



Advanced Puppet

for Puppet Masters

Puppet Education
puppetlabs.com/education

© 2013 Puppet Labs

Training & Certification

Advanced Puppet is part of the certification curriculum for the Puppet Professional Certification.

For more information about Puppet Education & Training, please visit:
<http://puppetlabs.com/education>.

For more information about the Puppet Certification Program, please visit:
<http://puppetlabs.com/certification>.

Table of Contents

| | |
|--|-----|
| Course Objective | 2 |
| Course Overview..... | 3 |
| Course Agenda | |
| Day1 | 4 |
| Day2 | 5 |
| Day3 | 6 |
| Day 1 | |
| Lesson 1: Puppet Basics Review..... | 16 |
| Lesson 2: Facts & Functions..... | 50 |
| Lesson 3: Classification | 67 |
| Lesson 4: Advanced Coding Techniques | 81 |
| Lesson 5: Roles & Profiles..... | 114 |
| Lesson 6: Best Practices | 139 |
| Day 2 | |
| Lesson 7: File Manipulation | 146 |
| Lesson 8: Data Separation | 157 |
| Lesson 9: Virtual Resources | 180 |
| Lesson 10: Exported Resources | 197 |
| Lesson 11: Scaling Puppet..... | 215 |
| Day 3 | |
| Lesson 12: Advanced Reporting | 235 |
| Lesson 13: Troubleshooting..... | 251 |
| Lesson 14: Provisioning..... | 268 |
| Lesson 15: MCollective | 285 |
| Capstone Lab | 325 |
| Course Conclusion | 328 |
| Appendix | 333 |

Course Objective

After completing this course, system admins will be able to configure complex system infrastructure using optimized syntax in a Puppet environment with multiple Masters and agents.

Course Overview

In this course you will build a production style environment with:

- Hiera as a single source of truth data source
- PuppetDB as a performant backend for `storeconfigs`
- Multiple masters using a shared certificate authority
- A series of modules adhering to common design patterns

Course Agenda

Day 1

- Puppet Basics Review
- Facts & Functions
- Classification
- Advanced Coding Techniques
- Roles & Profiles
- Best Practices

Course Agenda

Day 2

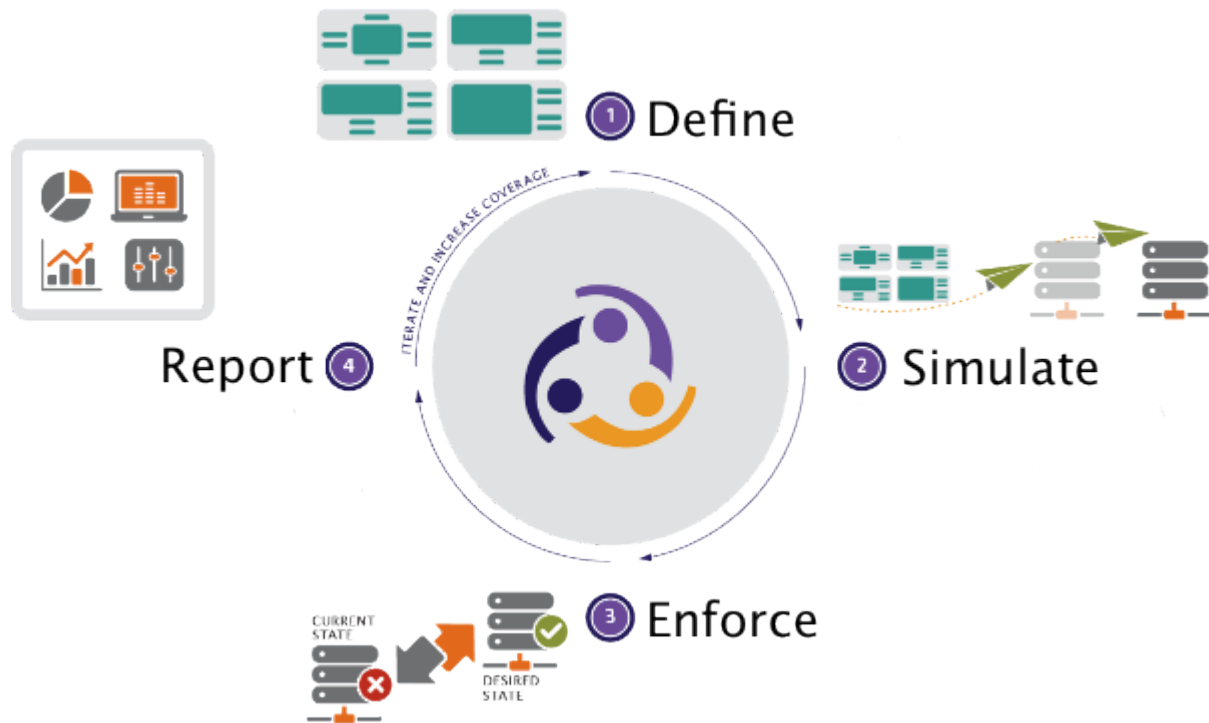
- File Manipulation
- Data Separation
- Virtual Resources
- Exported Resources and Collections
- Server Scaling

