, live it, love it. If you can only do this, you can still do a lot.

```
package { 'openssh-server':
    ensure => installed,
}

file { '/etc/ssh/sshd_config':
    source  => 'puppet:///modules/sshd/
sshd_config',
    owner   => 'root',
    group   => 'root',
    mode    => '640',
    notify  => Service['sshd'], # sshd
               will restart whenever you
               edit this file.
    require => Package['openssh-server'],
}

service { 'sshd':
    ensure => running,
    enable => true,
    hasstatus => true,
    hasrestart => true,
}
```



file

Manages local files.

ATTRIBUTES

- `ensure` — Whether the file should exist, and what it should be.
  - present
  - absent
  - file
  - directory
  - link
- `path` — The fully qualified path to the file; **defaults to title.**
- `source` — Where to download the file. A puppet:/// URL to a file on the master, or a path to a local file on the agent.
- `content` — A string with the file's desired contents. Most useful when paired with templates, but you can also use the output of the file function.
- `target` — The symlink target. (When ensure => link.)
- `recurse` — Whether to recursively manage the directory. (When ensure => directory.)
  - true or false
- `purge` — Whether to keep unmanaged files out of the directory. (When recurse => true.)
  - true or false
- `owner` — By name or UID.
- `group` — By name or GID.
- `mode` — Must be specified exactly. Does the right thing for directories.
- See also: `backup, checksum, force, ignore, links, provider, recurselimit, replace, selrange, selrole, seltype, seluser, sourceselect, type.`

package

Manages software packages. Some platforms have better package tools than others, so you'll have to do some research on yours; check the type reference for more info.

ATTRIBUTES

- `ensure` — The state for this package.
  - present
  - latest
  - {any version string}
  - absent
  - purged (Potentially dangerous. Ensures absent, then zaps configuration files and dependencies, including those that other packages depend on. Provider-dependent.)
- `name` — The name of the package, as known to your packaging system; **defaults to title.**
- `source` — Where to obtain the package, if your system's packaging tools don't use a repository.
- See also: `adminfile, allowcdrom, category, configfiles, description, flavor, instance, platform, provider, responsefile, root, status, type, vendor.`

service

Manages services running on the node. Like with packages, some platforms have better tools than others, so read up. To restart a service whenever a file changes, subscribe to the file or have the file notify the service. `(subscribe => File['sshd_config'] or notify => Service['sshd'])`

ATTRIBUTES

- `ensure` — The desired status of the service.
  - running (or true)
  - stopped (or false)
- `enable` — Whether the service should start on boot. Doesn't work everywhere.
  - true or false
- `name` — The name of the service to run; **defaults to title.**
- `status, start, stop, and restart` — Manually specified commands for working around bad init scripts.
- `hasrestart` — Whether to use the init script's restart command instead of stop+start. Defaults to false.
  - true or false
- `hasstatus` — Whether to use the init script's status command instead of grepping the process table. Defaults to false.
  - true or false
- `pattern` — A regular expression to use when grepping the process table. Defaults to the name of the service.
- See also: `binary, control, manifest, path, provider.`

PUPPET-LINT

- **`puppet-lint`** is a tool that checks your manifests to ensure they conform to the Puppet style guide
- we can install puppet-lint using Ruby's package manager, gem: **`gem install puppet-lint`**

```
file { '/tmp/testfile.txt':
 <TAB> ensure  => file,
     mode => 'ug=rw,o=r',
   replace => true,
   content => 'holy cow!',
}
```

LAB: PACKAGE RESOURCE

- use a **`package`** resource (as opposed to a file resource) to install **`puppet-lint`** on your instance
- ensure the latest version of **`puppet-lint`** will be installed
- the provider, as we saw on the previous slide, is **`gem`**
  - …if you've already installed **`puppet-lint`** using **`gem`**, you might want to uninstall it before testing your manifest

RECAP: PUPPET RESOURCES

- resource = smallest building block of Puppet code
- Puppet code describing a resource is a resource declaration
- syntax: key-value pairs inside {}s
- core types we used: **`notify/file/package/user/group`**
- remember Puppet docs!

CLASSES

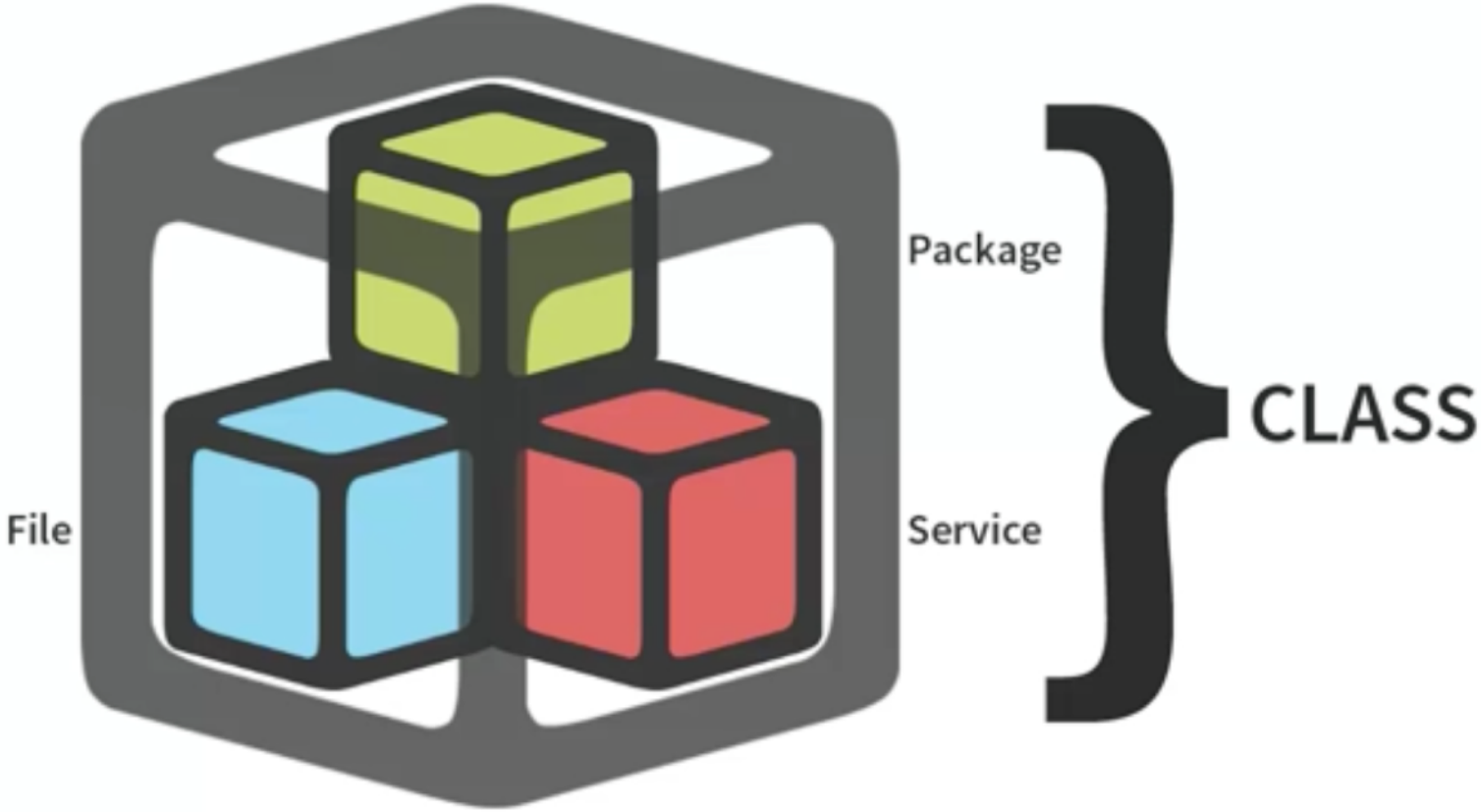
CLASSES

- define a collection of resources that are managed together as a single unit
- defining a class makes it available, but does not declare it (i.e., it isn't used/applied)
- classes are singletons (vs. resources, where there are many)

CLASS EXAMPLE

```
class apache {
  package { 'httpd':
    ensure => present,
  }
  file { '/etc/httpd/conf/httpd.conf':
    ensure => file,
    owner  => 'root',
    group  => 'root',
    mode   => '0644',
    source => 'puppet:///modules/apache/httpd.conf',
  }
  service { 'httpd':
    ensure => running,
  }
}
```