Course Agenda

Day 3

- Advanced Reporting
- Troubleshooting
- Provisioning
- MCollective
- Capstone Lab

How Puppet Works



Notes:

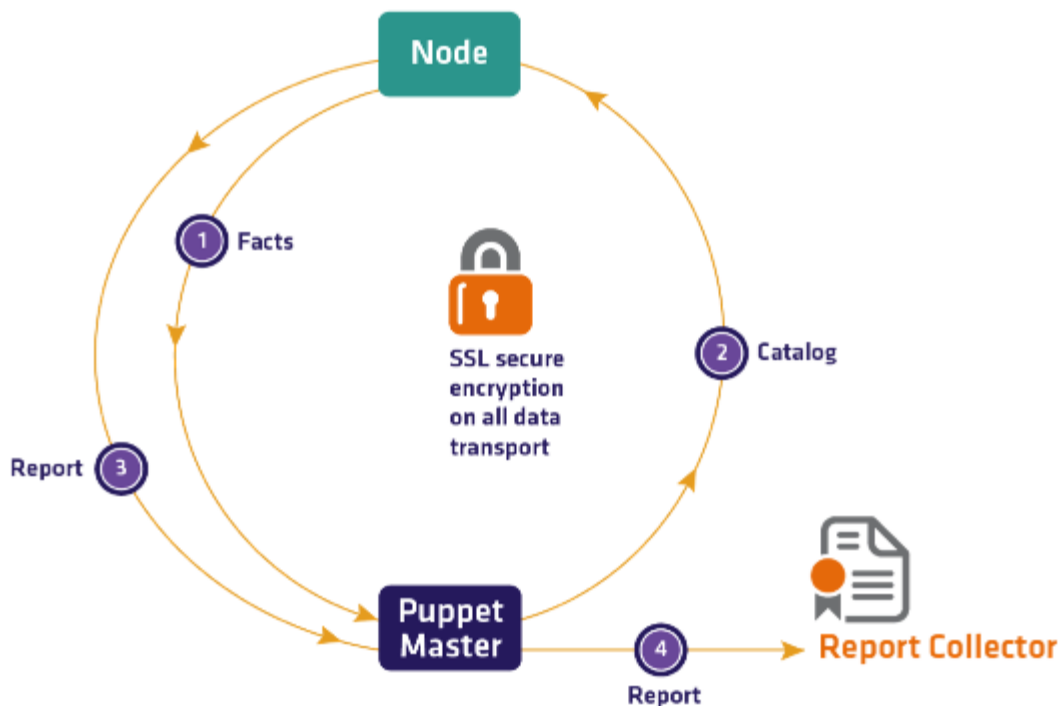
Puppet's unique, model-based approach to automating discovery, configuration, and management follows four steps to automating your infrastructure: 1. Define, 2. Simulate, 3. Enforce, and 4. Report.

1. Define your infrastructure and its desired state.
 - Use Puppet's domain specific language (DSL) to define the resources - users, services, packages, anything - you want to manage
 - You can also use one of the hundreds of pre-built, freely downloadable modules from The Forge repository as a starting point
 - You can choose to manage as few resources as a single file, or as many as all resources on the entire node - it's totally up to you.
 - CUSTOMER BENEFIT: These definitions are now both re-useable and portable across operating systems, deployment environments (physical, virtual, public cloud)
2. Simulate these resource definitions
 - The model-based approach is what allows Puppet to simulate configuration changes without impacting anything in production
 - This allows you to understand the impact of these changes on your IT environment before putting them into production

- CUSTOMER BENEFIT: The result is a much more visible and controlled change management of IT automation configuration, resulting in higher service levels
3. Enforce the desired state of your infrastructure
 - Once these definitions are in production, the Puppet Agent checks their actual state against the Puppet Master server every 30 minutes.
 - If the state of an Agent's definitions has drifted or experienced an unauthorized change, Puppet can automatically revert them back into its originally defined, desired state.
 - CUSTOMER BENEFIT: The result is the elimination of configuration drift.
 4. Report on the state of your infrastructure
 - The last step is to aggregate all the changes to the desired state of resources across all nodes into a single report.
 - This gives you complete visibility into the types of changes, the rate of those changes, who is making those changes, etc.
 - CUSTOMER BENEFIT: The result is that you're able to understand how changes in your infrastructure impact changes in service levels, including availability, reliability, performance, etc.

Lifecycle of a Puppet Agent Run

Data Flow Between Puppet Components



Notes:

1. The Puppet Agent on the node tells the Puppet Master information about itself (hostname, node name, operating system, etc.).
2. The Puppet Master looks up the configuration for that node and sends a Catalog representing that intended configuration back to the node.
3. The node reports back any actions that were taken to enforce that configuration.
4. The Puppet Master server aggregates all the reports from all the nodes and provides a single overview on the state of your infrastructure.

Demo

Puppet Master Installation



Notes:

The instructor will demonstrate the installation the classroom Puppet Master. Students should pay special attention to the network configuration, but should not follow along until instructed to do so.

Pre-installation



- **Objective:**

- Assign a hostname to your master and make that name persist across reboots.