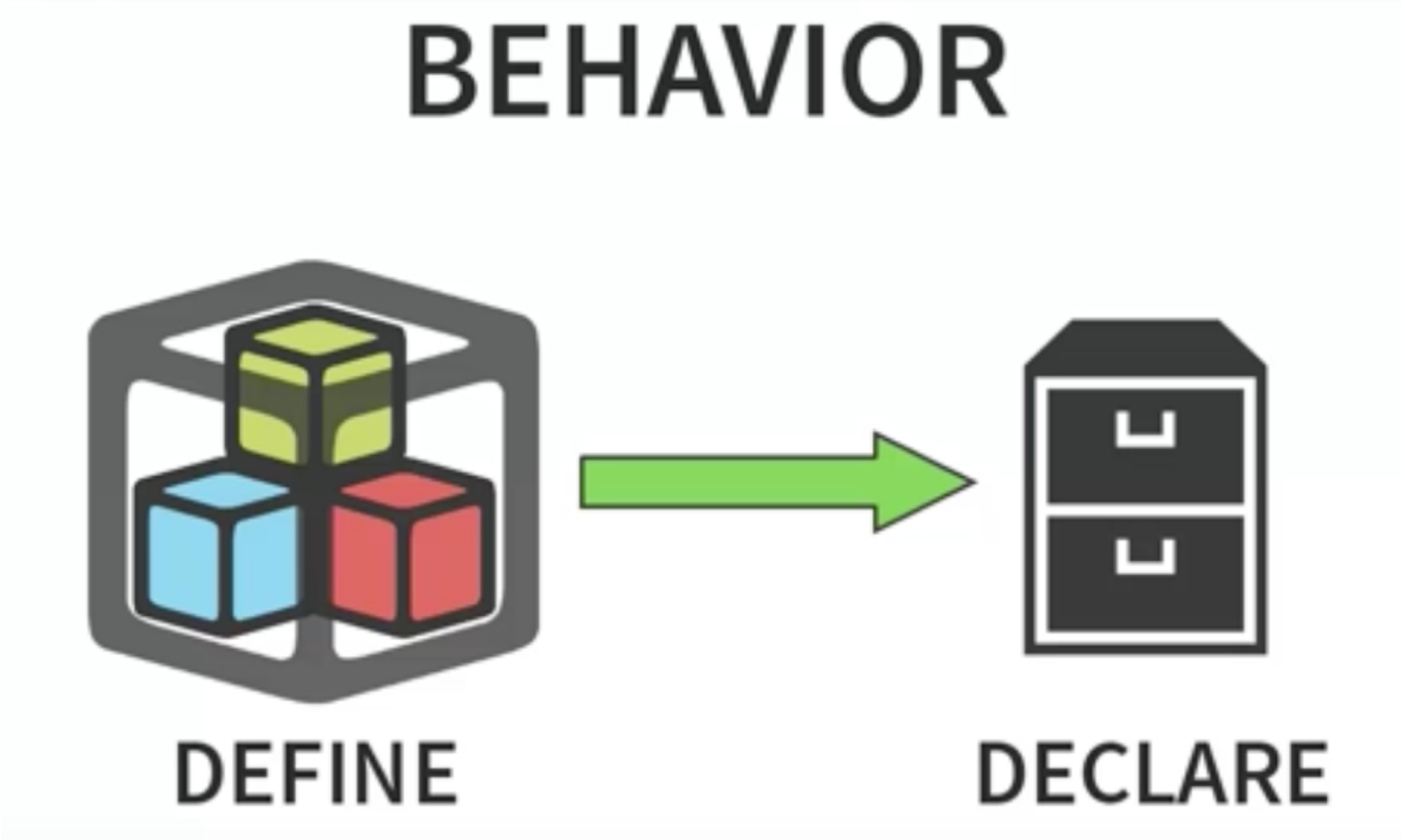
CLASSES (CONT'D)



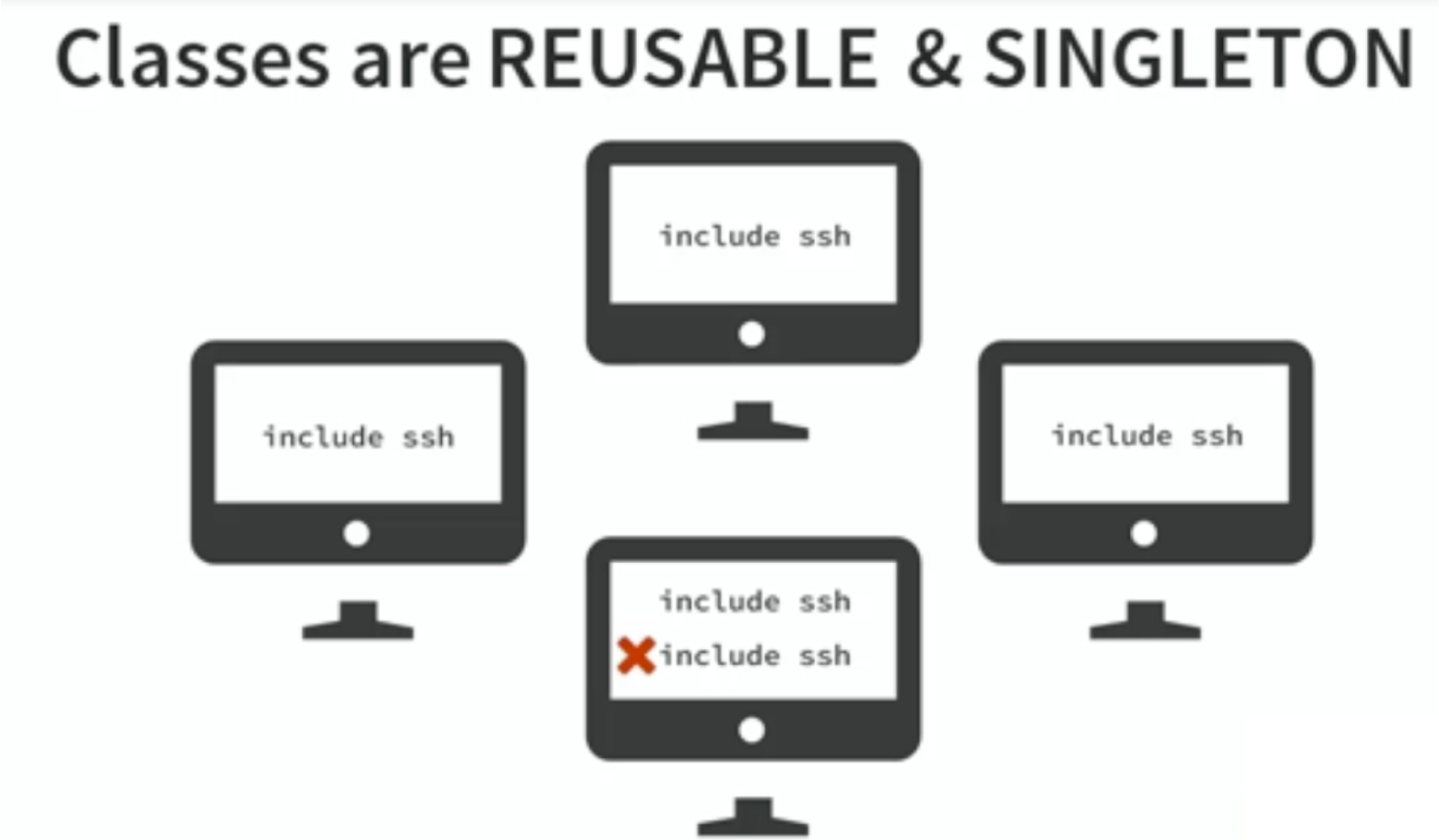
!! DAILY DOUBLE !!

- Why use Configuration Management?
- What is Puppet?
- What is a resource?
- What is RAL and how does it help us?

CLASSES (CONT'D)



CLASSES (CONT'D)



PUPPET OBJECT HIERARCHY

RECAP: CLASSES

- classes = collection of resources managed together
- class definition vs. class declaration
- reusable
- singleton
- object hierarchy: resource / class / manifest

MODULES

MODULES

- modules are self-contained bundles of code and data
- all Puppet manifests belong in modules (except **`site.pp`**, which we will talk about tomorrow)
- you can download pre-built modules from the Puppet Forge or you can write your own
- the only way Puppet can find (and load) classes is if they are in modules
- so what is a module…?

MODULES (CONT'D)

- a directory tree with a predictable structure:

```
/etc/puppet/modules/<MODULE NAME>/
                                  manifests/
                                            init.pp
                                      files/
                                  templates/
                                        lib/
                                    facts.d/
                                   examples/
                                       spec/
```

- **`init.pp`** contains the class definition
- class name must match module name
- try **`puppet config print modulepath`**

LAB: CREATE A MODULE

1. create a manifest called **`users.pp`** which contains two user resources and a group resource, i.e., let's create two users, both of whom are in a group called supercool
2. apply your manifest to be sure it works
3. use **`userdel`** (or **`puppet`**) to remove one of the users
4. verify user has been removed using **`id`** command
5. make your code into a class by surrounding it with

```
class users {
  …
}
```

6. apply your manifest
7. verify that it did nothing–why?

LAB: CREATE A MODULE (CONT'D)

8. **`mkdir -p /etc/puppet/modules/users/manifests`**
   `mkdir /etc/puppet/modules/users/examples`
   `cp users.pp /etc/puppet/modules/users/manifests/init.pp`
9. **`puppet apply /etc/puppet/modules/users/manifests/init.pp`**
   (should do nothing)
10. create **`/etc/puppet/modules/users/examples/init.pp`** and put **`include users`** in it
11. **`cp /etc/puppet/modules/users/examples/init.pp ./users.pp`**
12. **`puppet apply users.pp`**
13. verify that user who was deleted previously has now been created

PACKAGE-FILE-SERVICE

- what does a sysadmin do?
  - install and configure software
- a pattern in Puppet has evolved to serve the sysadmin who typically
  - installs a package
  - add data to configure that software (e.g., **`/etc`**)
  - start the service
- common pattern: a package *requires* a *file*, such as a config file, and it needs to be running as a *service*
- sometimes called the Puppet "trifecta"

PACKAGE-FILE-SERVICE (CONT'D)

```
class apache {
  package { 'httpd':
    ensure => present,
  }
  file { '/etc/httpd/conf/httpd.conf':
    ensure => file,
    owner  => 'root',
    group  => 'root',
    mode   => '0644',
    source => 'puppet:///modules/apache/httpd.conf',
  }
  service { 'httpd':
    ensure => running,
  }
}
```

LAB: PACKAGE-FILE-SERVICE

- create your own **`ntp`** module (even though it exists in the Puppet Forge)
- create an **`ntp.conf`** file using **`time.apple.com`** as the timeserver
- use **`puppet apply`** to invoke the ntp module
- ensure the **`ntpd`** service is running

ORDERING?

- by default, Puppet 3 uses title hash ordering
- computes a SHA-1 hash for each resource title, then sorts them, and applies them in that order
- Puppet 4 applies resources in the order they're declared in the manifest
- in **`puppet.conf`** the default ordering method can be changed to *random* for testing purposes
- if a group of resources must always be managed in a specific order, the relationships should always be declared with *metaparameters*

BEFORE/REQUIRE

- the **before** metaparameter states that one resource must be applied before another

```
package { 'httpd':
  ensure => present,
  before => Service['httpd'],
}
```

- the same relationship can be expressed using the **require** metaparameter

```
service { 'httpd':
    ensure  => running,
    require => Package['httpd'],
}
```

NOTIFY/SUBSCRIBE

- the **notify** metaparameter is used to express a dependency

```
file { '/etc/httpd/conf/httpd.conf':
  ensure => file,
  ...
  notify => Service['httpd'],
}
```

- or instead use a **subscribe** metaparameter

```
service { 'httpd':
    ensure    => running,
    subscribe => File['/etc/httpd/conf/httpd.conf'],
}
```

MORE ORDERING

- multiple dependencies can be expressed using an array

```
service { 'httpd':
    ensure => running,
    require => [
        Package['httpd'],
        File['/etc/httpd/conf/httpd.conf'],
    ],
  }
}
```

- (or we could have used the before metaparameter)

CHAINING ARROWS

```
# ntp.conf is applied first, and notifies the ntpd service if it changes:
File['/etc/ntp.conf'] ~> Service['ntpd']
```

```
Package['ntp'] -> File['/etc/ntp.conf'] ~> Service['ntpd']
```

```
# first:
package { 'openssh-server':
  ensure => present,
} -> # and then:
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => '0600',
  source => 'puppet:///modules/sshd/sshd_config',
} ~> # and then:
service { 'sshd':
  ensure => running,
  enable => true,
}
```

!! DAILY DOUBLE !!

- What is a Puppet manifest?
- What is a class and where do define them?
- Ok, about those modules … What's their main purpose?
- How do we get Puppet to tell us what it knows about a resource - e.g. a user or host?
- How do we tell Puppet about dependencies between resources?

LAB: ORDERING

- modify your **`ntp`** class to implement ordering
- change the configuration file in your module to add a comment at the top which says

```
# This file is managed by Puppet
```

- apply the example manifest from the new module and make sure it is working as expected

RECAP: MODULES

- self-contained bundles of code and data
- all Puppet manifests belong in modules (except **`site.pp`**)
- Puppet Forge vs. write your own
- only way to find/load classes is in modules
- package–file–service
- ordering

RECAP: DAY 1

- why use Puppet?
- what is Puppet?
- thinking declaratively
- resources
- classes
- modules

DAY 2 AGENDA

- Variables
- Facts and **`facter`**
- Templates
- Master/agent configuration
- Node matching
- Puppet data flow
  - How Puppet works under the hood
- Functions
- Conditionals
- Class parameters

VARIABLES

VARIABLES

- prefaced with a $
- variable names may contain upper and lower case letters, numbers, and underscores
- style guide recommendation: don't use upper case letters (must begin with a lower case letter in Puppet 4)

```
$myvar          # valid
$MyVar          # invalid in Puppet 4

$my_var         # valid
$my-var         # invalid

$my3numbers     # valid
$3numbers       # invalid
```

VARIABLES (CONT'D)

- assigned with an '='
- values can be boolean, numbers, strings, arrays, hashes, or "undef"

```
$not_true         = false       # boolean
$num_tokens       = 115         # number
$my_name          = 'Dave'      # string
$my_list          = [1,4,7]     # array
# array to array
[$first, $last] = ['Gutzon','Borglum']
# hash
$key_pairs        = {name => 'Joe', uid => 1001}
```

VARIABLES (CONT'D)

- may not be reassigned...

```
$var = 1
$var = 2
```

LAB: VARIABLES

- add a **`$super_group`** variable to your users module so that we don't have the group name repeated multiple times

```
$super_group = 'supercool'
```

FACTER

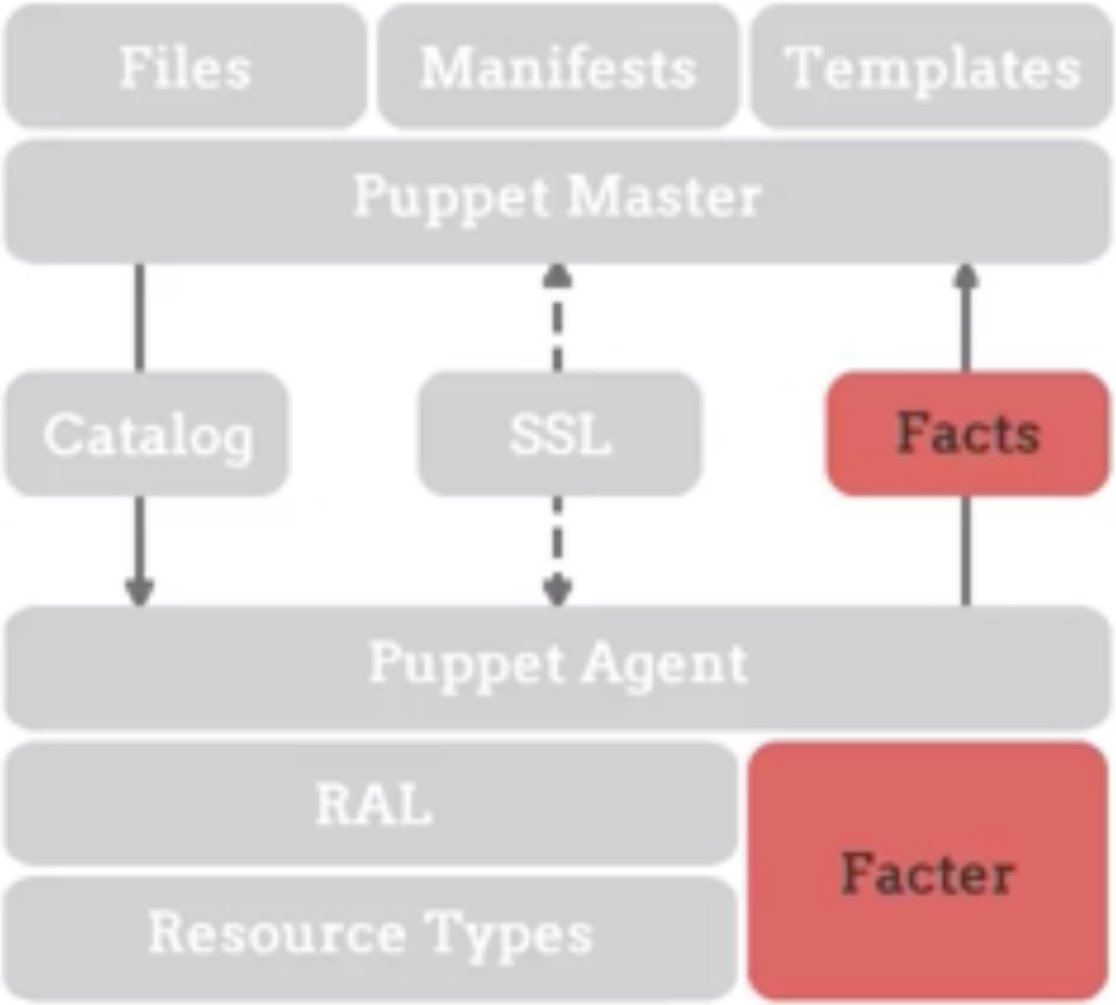
FACTER

- cross-platform system profiler
- gathers node info (e.g., hardware, OS, etc.)
- discovers and reports per-node *facts*, which then become available in your Puppet manifest as variables

FACTER

- let's run **`facter`** on our node
- all facts are available as **`$::fact_name`** (variables) in your manifests
- the '::' means "top scope"–we'll have more to say about this later

FACTER IN THE PUPPET ARCHITECTURE



LAB: FACTER

- create a manifest with a **`notify`** resource declaration
- add a fact such as **`osfamily`** to your notify resource to show that the fact was available as a top-scope variable

FACTER: EXTERNAL FACTS

- sometimes you want to add your own site-specific data
- Puppet can differentiate nodes in ways that are important to your organization
- e.g., using different **`ntp`** servers in different datacenters
- external facts
  - **`/etc/facter/facts.d/foo.{txt,yaml,json}`**
  - could be executable and **`facter`** will run it to parse key/value pairs
  - **`export FACTER_fact=value`**
- custom facts
  - same idea, but they live in modules
  - typically Ruby code but can be other executables

FACTER EXTERNAL FACTS EXAMPLE

```
class postgres {
   user { "postgres":
      ensure     => present,
      managehome => true,
   }

   file { "/home/postgres/.psqlrc":
      ensure      => present,
      owner       => postgres,
      group       => postgres,
      mode        => ‘0644’,
      source      => "puppet:///modules/postgres/psqlrc",
      require     => User["postgres"],
   }
}
```

LAB: EXTERNAL FACTS

- As root…
- add external facts, both as a key/value text file and as an executable bash script
- verify the external facts exists using **`facter`**
- create a manifest with a **`notify`** resource that accesses both of your external facts
- apply your manifest

- **NOTE: `facter` facts end up as lower case variables in your manifests, even if they are mixed case to being with.**

RECAP: FACTER AND EXTERNAL FACTS

- discovers and reports per-node facts
- facts are available in your manifests as **`$variables`**
- you can create external facts
- custom facts