- **Steps:**

- Edit your system's `/etc/hosts` file.
- Configure your system's hostname as `yourname.puppetlabs.vm`

Installation



- **Objective:**

- Install the Puppet Master on your virtual machine.

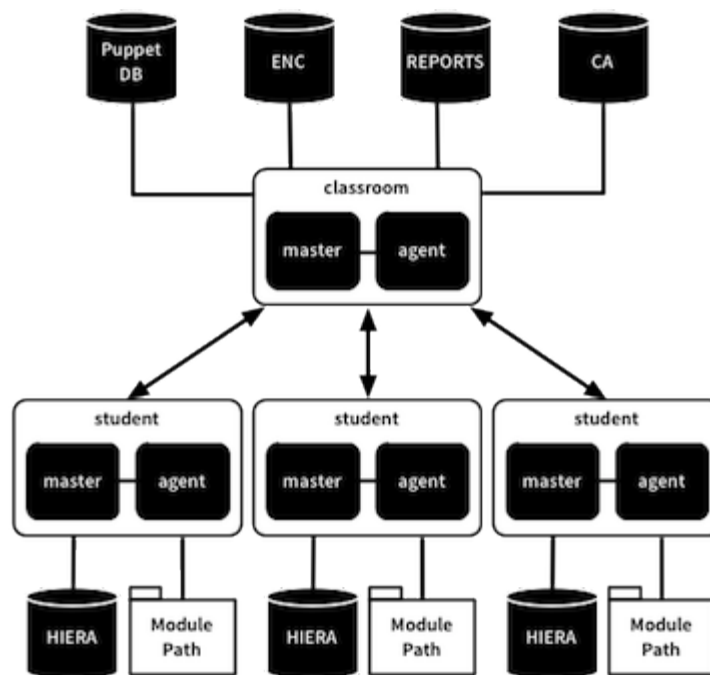
- **Steps:**

- Create an answer file using the Puppet Enterprise Installer.
- The classroom Puppet Master will serve as the Console for your installation.
- Install Puppet Enterprise using the answer file you created.
- Explore your new installation.

Classroom Infrastructure

Each student runs their own master and agent.

- Centralized Roles:
 - Certificate Authority
 - Enterprise Console and Reporting
 - Inventory Service



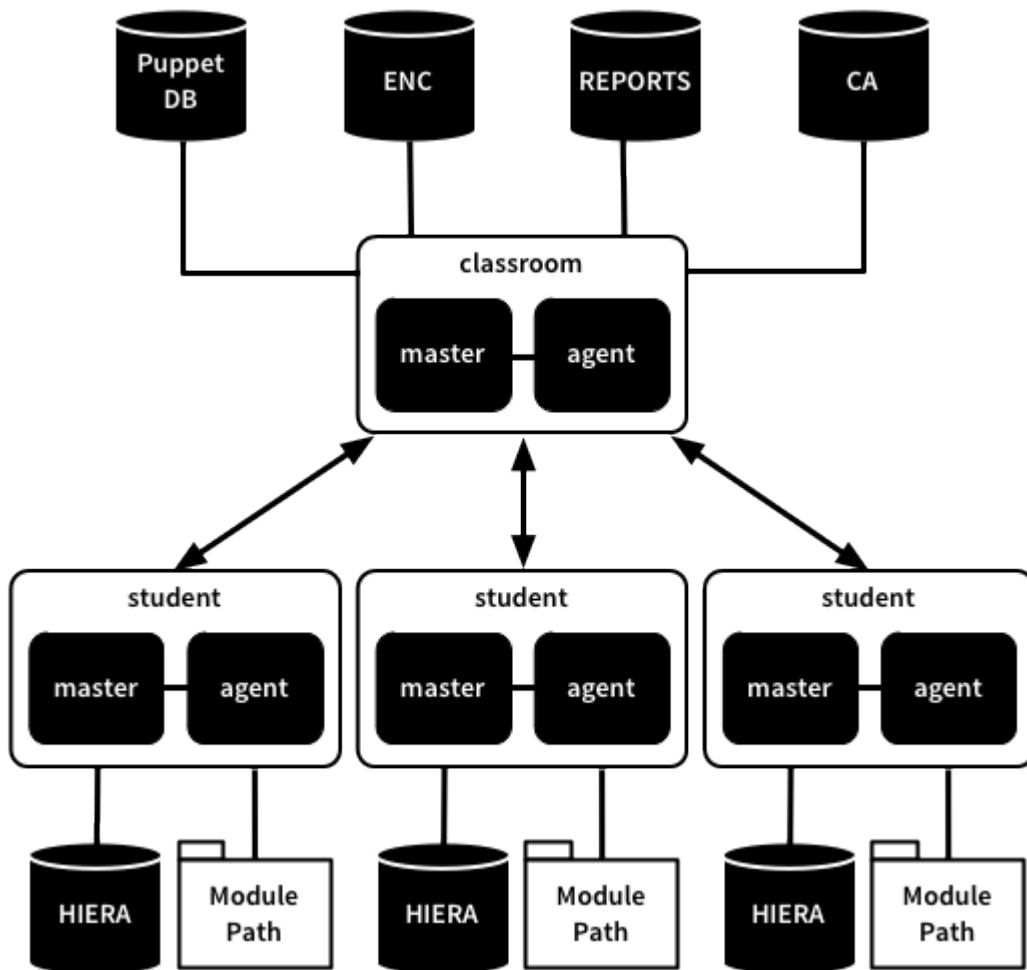
Notes:

The classroom master is also used for initial configuration of the classroom environment.