TEMPLATES

TEMPLATES

- documents that combine code, data, and text to produce a final rendered (substituted) output
- goal is to manage a complicated piece of text with simple inputs
- often used to manage the contents of configuration files
- written in a templating language
- Embedded Ruby (ERB): uses Ruby code in tags
- Embedded Puppet (EPP): uses Puppet expressions in special tags (only works with Puppet 4+)

TEMPLATES (CONT'D)

- templates are files or strings that are passed to a function to be evaluated, returning a final string value
- to use variables in templates, they need to be available ("in scope") when the template function is called

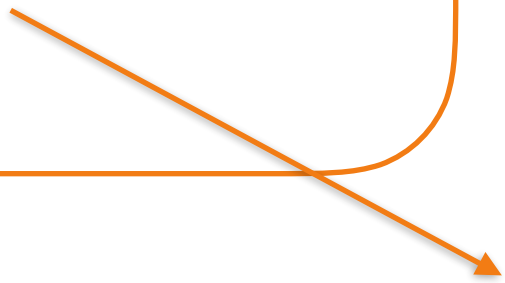
TEMPLATED CLASS MOTD

```
class motd {
  $hostname = $::hostname
  file { '/etc/motd':
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => template('motd/motd.conf.erb'),
  }
}
```

```
Welcome to <%= @hostname %>
```

TEMPLATED CLASS MOTD (CONT'D)

```
class motd {
  $hostname = $::hostname
  $java_versions = [ '1.6', '1.7' ]
  file { '/etc/motd':
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => template('motd/motd.conf.erb'),
  }
}
```



MOTD.CONF.ERB

```
Welcome to <%= @hostname %>

Installed Java versions are:
<% @java_versions.each do |version| -%>
  <%= version %>
<% end -%>
```

- `<%, %>` means begin/end Ruby evaluation
- `<%=` means include text in template you're making
- `-%>` means swallow the NL

LAB: TEMPLATES

- manage **`ntp.conf`** via a template
- add an array variable named servers which holds hostnames (**`time.apple.com`**, **`us.pool.ntp.org`**, **`time.nist.gov`**) as we did with **`motd`** example
- apply your class to ensure templating is working

RECAP: TEMPLATES

- templates are documents combining code, data, and text to produce a final rendered string
- often used for config files
- templated classes
- embedded Ruby

MASTER/AGENT CONFIGURATION

!! DAILY DOUBLE !!

- What are the valid characters for a variable name?
- What is facter and why would we use it?
- What are templates used for?
- What is needed to make templates work?

MASTER/AGENT CONFIGURATION

- so far we've only used **`puppet apply`** on a node, or agent…
- Puppet is typically used in a client-server environment
- It's time to connect an agent to our server…
- on your master: run **`facter fqdn`** to find out its internal hostname

MASTER/AGENT CONFIGURATION (CONT'D)

- ON YOUR AGENT
  - …add an entry in **`/etc/hosts`** for the master **using its IP address and "fqdn" values** similar to the following:

```
10.0.0.??          ip-10-0-0-?? .us-west-2.compute.internal
```

  - …add the server setting in **`/etc/puppet/puppet.conf`** using the fqdn value from the master

```
[main]
server=ip-10-0-0-?? .us-west-2.compute.internal
```

MASTER/AGENT CONFIGURATION (CONT'D)

NOTE: do this as root or sudo on agent!

```
puppet agent --test
```

-or-

```
puppet agent --test --server=<FQDN>
```

(if server not listed in **puppet.conf**)

- executing this command will
  - create an SSL key for your agent and an SSL cert request for your agent
- if no server specified, Puppet will look for a host named **puppet** via **/etc/hosts** or DNS
- server can be specified in **puppet.conf**, as we did on last slide

MASTER/AGENT CONFIGURATION (CONT'D)

- if we run **`puppet agent --test`** on an agent whose cert has not yet been signed by master, we'll get a potentially confusing error message

```
root@ip-10-0-0-236:~# puppet agent -t
Info: Creating a new SSL key for ip-10-0-0-236.us-west-2.compute.internal
Info: Caching certificate for ca
Info: Caching certificate_request for ip-10-0-0-236.us-west-2.compute.internal
Info: Caching certificate for ca
Exiting; no certificate found and waitforcert is disabled
```

MASTER/AGENT CONFIGURATION (CONT'D)

- to complete the connection we need to sign the CSR that the agent sent to the master
- on the master **as root**: we can look at the CSRs that are waiting to be signed:

```
puppet cert list
puppet cert list --all
```

- and we can sign the cert as follows:

```
puppet cert sign <FQDN>
puppet cert sign --all
```

- autosign mode (secure if you trust network)
- on your agent **as root**: run **`puppet agent -t`**

MASTER/AGENT CONFIGURATION (CONT'D)

```
root@ip-10-0-0-236:~# puppet agent -t
Info: Caching certificate for ip-10-0-0-236.us-west-2.compute.internal
Info: Caching certificate_revocation_list for ca
Info: Caching certificate for ip-10-0-0-236.us-west-2.compute.internal
Warning: Unable to fetch my node definition, but the agent run will continue:
Warning: undefined method `include?' for nil:NilClass
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for ip-10-0-0-236.us-west-2.compute.internal
Info: Applying configuration version '1497926783'
Notice: Finished catalog run in 0.05 seconds
```

- agent has connected and our signed cert has authenticated the session with the master
- master doesn't how to classify the agent yet
- so we'll either get an empty catalog or an error depending on configuration

LAB: FIRST CONFIGURATION ITEM

- on master: edit **`/etc/puppet/manifests/site.pp`** to include a node definition (also called a node statement)

```
node '<agent's FQDN>' {
   include users
}
```

- on agent: run **`id`** or look at **`/etc/passwd`** to verify the users are not there
- run **`puppet agent --test`**
- run **`id`** or look at **`/etc/passwd`** to see users ARE there

NODE MATCHING/ CLASSIFICATION

NODE MATCHING/CLASSIFICATION

- the process of assigning classes to agents
- node statements can contain (almost) any Puppet code, it is recommended that you only use them to declare classes
- even if a node matches multiple node statements, the master will match at most one, and it should be the most specific

NODE MATCHING (CONT'D)

```
File {
  mode => '0600',
}

node 'mail.example.com' {
  include mail
}

node /.*\.example\.com$/ {
  include web
}

node default {
  notify { "${clientcert} is not classified": }
}
```

LAB: NODE MATCHING

- on master: create three types of node definitions in **`site.pp`**
  - hostname
  - regex
  - default
- on agent: verify results with **`puppet agent -t`**

PUPPET DATA FLOW

PUPPET DATA FLOW

- manifests/classes exist only on master…why?
  - centralize where you deploy code
  - the things that master does are all CPU-intensive and you can offload it from agents
  - code deploying and logging becomes easier
- *node* is a discrete system managed by a Puppet agent
- so how does information flow between them…?

PUPPET DATA FLOW (CONT'D)

- agent submits name, environment, and facts to Puppet server
- server compiles a Puppet catalog for the node
- agent implements the catalog on the node
- agent does not see Puppet code, just the compiled catalog
- (by the way the catalog exists in **`/var/lib/puppet`** if you want to look at it…)

FUNCTIONS

FUNCTIONS

- written in Ruby
- Puppet labs **`stdlib`** module on the forge provides a ton of useful functions…go take a look!
  - https://forge.puppet.com/
- live in Puppet source code and **`<module_name>/lib/puppet/parser/functions/function_name.rb`**
- **`https://docs.puppet.com/puppet/3.8/function.html`**

FUNCTIONS (CONT'D)

- some built-in functions to be familiar with
  - **fail(): abort building catalog with error message**
  - **template(): evaluate template into string**
  - **hiera(): retrieve data from data broker**
  - include(): declare class if not already declared
  - create_resources(): declare resources from structured data
  - split(): split a string into an array

CONDITIONALS

CONDITIONALS: IF

```
if $::virtual == 'xen' {
  notify { 'avoid':
    message => "Avoid this host."
  }
}
elsif $::virtual == 'physical' {
  notify { 'metal':
    message => "This is a real server."
  }
}
else {
  notify { 'virtualized':
    message => "This is a worker node."
  }
}
```

CONDITIONALS: UNLESS

```
unless $::id == "root" {
  notify { 'hoi_polloi':
    message => "You're a non-root user."
  }
}
```

- same as **`if $::id != "root"`**
  - can be confusing!
- Puppet 4 lets you have an **`else`** clause

CONDITIONALS: CASE

```
case $::id {
  'root': {
    notify { 'super':
      message => "You da boss!",
    }
  }
  'daemon': {
    notify { 'ghastly':
      message => "Ghostly!",
    }
  }
  default: {
    notify { 'rank_and_file':
      message => "Sorry, you're a nobody.",
    }
  }
}
```

CONDITIONALS: SELECTORS

```
$user_level = $::id ? {
  'root'       => 'super',
  'daemon'     => 'ops',
  /jane|john/ => 'cool',
  default      => 'standard',
}

notify { 'test':
  message => "You are a ${user_level} user.",
}
```

LAB: CONDITIONALS

- let's make our **`ntp`** module work on CentOS and Ubuntu
- we'll need conditional logic to handle name differences for the service (**`ntpd`** on CentOS vs. **`ntp`** on Ubuntu)
- we'll need to use the **`osfamily`** fact ('RedHat' covers CentOS, and 'Debian' covers Ubuntu)

RECAP: CONDITIONALS

- **`if/else/elsif`** same as programming languages
- **`unless`** is same as **`if not`**, can be confusing
- **`case`** very commonly used
- selectors are less common
- TIP: save your current version of the ntp module, e.g.
  - **`cd /etc/puppet/modules; cp -r ntp ntp_save`**

CLASS PARAMETERS

CLASS PARAMETERS

- variables allow class behavior to be modified without modifying the class
  - API for your classes
- parameters can have default values so they are not mandatory in the declaration
  - parameters without default values must be set in the declaration to prevent an error
- parameters are useable as variables in other classes
- e.g., the **`servers`** variable of an **`ntp`** class is **`$ntp::servers`**

CLASS PARAMETERS EXAMPLE

```
class motd (
  $warning_prefix = false,
  $message,
) {

  $standard_warning = 'THIS SYSTEM IS MONITORED, PRIVATE PROPERTY.  UNAUTHORIZED ACCESS IS NAUGHTY!'

  if $warning_prefix {
    $message_real = "${standard_warning}\n${message}"
  }
  else {
    $message_real = $message
  }

  file { '/etc/motd':
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => $message_real,
  }

}
```

CLASS PARAMETERS (CONT'D)

- class would be declared as follows

```
class { 'motd':
  message => "Restricted.",
}
```

- **`$message`** is a required variable so must be supplied

LAB: CLASS PARAMETERS

- parameterize your **`ntp`** class with **`$isserver, $isclient,`** and **`$ntp_servers`** params, where **`$isserver`** and **`$isclient`** are Booleans with a default value (remember that it's possible for a machine to be both an NTP server and an NTP client), and **`$ntp_servers`** is a required array of timeservers
  - we don't really go through what's needed to make your machine be an NTP server, but it's enough to do something like a **`notify`** that indicates whether it's a server or a client (or both)
- declare your class and pass with various combinations of parameters
- apply!

RECAP: CLASS PARAMETERS

- allow modification of class behavior without modifying class code
- parameters can have default values (and should have them, when applicable)

RECAP: DAY 2

- Variables
- Facts and **`facter`**
- Templates
- Master/agent configuration
- Node matching
- Puppet data flow
  - How Puppet works under the hood
- Functions
- Conditionals
- Class parameters

DAY 3 AGENDA

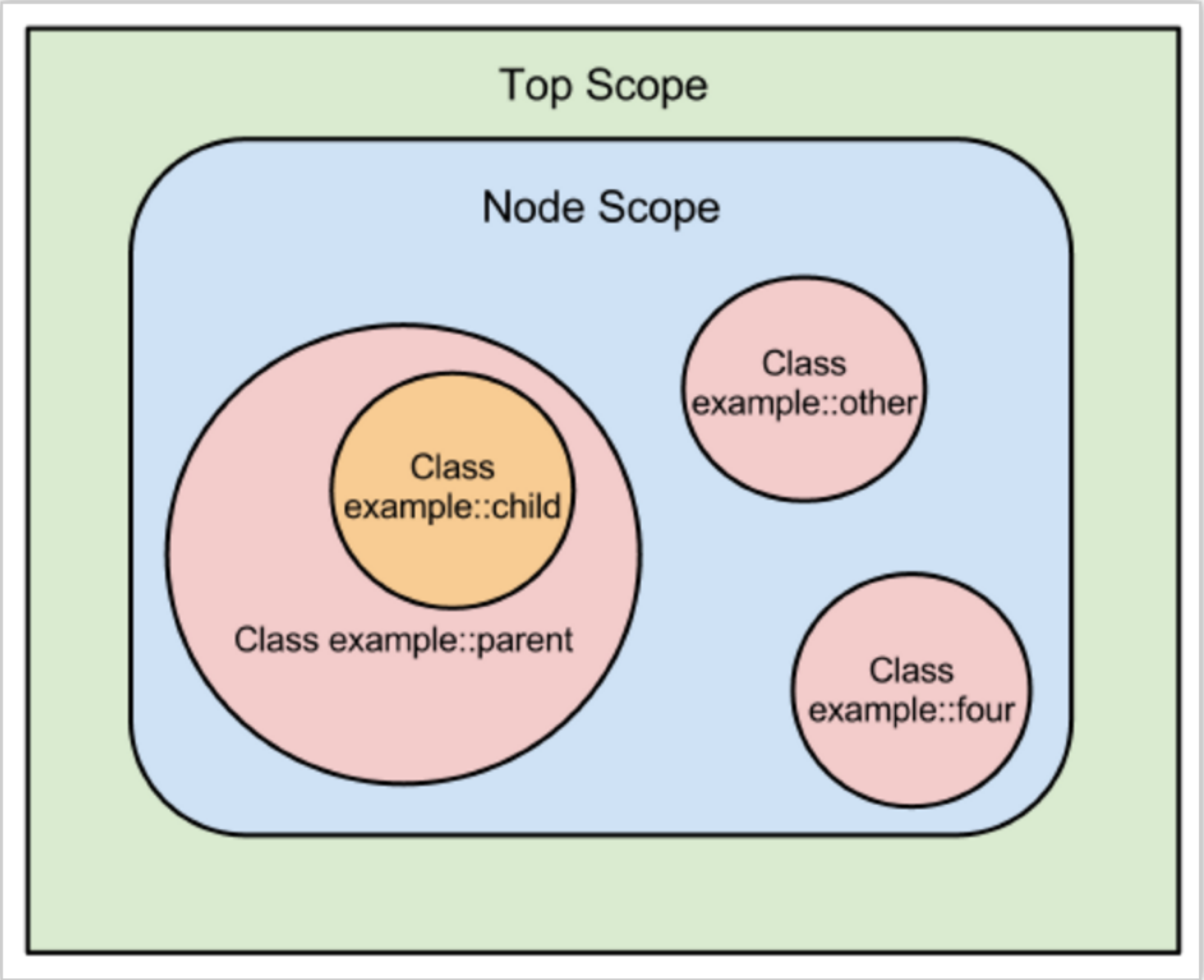
- Scope & Inheritance (via **`params.pp`** pattern)
- **`hiera:`** key/value lookup, assign class params, hiera_include, assign classes to nodes
- Reporting
- Puppet Forge
- Roles and Profiles
- R10k (and Environments)
- External node classifiers
- RSpec
- Rouster

SCOPE

!! DAILY DOUBLE !!

- What mechanism did we use to introduce per-node facts?
- What is one “gotcha” with per-node fact variable names?
- What feature of Puppet allows us to extend the capabilities for creating complex configuration files?
- Name a language feature we used to define classes that allow for differences in implementation - e.g. NTP service names on different OSes? (NOTE: there are at least two features)

SCOPE


Top Scope
Node Scope
Class example::other
Class example::child
Class example::parent
Class example::four

TOP SCOPE

- code that is outside any class definition, type definition, or node definition exists at top scope
- variables and defaults declared at top scope are available everywhere

```
# site.pp
$variable = "Hi!"

class example {
  notify {"Message from elsewhere: $variable":}
}

include example
```

NODE SCOPE

- code inside a node definition exists at node scope
- only one node definition can match a given node, therefore only one node scope can exist at a time
- variables and defaults declared at node scope are available everywhere except top scope

```
# site.pp
$top_variable = "Available!"
node 'puppet.example.com' {
  $variable = "Hi!"
  notify {"Message from here: $variable":}
  notify {"Top scope: $top_variable":}
}
notify {"Message from top scope: $variable":}
```

LOCAL SCOPE

- code inside a class definition or defined type exists in a local scope
- variables and defaults declared in a local scope are only available in that scope and its children

```
# /etc/puppet/modules/scope_example/manifests/init.pp
class scope_example {
  $variable = "Hi!"
  notify {"Message from here: $variable":}
  notify {"Node scope: $node_variable Top scope: $top_variable":}
}

# /etc/puppet/manifests/site.pp
$top_variable = "Available!"
node 'puppet.example.com' {
  $node_variable = "Available!"
  include scope_example
  notify {"Message from node scope: $variable":}
}
notify {"Message from top scope: $variable":}
```

SCOPE (CONT'D)

```
class motd {
  include motd::secure
  file { '/etc/motd':
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => $motd::secure::warning,
  }
}
```

```
class motd::secure {
  $warning = 'THIS SYSTEM IS MONITORED, PRIVATE
        PROPERTY. UNAUTHORIZED ACCESS IS NAUGHTY!'
}
```

INHERITANCE

INHERITANCE (PARAMS.PP PATTERN)

- default values of class parameters cannot use conditional logic before the class begins (at the `{`)
- to allow default values to differ based on fact data (typically **`$::osfamily`**), their values must come from variables in Puppet code that has been already parsed
- this allows for default values to be overridden and therefore work in more diverse situations
- inheriting a class forces it to be parsed before the inheriting class
- inheritance in Puppet was not designed for this, but it is the only recommended use of it anymore

INHERITANCE (PARAMS.PP PATTERN)...(CONT'D)

```
class webserver::params {
  $packages = $::osfamily ? {
    'Debian' => 'apache2',
    'RedHat' => 'httpd',
  }
}
```

```
class webserver(
  $package = $webserver::params::packages
) inherits webserver::params {
  package { $package: ensure => present }
}
```

INHERITANCE (PARAMS.PP PATTERN)...(CONT'D)

```
class ntp (
  $client       = $ntp::params::client,
  $server       = $ntp::params::server,
  $servers      = $ntp::params::servers,
  $service_name = $ntp::params::service_name,
) inherits ntp::params {
  package { 'ntp':
    ensure => present,
  }
  # template ntp.conf.erb uses $client, $server, $servers
  file { '/etc/ntp.conf':
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => template('ntp/ntp.conf.erb'),
  }
  service { $service_name:
    ensure => running,
  }
}
```

LAB: NTP::PARAMS.PP

- use the params.pp pattern to parameterize the **`$ntp_servers`** variable in your **`ntp`** class
- define the params.pp class to set the **`$ntp_servers`** variable based on the value of the **`$::datacenter`** variable
  - you can add (or keep the other **`$is`** variables), if desired
- declare your class and test with datacenter=east
- when datacenter is not defined — use **`undef`** (no quotes) as the left value in your selector

RECAP: INHERITANCE

- Best practice: only use inheritance when
  - you have conditional default params
  - you want those params to be able to be overridden at declaration time
- Note: it can be confusing where param values are coming from
- Puppet is not object oriented!