This architecture is not common as we are sharing some roles, but still maintaining individual Puppet Master environments for each student. This is not as far fetched as it might seem. Consider the use case of a central IT department supporting multiple departments in a company. The certificate authority should be shared, and reporting may be centralized.

This is the infrastructure that we are replicating today. Later on in the class, we will homogenize our masters and place them behind a load balancer to provide a more traditional clustered configuration management service.

Classroom Infrastructure



Configure Your Certificate Authority



- **Objective:**

- Configure your Puppet Master to use the classroom shared certificate authority.

- **Steps:**

- Update your Puppet Master to use a common CA.
- Request a new certificate from the classroom certificate authority.
- Restart your Puppet Master `pe-httpd` service and trigger a Puppet Agent run.

Puppet Basics Review

Lesson 1: Puppet Basics Review

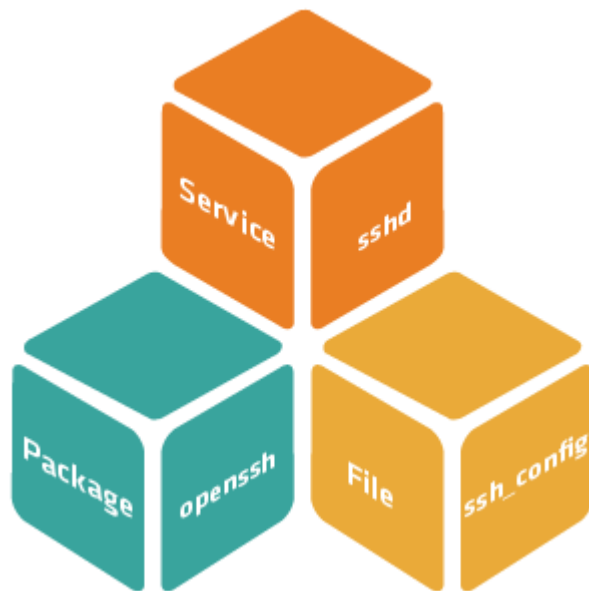
OBJECTIVES

At the end of this lesson, you will be able to:

- Recall fundamental concepts about classes and modules.
- Describe the Resource Abstraction Layer.
- Describe how scope affects the Puppet language.
- Examine `puppet.conf` and identify important configuration options.

Puppet Resources

- Resources are building blocks.
- They can be combined to make larger components.
- Together they can model the expected state of your system.



Resource Types

Specification of attributes that can be used to describe a given resource.

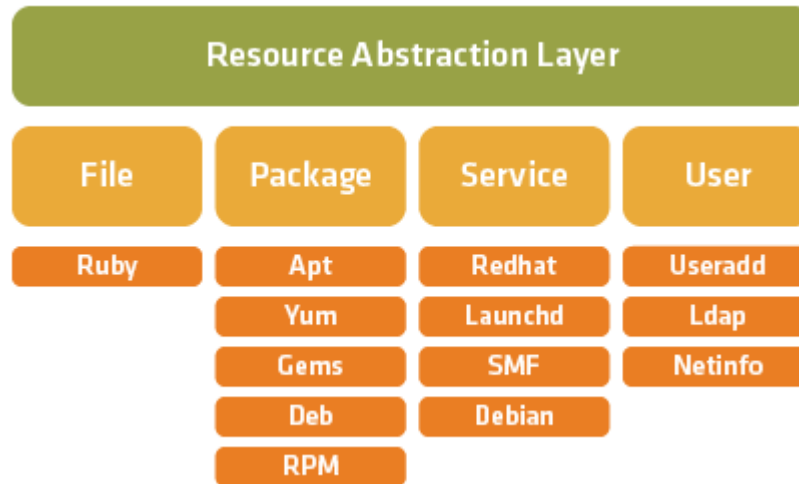
Example:

```
file { '/etc/motd':  
  ensure    => file,  
  path      => '/etc/motd',  
  owner     => 'root',  
  group     => 'root',  
  mode      => '0644',  
  content   => 'This system is managed by Puppet.',  
}
```

Puppet will manage only what you tell it to manage.

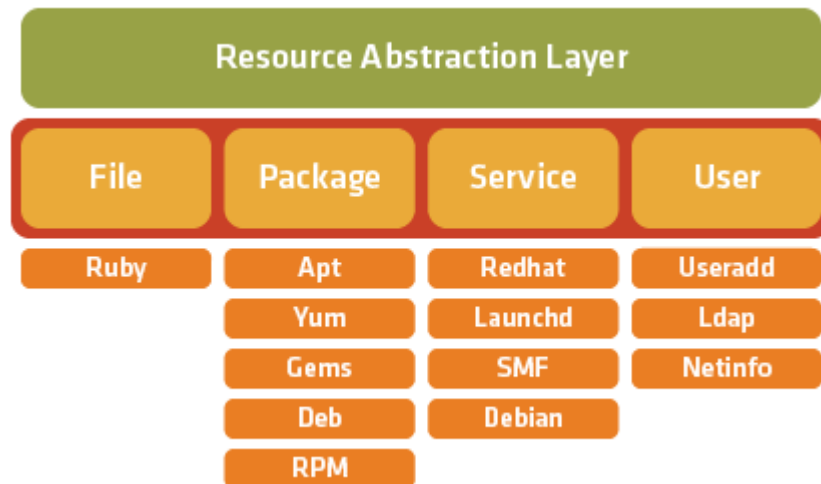
Resource Abstraction Layer

Provides a consistent model across supported platforms.



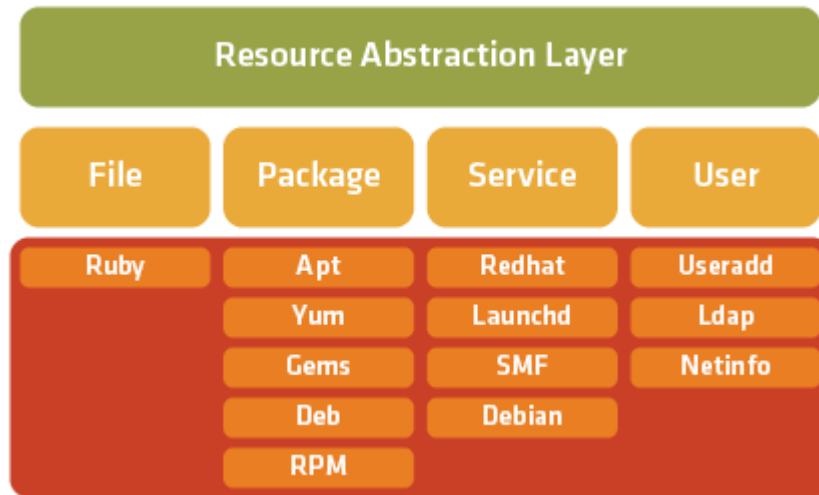
Resource Type

Each resource type has one or more providers.



Providers

The interface between the resource and the OS.



Declarative Modeling Language

- The user will model the desired state.
- Let Puppet figure out how to enforce it.

Comparison

Imperative Shell Code

```
if [ 0 -ne $(getent passwd elmo > /dev/null)$? ]
then
  useradd elmo --gid sysadmin -n
fi

GID=$(getent passwd elmo | awk -F: '{print $4}'`
GROUP=$(getent group $GID | awk -F: '{print $1}'`

if [ "$GROUP" != "$GID" ] && [ "$GROUP" != "sysadmin" ]
then
  usermod --gid $GROUP $USER
fi
```

```
if [ "`getent group sysadmin | awk -F: '{print $1}'`" == "" ]
then
  groupadd sysadmin
fi
```

Declarative Puppet Code

```
user { 'elmo':
  ensure => present,
  gid    => 'sysadmin',
}
```

```
group { 'sysadmin':
  ensure => present,
}
```

Idempotency

- Puppet enforces resources in an idempotent way.
 - only changes resources or attributes that are out of sync
- The same end result no matter how many times Puppet runs.

```
# First Puppet Run
[root@training ~]# puppet agent -t
notice: /Group[sysadmin]/ensure: created
notice: /User[elmo]/ensure: created
notice: Finished catalog run in 0.08 seconds

# Second Puppet Run
[root@training ~]# puppet agent -t
notice: Finished catalog run in 0.03 seconds
```

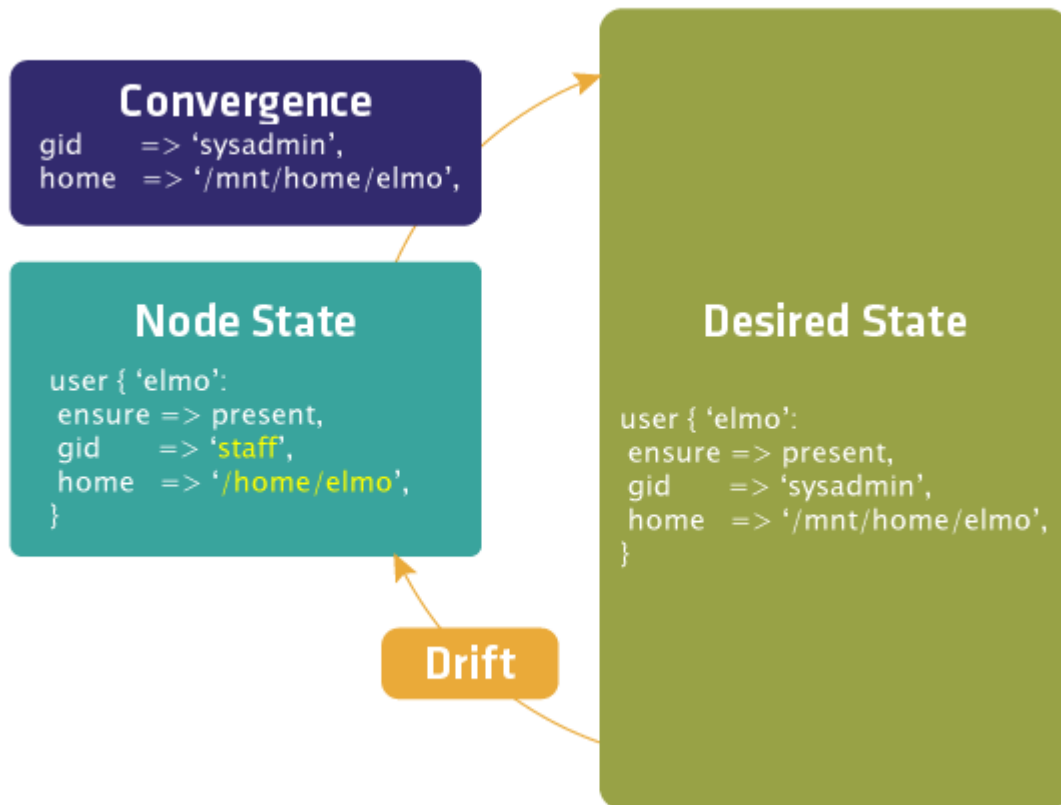
Idempotence: The property of certain operations in mathematics or computer science in that they can be applied multiple times without further changing the result beyond the initial application.