HIERA

HIERA ("HIGH-RA")

- key/value lookup tool for configuration data
- enhances Puppet and lets you set node-specific data without repeating yourself (DRY)
- keeps site-specific data out of your manifests
- Puppet classes can request the data they need, and your hiera data will act like a sitewide config file

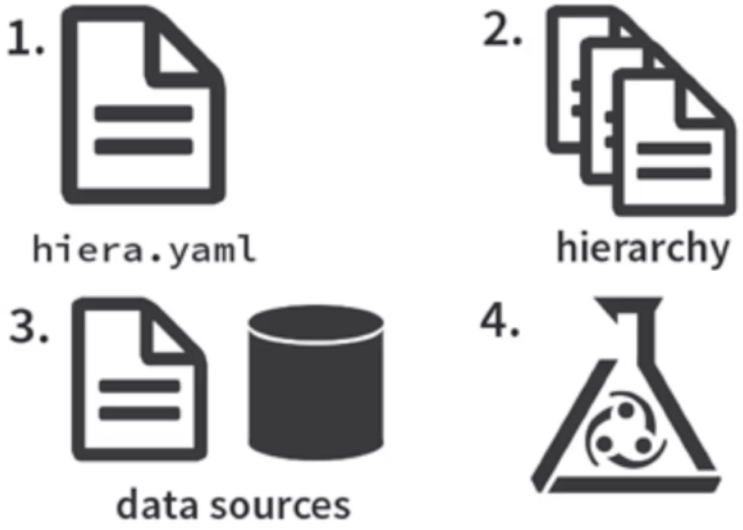
BENEFITS OF HIERA

- easier to configure your own nodes
  - default data with multiple levels of override is EASY
- easier to reuse public Puppet modules
  - no need to edit code, just put the needed data in **`hiera`** and include the class you need
- easier to publish your own modules for collaboration
  - no need to scrub your site-specific data

BENEFITS OF HIERA (CONT'D)

- write common data for most nodes
  - override some values, say, for machines located at a particular facility
  - and further override some of those values for one or two unique nodes
- … you only have to write down the differences between nodes

USING HIERA



1. create a **`hiera.yaml`** configuration
2. arrange a hierarchy that fits your site and your data
3. write your data sources
4. use your data in puppet…let's see how this works

USING HIERA

- when running **`hiera`** from the command line, config file is at **`/etc/hiera.yaml`**, but when run via Puppet it's at **`/etc/puppet/hiera.yaml`**

```
---
:backends:
  - yaml
:yaml:
  :datadir: /etc/puppet/hieradata
:hierarchy:
  - "%{::clientcert}"
  - "%{::datacenter}"
  - common
```

EXAMPLE .YAML FILE

```
---
message: this is a message

ntp_servers:
  - time.apple.com
  - us.pool.ntp.org
  - foo.bar.com
```

USING HIERA (CONT'D)

- We can run hiera directly from the command line:

```
hiera message
```

- We can provide variables for testing:

```
hiera message ::datacenter=east
```

- Or run puppet from the command line to test hiera

```
puppet apply -e 'notice(hiera(message))'
```

- Or invoke hiera from within a manifest and use the resulting value:

```
$msgval = hiera('message')
notify { "$msgval": }
```

LAB: HIERA

- start with a version of your **ntp** class which is not parameterized (make a copy and remove the parameters, if need be)
- call out to **hiera** to get the ntp servers to be inserted into the **ntp.conf** file, using a call like this:

```
$ntp_servers = hiera('ntp_servers')
```

  (that's not the way it would be done, but it's more interesting than just looking up a key such as 'message', as we did in class)
- add an **ntp_servers** key to multiple levels of your hierarchy (e.g., **common**, **datacenter**, and **node**) and demonstrate that the value of **ntp_servers** changes, i.e., one set of servers for the common machine in your hierarchy, a different set for the east datacenter, and a still different set for a specific machine

USING HIERA TO SET CLASS PARAMETERS

USING HIERA TO SET CLASS PARAMS

- Our parameterized **`ntp`** class requires that we supply an **`ntp_servers`** parameter
- Puppet automatically looks up parameters in **`hiera`**, using **`<CLASS NAME>::<PARAMETER NAME>`** as the lookup key
- So, if our class is named **`ntp_param`**, we can put the data into **`hiera`** as follows:

```
ntp_param::ntp_servers:
        - time.apple.com
        - us.pool.ntp.org
        - time.nist.gov
```

LAB: USING HIERA TO SET CLASS PARAMS

- Verify you have added an **`ntp_servers`** key to multiple levels of your hierarchy (e.g., **`common`**, **`datacenter`**, and **`node`**)
  - if not, add this key where missing and modify the key name appropriately
- Modify your **`site.pp`** to declare your parameterized **`ntp`** class without passing the **`ntp_servers`** parameter
- Demonstrate that the value of **`ntp_servers`** changes, i.e., one set of servers for the common machine in your hierarchy, a different set for the east datacenter, and a still different set for a specific machine

USING HIERA TO REPLACE PARAMS.PP PATTERN

```
class webserver(
  $package = hiera(webserver::package, 'httpd')
) {
  package { $package:
    ensure => present,
  }
}
```

- if **`hiera`** returns a value for **`webserver::package`**, then **`$package`** is set to that value, otherwise **`$package`** is set to 'httpd'

RECAP: HIERA

- key/value lookup tool for configuration data
- makes multiple levels of override EASY
- but adds complication so don't use it unless you really need it
  - can be frustrating to troubleshoot!
- YAML can be cumbersome to edit and typos can be disastrous
- TIP: test a lot before implementing for real!

USING HIERA TO ASSIGN CLASSES TO NODES

HIERA_INCLUDE

- you can use **`hiera`** to assign classes to nodes with the special **`hiera_include`** function
- choose a key name, e.g., **`classes`**, to represent the classes you want to assign to a node
- add the **`classes`** key throughout your hierarchy
- add **`hiera_include('classes')`** to **`site.pp`**
- **`hiera_include()`** uses an array merge lookup to create classes array, i.e., ALL values for the classes key are combined and flattened into a single array
- classes are merged with any that are specified in **`site.pp`**

HIERA_INCLUDE (CONT'D)

```
--- # common.yaml
classes:
  - base
  - security
```

HIERA_INCLUDE (CONT'D)

```
--- # east.yaml
classes:
  - base::linux
```

HIERA_INCLUDE (CONT'D)

```
--- # foo.bar.com.yaml
  classes:
    - apache
```

(so every node would get `base` and `security` classes, east nodes would get those plus `base::linux`, and foo.bar.com would get all of those plus `apache`)

SITE.PP

```
hiera_include('classes')

node 'ip-192-168-168-134.us-west-2.compute.internal' {
  notify { 'node!': }
}

node /ip-.*/ {
  notify { 'regex match!': }
}

node default {
  notify { 'default!': }
}
```

LAB: HIERA_INCLUDE

- Add the **`classes`** key throughout your hierarchy
- Add **`hiera_include('classes')`** to site.pp
- Test and ensure merge is occurring

THE PUPPET FORGE

PUPPET FORGE

- online community of Puppet modules submitted by community members
- modules for helping you manage applications such as the Apache webserver
- custom facts for things like getting warranty information from laptops you've deployed
- custom types and providers for helping you manage things like MySQL databases natively with Puppet

LAB: PUPPET FORGE

- go to Puppet Forge
- find a module you want to try and see what it can do (suggestions: **`mysql`**, **`postgresql`**, **`apache`**)
- read **`README`** and sample manifests to be sure you understand what it will do
- download the module
- ensure it was installed

PUPPET FORGE: WRAPPER CLASSES

```
class my_lamp {
  include apache
  include apache::mod::php
  class { 'mysql::bindings':
    php_enable => true,
  }
  class { 'mysql::server':
    root_password => 'hunter2',
  }
}
```

RECAP: PUPPET FORGE

- good for you, good for the community
- custom facts
- custom types and providers
- don't reinvent the wheel!

REPORTING WITH PUPPET

REPORTING WITH PUPPET

- the final thing the agent does is tell you what worked, what didn't work, etc.
- for all of the "should" values, you get a report
- not necessarily state data, it's change data
  - e.g., "package foo was absent; it is now present"
  - not "these packages are installed on the agent…"
- once master has the report, it decides what to do with it
- controlled by the **`report`** setting in **`puppet.conf`**
  - email, chat, twitter, database, splunk, syslog

REPORTING WITH PUPPET (CONT'D)

- how to configure report processors

```
# /etc/puppet/puppet.conf
# [...]
[main]
reports = store,log
```

REPORTING WITH PUPPET (CONT'D)

- **`http.rb`**: send reports via http or https
- **`log.rb`**: send all received logs to the local log destinations, usually syslog
- **`rrdgraph.rb`**: graph all available data about hosts using the RRD library
  - As of Puppet 4, RRD functionality has been removed, see https://tickets.puppetlabs.com/browse/PUP-3260
- **`store.rb`**: store YAML report on disk in the reportdir directory
- **`tagmail.rb`**: sends specific log messages to specific email addresses based on the tags in the log messages

LAB: REPORTING WITH PUPPET

- run **`puppet agent -t`** on a node
- view the report in syslog on the master
- view the report file in the **`/var/lib/puppet`** dir
- disable the store report processor (**`reports=log`**)
- restart **`puppetmaster`**
  - e.g. **`service puppetmaster restart`**
- run **`puppet agent -t`** on a node
- reporting should have gone to syslog but not to the reports dir