Notes:

- Idempotent - able to be applied multiple times with the same outcome.
- Puppet resources are **idempotent**, since they describe a desired final state rather than a series of steps to follow.
- Source: Puppet Docs - <http://docs.puppetlabs.com/references/glossary.html#idempotent>

Desired State

Describe the state you want.



Modules

Modules are directories that contain your configuration. They are designed to encapsulate all of the components related to a given configuration in a single directory hierarchy.

- They have a pre-defined structure that enable the following:
 - auto-loading of classes
 - file-serving for templates and files
 - auto-delivery of custom Puppet extensions
 - easy sharing with others

Module Layout

- The module directory is named after the module.
- Naming conventions allow for easy collaboration & reusability.

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/ssh/
├── files
│   └── ssh_config          ## source => 'puppet:///modules/ssh/ssh_config',
├── lib
│   ├── facter
│   │   └── role.rb        ## $::role
│   ├── puppet
│   │   └── parser
│   │       └── functions
│   │           └── mastername.rb ## $mastername = mastername()
├── manifests
│   ├── init.pp            ## class ssh { ... }
│   └── server.pp          ## class ssh::server { ... }
├── templates
│   └── sshd_config.erb     ## content => template('ssh/sshd_config.erb'),
└── tests
    ├── init.pp            ## include ssh
    └── server.pp          ## include ssh::server
```

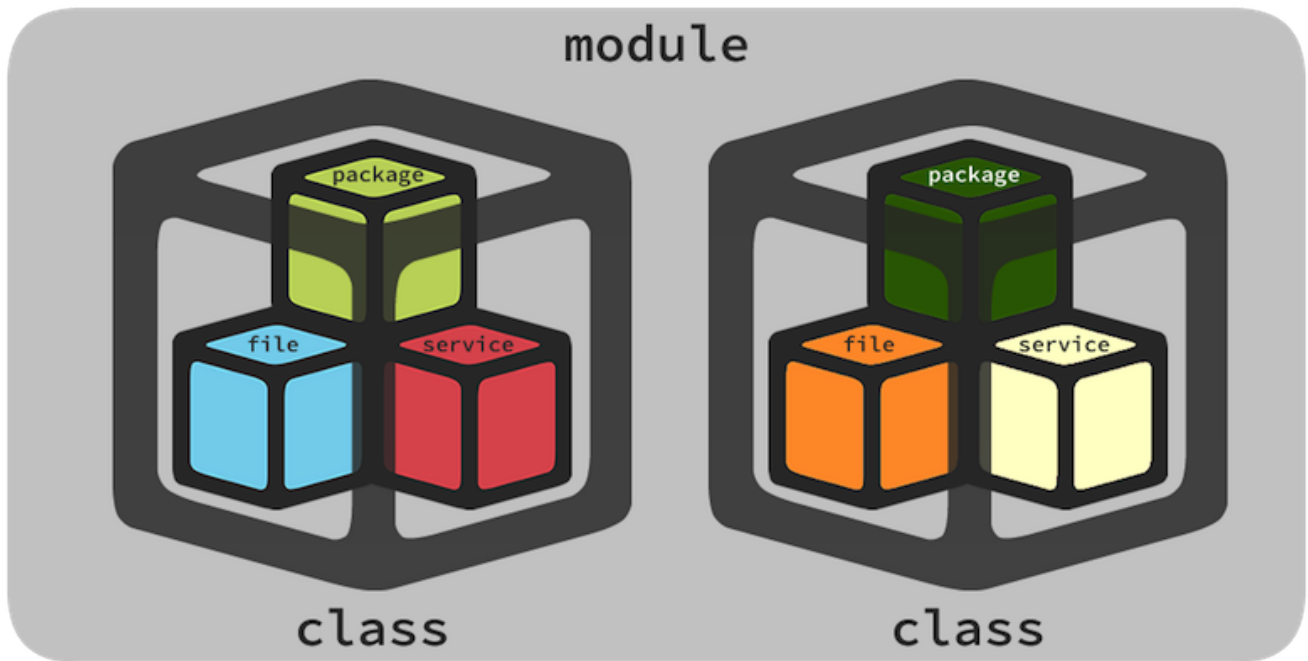
Manifests Directory

The `manifests` directory is where Puppet finds the definitions for:

- classes
- defined types

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/example
├── manifests
│   ├── config.pp    ## class example::config {}
│   ├── init.pp      ## class example {}
│   ├── packages.pp  ## class example::packages {}
│   ├── params.pp    ## class example::params {}
│   ├── server.pp    ## class example::server {}
│   └── user.pp       ## define example::user {}
└── tests
    └── init.pp       ## include example
```