CREATING DEFINED RESOURCE TYPES

DEFINED RESOURCE TYPES

- blocks of Puppet code that can be evaluated multiple times with different parameters
- once defined, they act like a new resource type–you can cause the block to be evaluated by declaring a resource of that type
- suppose every time we create a new user, we also want to create the **`.ssh`** subdir and the **`authorized_keys`** file
- we can create a defined resource type to achieve this…

DEFINED RESOURCE TYPES (CONT'D)

```
define users::my_add_user(
    $comment, $shell, $uid, $gid, $password
) {
    user { "$title":
        name     => "$title",
        ensure   => present,
        comment  => "$comment",
        home     => "/home/$title",
        shell    => "$shell",
        uid      => "$uid",
        gid      => "$gid",
        password => "$password",
    }

    file { "/home/$title/":
        ensure  => directory,
        owner   => $title,
        group   => $gid,
        mode    => 750,
        require => User[$title],
    } # ...continued...
```

$title is a "free" parameter and always contains the title of the resource

DEFINED RESOURCE TYPES (CONT'D)

```
    file { "/home/$title/.ssh":
        ensure  => directory,
        owner   => $title,
        group   => $gid,
        mode    => 700,
        require => File["/home/$title/"]
    }

    file { "/home/$title/.ssh/authorized_keys":
        ensure  => present,
        owner   => $title,
        group   => $gid,
        mode    => 600,
        require => File["/home/$title/.ssh"]
    }
}
```

LAB: DEFINED RESOURCE TYPES

- Create a defined resource type for a "file-backup"
- When you declare a file-backup resource, it makes a backup of the file in the /tmp directory
  - make a copy of the file, rather than a link
- Set the permissions on the backup copy to 0400 to prevent accidental deletion
- Add any other features you desire

ROLES AND PROFILES

ROLES AND PROFILES

- the roles and profiles method is the most reliable way to build reusable, configurable, and refactorable system configurations
- it's not a straightforward recipe: you must think hard about the nature of your infrastructure and your team
- it's also not a final state: expect to refine your configurations over time
- rather, it's an approach to designing your infrastructure's interface–sealing away incidental complexity

ROLES AND PROFILES (CONT'D)

- it adds two extra layers of indirection between your node classifier and your component modules
- using roles and profiles separates your code into three levels:
  - component modules–normal modules that manage one particular technology (e.g., **`puppetlabs/apache`**)
  - **`profiles`**–wrapper classes that use multiple component modules to configure a layered technology stack
  - **`roles`**–wrapper classes that use multiple profiles to build a complete system configuration
- extra layers of indirection might seem like they add complexity, but they give you a space to build practical, infrastructure-specific interfaces

PROFILES

- normal Puppet module, with the same rules and structure as any other module
- not shared with the community, written for a specific organization
- contains wrapper classes that declare classes from downloaded community modules and assign desired class parameters to customize behavior of the "component" modules
- can contain resource declarations, like user resources in **`profile::admins`** class

PROFILES (CONT'D)

```
class profile::base {
  $ntpservers = hiera('ntp_servers'),
  class { 'ntp':
    servers => $ntpservers,
  }
  user { 'dave':
    ensure => present,
    gid    => 'supercool',
  }
}
```

PROFILES (CONT'D)

```
class profile::webapp::frontend {
  include apache
  include apache::mod::php
  class { 'mysql::bindings':
    php_enable => true,
  }
  package { 'my-webapp':
    ensure => latest,
  }
}
```

PROFILES (CONT'D)

```
class profile::webapp::backend {
  $webapp_mysql_rootpw        =
           hiera('webapp_mysql_rootpw')
  $webapp_frontend_host       =
           hiera('webapp_frontend_host')
  $webapp_frontend_mysql_user =
           hiera('webapp_frontend_mysql_user')
  $webapp_frontend_mysql_pass =
           hiera('webapp_frontend_mysql_pass')
  class { 'mysql::server':
    root_password => $webapp_mysql_rootpw,
  }
  mysql::db { 'webapp':
    user     => $webapp_frontend_mysql_user,
    password => $webapp_frontend_mysql_pass,
    host     => $webapp_frontend_host,
    grant    => ['SELECT', 'UPDATE'],
  }
}
```

LAB: PROFILES

- create **`profile`** module
- create **`base`** profile that includes **`users`** class that we wrote earlier
- create an **`ntp`** server profile
- create an **`ntp`** client profile
- if desired, create a **`puppetmaster`** profile
  - (the package is called **`puppet-server`** and the service is called **`puppetmaster`**)

ROLES

ROLES

- also a normal Puppet module
- also specific to an organization and not shared
- contains classes that combine profile classes into higher level functional classifications to be used when classifying nodes
- should only contain **`include`** calls
- only one role should be assigned to a node, create a new role instead of assigning more than one to a node

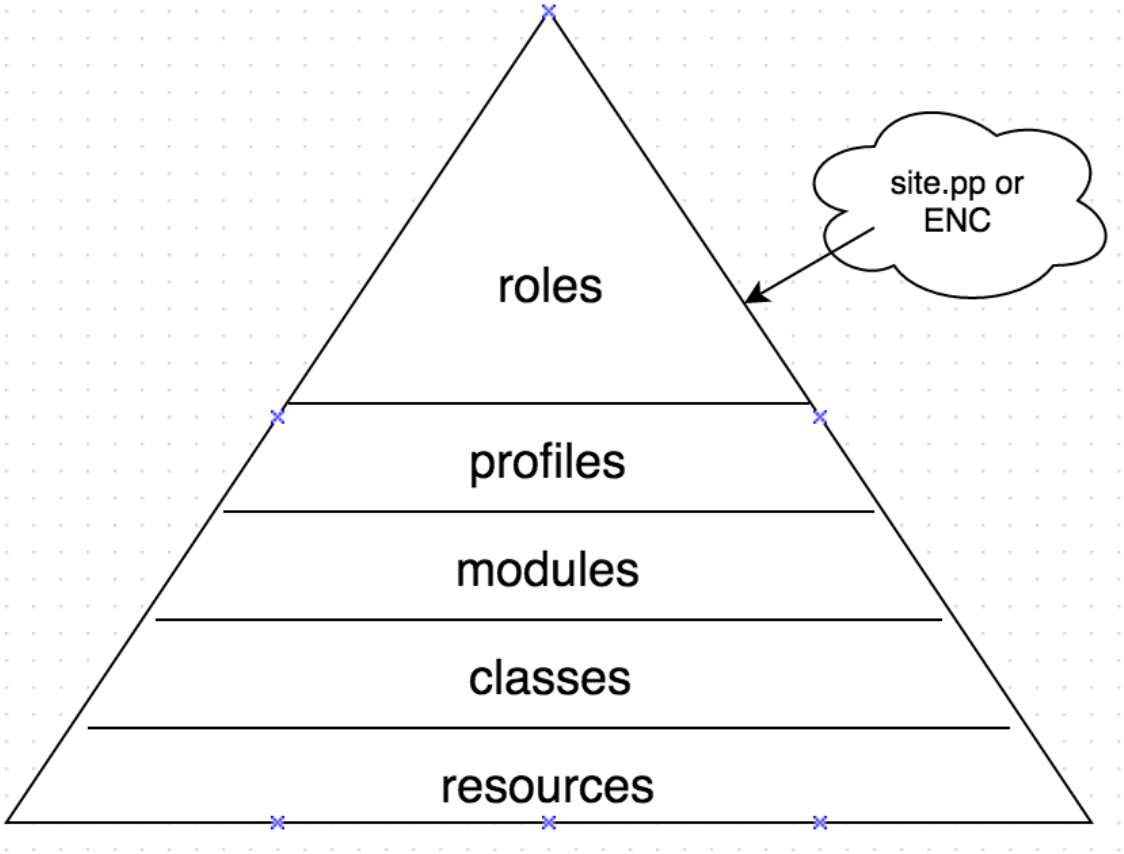
ROLES (CONT'D)

```
class role::webapp::backend {
  include profile::base
  include profile::webapp::backend
}

class role::webapp::frontend {
  include profile::base
  include profile::webapp::frontend
}

class role::webapp::aio {
  include profile::base
  include profile::webapp::backend
  include profile::webapp::frontend
}
```

ROLES (CONT'D)



LAB: ROLES

- Uninstall **`ntp`** from both your master and your agent
- create the **`role`** module
- create an **`ntp`** server role
- create an **`ntp`** client role
- if desired, create a **`puppetmaster`** role
- classify your two nodes with your new roles in **`site.pp`**
- run **`puppet agent -t`** on both master and agent and observe that **`ntp`** was installed on both and configured differently in the process (if you didn't implement an **`ntp`** server, you can simulate this by modifying your **`ntp`** class to create different config files for server and client, i.e,. add an extra comment for a server)

EXTERNAL NODE CLASSIFIERS

EXTERNAL NODE CLASSIFIERS

- external program used to classify nodes in your infrastructure
- merges data with what's in **`site.pp`**, and if there are conflicts, the catalog compile fails
  - typical recommendation is to use **`site.pp`** or ENC but not both for classification
- What can an ENC do?
  - declare classes w/parameters and parameter values
  - set an environment, overriding the environment requested by the agent
  - set top scope variables

EXTERNAL NODE CLASSIFIERS (CONT'D)

- What can't an ENC do?
  - resource defaults
  - other Puppet code such as functions, conditional logic
- ...therefore, an ENC is most effective if you've done a good job of separating your configurations out into classes and modules

EXTERNAL NODE CLASSIFIERS (CONT'D)

- configuring ENC in **`puppet.conf`** on the master

```
[master]
  node_terminus = exec
  external_nodes = /usr/local/bin/puppet_node_classifier
```

EXTERNAL NODE CLASSIFIERS (CONT'D)

- example output from an ENC

```
---
classes:
    common:
    puppet:
    ntp:
        ntpserver: 0.pool.ntp.org
    aptsetup:
        additional_apt_repos:
            - deb localrepo.example.com/ubuntu lucid production
            - deb localrepo.example.com/ubuntu lucid vendor
parameters:
    ntp_servers:
        - 0.pool.ntp.org
        - ntp.example.com
    mail_server: mail.example.com
    iburst: true
environment: production
```

EXTERNAL NODE CLASSIFIERS (CONT'D)

- Puppet Enterprise includes a fully-featured ENC with a RESTful (http) API and uses the idea of groups of nodes that are programmatically matched based on rules,
  e.g., **`memsize > 200GB && kernel = Windows`**,
  then **`include role::mssql`**, etc.

LAB: EXTERNAL NODE CLASSIFIERS

- create a script (bash, Python, Perl, Ruby, etc.) to function as a simple external node classifier
- should accept a certname as a single argument
- should output a YAML node object like the one we just saw
- configure your ENC in **`puppet.conf`** and test
- be sure that the class(es) from your ENC is/are merged with the class(es) from **`site.pp`** (e.g., you might have your ENC output the users class, and your **`site.pp`** could do a **`notify`**)

RECAP: EXTERNAL NODE CLASSIFIERS

- use ENC or site.pp but not both
- what can an ENC do?
- what can site.pp do?
- why use an ENC?

R10K

ENVIRONMENTS

- separate directories on the master, each of which has its own **`site.pp`**, everything…
- changes can be tested in the non-production/test environment before making them live (or production)
- 100% segregated sets of source code, modules, dependencies, etc.

ENVIRONMENTS (CONT'D)

- to configure environments:
  - edit **`puppet.conf`** and add

    `environmentpath = $confdir/environments`

    ...in [main] or [master] section
- you must have a directory for every environment that any nodes are assigned to
- at minimum, you should have a **`production`** environment
- create a directory named production in your **`environmentpath`**
- once created, you can add modules, a main manifest, and an **`environment.conf`** file to it

R10K

- Puppet environment tool
- given a list of Puppet modules, version numbers, and source locations in the **`Puppetfile`**, it downloads them all into the modules directory
- enables the dynamic environment workflow via revision control with **`git`**
- when one (or more) repository(ies) is/are configured it can use the branches of this repository to deploy Puppet environments and other code on the master

R10K EXAMPLE PUPPET FILE

```
# Install puppetlabs/apache and keep it up to date with 'master'
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache'

# Install puppetlabs/apache and track the 'docs_experiment' branch
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache',
  :ref => 'docs_experiment'

# Install puppetlabs/apache and pin to the '0.9.0' tag
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache',
  :tag => '0.9.0'

# Install puppetlabs/apache and pin to the '83401079' commit
mod 'apache',
  :git    => 'https://github.com/puppetlabs/puppetlabs-apache',
  :commit => '83401079053dca11d61945bd9beef9ecf7576cbf'

# Install puppetlabs/apache and track the 'docs_experiment' branch
mod 'apache',
  :git    => 'https://github.com/puppetlabs/puppetlabs-apache',
  :branch => 'docs_experiment'
```

R10K EXAMPLE OUTPUT

```
[root@master1 puppet]# r10k deploy environment -pv
[R10K::Task::Deployment::DeployEnvironments - INFO] Loading environments from all sources
[R10K::Task::Environment::Deploy - NOTICE] Deploying environment production
[R10K::Task::Puppetfile::Sync - INFO] Loading modules from Puppetfile into queue
[R10K::Task::Module::Sync - INFO] Deploying redis into /etc/puppetlabs/puppet/
environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying property_list_key into /etc/puppetlabs/
puppet/environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying wordpress into /etc/puppetlabs/puppet/
environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying r10k into /etc/puppetlabs/puppet/
environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying make into /etc/puppetlabs/puppet/
environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying concat into /etc/puppetlabs/puppet/
environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying vsftpd into /etc/puppetlabs/puppet/
environments/production/modules
[R10K::Task::Module::Sync - INFO] Deploying portage into /etc/puppetlabs/puppet/
environments/production/modules
…
```

LAB: R10K

- **`gem install git`**
- **`gem install r10k --version 1.4.2`**
- write a simple **`Puppetfile`** (in /etc/puppet/environments) containing a module from the Puppet Forge
  - suggest the following:

```
    mod 'apache',
      :git => 'https://github.com/puppetlabs/
puppetlabs-apache',
     :tag => '1.11.1'
```

- run **`r10k puppetfile install --verbose`**
- look in the modules directory and ensure the desired modules have been installed
- look at the **`metadata.json`** file and look at the version that was installed to be sure it is what you asked for

RECAP: R10K

- Puppet environment deployment tool
- enables dynamic environment workflow
- http://somethingsinistral.net/blog/rethinking-puppet-deployment/

RSPEC-PUPPET

TESTING WITH RSPEC-PUPPET

- rspec is a testing framework for Ruby
  - mocks out the variables so that we can test manifests without actually applying them
- fast!
- typically part of a testing pipeline after syntax check and **`puppet-lint`** but before actually applying code to VM to test
  - do the fast stuff first, before possibly wasting time doing the slow stuff

TESTING WITH RSPEC-PUPPET (CONT'D)

- unit testing
  - given some input (manifests), Puppet will give us an expected output (catalog)
  - catalog never makes it to an agent and is never enforced by an agent
- the main purpose of the spec subdir in your modules

TESTING WITH RSPEC-PUPPET (CONT'D)

```
require 'spec_helper'

describe '<name of the thing being tested>' do
  # Your tests go in here
end
```

```
require 'spec_helper'

describe 'logrotate::rule' do

end
```

TESTING WITH RSPEC-PUPPET (CONT'D)

```
require 'spec_helper'

describe 'logrotate::rule' do
  let(:title) { 'nginx' }

  it { should contain_class('logrotate::setup') }
end
```

```
it do
  should contain_file('/etc/logrotate.d/nginx').with({
    'ensure' => 'present',
    'owner'  => 'root',
    'group'  => 'root',
    'mode'   => '0444',
  })
end
```

TESTING WITH RSPEC-PUPPET (CONT'D)

```
context 'with compress => true' do
  let(:params) { {:compress => true} }

  it do
    should contain_file('/etc/logrotate.d/nginx') \
      .with_content(/^\s*compress$/)
  end
end

context 'with compress => false' do
  let(:params) { {:compress => false} }

  it do
    should contain_file('/etc/logrotate.d/nginx') \
      .with_content(/^\s*nocompress$/)
  end
end
```

TESTING WITH RSPEC-PUPPET (CONT'D)

```
context 'with compress => foo' do
    let(:params) { {:compress => 'foo'} }

    it do
      expect {
        should contain_file('/etc/logrotate.d/nginx')
      }.to raise_error(Puppet::Error, /compress must
          be true or false/)
    end
  end
```

TESTING WITH RSPEC-PUPPET (CONT'D)

```
require 'spec_helper'

describe 'logrotate::rule' do
  let(:title) { 'nginx' }

  it { should contain_class('logrotate::rule') }

  it do
    should contain_file('/etc/logrotate.d/nginx').with({
      'ensure' => 'present',
      'owner'  => 'root',
      'group'  => 'root',
      'mode'   => '0444',
    })
  end
```

TESTING WITH RSPEC-PUPPET (CONT'D)

```
context 'with compress => true' do
  let(:params) { {:compress => true} }

  it do
    should contain_file('/etc/logrotate.d/nginx') \
      .with_content(/^\s*compress$/)
  end
end

context 'with compress => false' do
  let(:params) { {:compress => false} }

  it do
    should contain_file('/etc/logrotate.d/nginx') \
      .with_content(/^\s*nocompress$/)
  end
end
```

TESTING WITH RSPEC-PUPPET (CONT'D)

```
  context 'with compress => foo' do
     let(:params) { {:compress => 'foo'} }

     it do
       expect {
         should contain_file('/etc/logrotate.d/nginx')
       }.to raise_error(Puppet::Error, /compress must
                be true or false/)
     end
   end
end
```

ROUSTER

ROUSTER

- developed at salesforce.com
- wrapper for Vagrant, a tool to make easily reproducible development environments on virtual machines, e.g., on your desktop
- adds the ability to run commands and capture their output
- can be used to fire up one or more VMs, run Puppet on them, and then verify that Puppet did the right thing
- has all of your Puppet code and tests it together like an acceptance test so that you can make sure your code continues to work as it changes

ROUSTER (CONT'D)

- enforces the code and tests the machine after the fact to ensure your code did what you thought it would do
- cf. **`beaker`**, a Puppet-supported tool for doing similar testing
- maintained by Puppet
- https://github.com/chorankates/rouster
- https://www.youtube.com/watch?v=N-E6x6MGBpY

LINKS/BOOKS

- Puppet 3.8 docs: https://docs.puppet.com/puppet/3.8/
- https://puppet.com/docs/puppet/4.7/style_guide.html#guiding-principles
- Puppet Forge: https://forge.puppet.com/
- Pro Puppet, Second Edition - Spencer Krum
  (already a bit out of date)
- Learning Puppet 4 - Jo Rhett
- Rouster: https://github.com/chorankates/rouster
  - https://www.youtube.com/watch?v=N-E6x6MGBpY
- Sh*t Garry Says http://garylarizza.com/
- https://github.com/JuanitoFatas/what-do-you-call-this-in-ruby
- https://blog.openshift.com/how-to-avoid-puppet-dependency-nightmares-with-defines/