Module Structure



Puppet Classes

Classes define a collection of resources that are managed together as a single unit.

```
# /etc/puppetlabs/puppet/modules/ssh/manifests/init.pp
class ssh {
  File {
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
  }

  package { ['openssh':
    ensure => present,
  ]

  file { ['/etc/ssh/ssh_config':
    require => Package['openssh'],
    source  => 'puppet:///modules/ssh/ssh_config',
  ]

  file { ['/etc/ssh/sshd_config':
    require => Package['openssh'],
    content => template('ssh/sshd_config'),
  ]

  service { ['sshd':
    ensure => running,
    enable => true,
    require => File['/etc/ssh/sshd_config'],
  ]
}
```

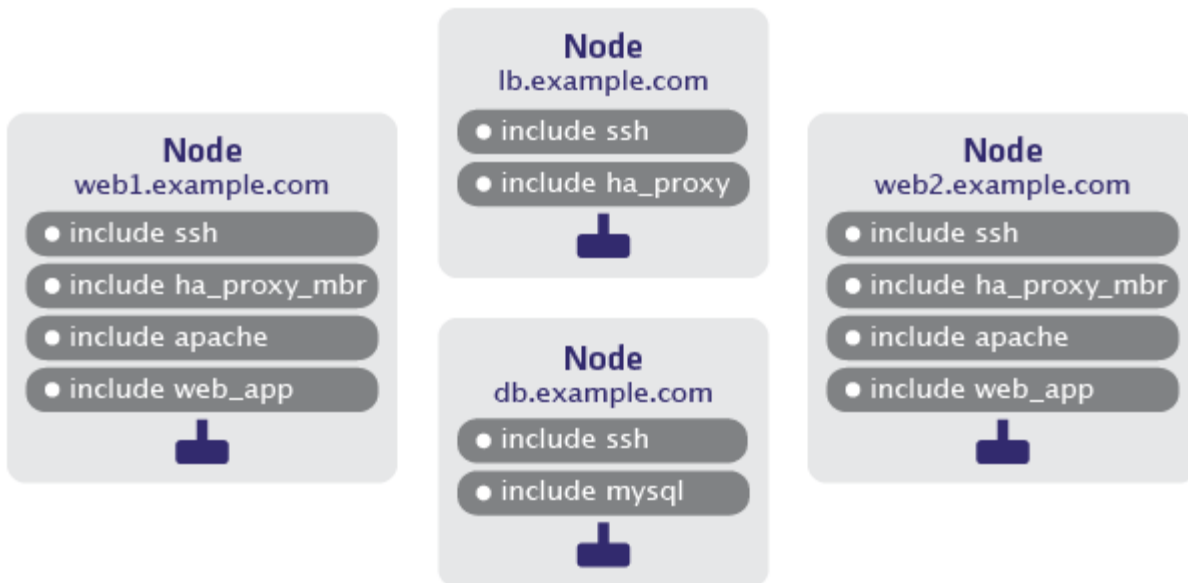
Notes:

Stated another way, package, file, and service are individual Puppet resources bundled together to define a single idea, or class.

Classes are Reusable

Configurations are composable.

- Simply include classes on nodes as appropriate.
- Saves effort and reduces error.



Auto-loading of Classes

In the past, you had to tell your Puppet Master where to find your manifests.

```
# Don't do this
import '/etc/puppet/path/to/your/manifest.pp'
```

- This is not manageable past a dozen or so manifests.
- Does not encourage reusable code.
- Not a best practice.

Auto-loading of Classes

Instead, Puppet searches a specific folder structure to automatically find your class based on its name.

First, Puppet needs to know where to find your modules.

```
# puppet.conf on puppet master
[master]
modulepath=/etc/puppetlabs/puppet/modules
```

Then, your Puppet classes are placed in this predictable structure.

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/ssh/
├── manifests
│   ├── init.pp          ## class ssh { ... }
│   └── server.pp        ## class ssh::server { ... }
└── tests
    └── init.pp          ## include ssh
```

Notes:

- Class or define name must match:
 - modulename
 - filename

files Directory

Predictable module structure enables file serving

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/sudo/  
├── files  
│   └── sudoers      ## source => 'puppet:///modules/sudo/sudoers',  
├── manifests  
│   └── init.pp      ## class sudo { ... }  
└── tests  
    └── init.pp      ## include sudo
```

Puppet Agent automatically retrieves the file from the Master.

- URI structure:
 - `puppet://<server>/modules/<module>/<template>`
 - `<server>` is optional: defaults to server catalog was retrieved from

```
# init.pp  
class sudo {  
  file { ['/etc/sudoers':  
    ensure => file,  
    source => 'puppet:///modules/sudo/sudoers',  
  }  
}
```

templates Directory

Templates are stored in your module much like files are.

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/apache/
├── manifests
│   ├── init.pp          ## class apache { ... }
├── templates
│   └── vhost.erb        ## content => template('apache/vhost.erb'),
├── tests
└── init.pp              ## include apache
```

- `template()` function takes a path instead of a Puppet URI
 - `template('<modulename>/<template>.erb')`
- Runs on the master during compilation
- String output is compiled into catalog

```
file { ['/etc/httpd/conf.d/my_host.conf':
  ensure => present,
  content => template('apache/vhost.erb'),
}
```

Notes:

Other similar functions:

- `file()`
 - this function simply returns the contents of a file on the Puppet Master.
 - note that this function does *not* interpolate the `modulename/filename` format that Puppet URIs and the `template()` function understand.
- `generate()`
 - this function calls a script and returns the `stdout` output of that script.

Lab 1.1: Create a Module



- **Objective:**

- Create a simple module that will manage various files.

- **Steps:**

- Create a module named `basics`.
- Manage `/etc/shells` as a static file resource.
- Manage `/etc/motd` as a templated file resource.
- Test your code and apply it.

lib Directory

Deploy custom plugins & extensions

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/custom
├── lib
│   ├── facter
│   │   └── custom_facts.rb
│   └── puppet
│       ├── parser
│       │   └── functions
│       │       └── custom_function.rb
│       ├── provider
│       │   ├── custom_type
│       │   └── ruby.rb
│       └── type
│           └── custom_type.rb
└── tests
    ├── facts.pp          ## notice($::custom_fact)
    ├── functions.pp      ## notice(custom_function())
    └── types.pp           ## custom_type { 'title': ensure => present, }
```

- Basic custom facts and functions will be covered in this course.
- *Extending Puppet using Ruby* course covers custom types, providers and other extensions.

Distributing Extensions

Puppet uses `pluginsync` to distribute extensions to all agents.

Enable in the `[agent]` section of `puppet.conf` on all agents:

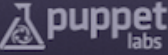
```
[agent]
pluginsync = true
```

- enabled by default in Puppet Enterprise.
- enabled by default in Puppet 3.0.X and higher.
- All extensions are synced before each agent run.

Puppet Forge

Share & Download Modules.

[Puppet Labs](#) : [Open Source Projects](#) [Support](#) [Docs](#) [Bugs](#)

 **forge**

917 modules 382,782 downloads
Since February 2012

Modules matching 'mysql'

20 modules found

nodes/php

243 downloads

Puppet module to manage PHP on debian (and others)

Version 0.2.1 released Jan 29, 2013 | 156 downloads of this version

crayfishx/hiera_mysql

43 downloads

MySQL back end for Hiera

Version 0.2.0 released Jan 16, 2013 | 43 downloads of this version

runthebusiness/mysqlexec

74 downloads

This class runs mysql functions from the command line.

Version 1.0.2 released Jan 15, 2013 | 39 downloads of this version

nextrevision/automysqlbackup

138 downloads

Deploy automysqlbackup via Puppet

[Log In](#)

[Sign Up](#)

[Publish a Module](#)

Find Modules

[Find](#)

[All modules](#)

Popular Tags

[ubuntu \(162 modules\)](#)

[debian \(136 modules\)](#)

[CentOS \(100 modules\)](#)

[rhel \(100 modules\)](#)

[monitoring \(63 modules\)](#)

[networking \(56 modules\)](#)

[security \(52 modules\)](#)

[applications \(50 modules\)](#)

Puppet Module Tool

Search the Puppet Forge & Install Modules from the command line.

Using `puppet module`:

- `puppet module list`
 - `puppet module install <module> [--version]`
 - `puppet module upgrade <module> [--version]`
 - `puppet module search <module>`
 - `puppet module uninstall <module>`
-

Notes:

You can view a full description of each action with `puppet man module` or by viewing the man page at <http://docs.puppetlabs.com/man/module.html>.

Lab 1.2: Install a Module



- **Objective:**

- Download the `puppetlabs/mysql` module and use its database type.

- **Steps:**

- Download the `puppetlabs/mysql` using the module face.
- Trigger a puppet run. Sync your new type & provider.
- Create new module named `databases`.

Variable Format

- Variables should only use letters, numbers, and underscores.
- Variables should not contain hyphens (dashes).

```
$var_one    = 1  
  
$var2       = 'variable two'  
  
# this is an invalid variable and will fail compilation in later Puppet versions  
$var-three = 'invalid'
```

- Earlier versions of Puppet were quite lenient about variable naming.
 - The Puppet 3.0 parser is more strict and will not allow lazy variable naming.
-

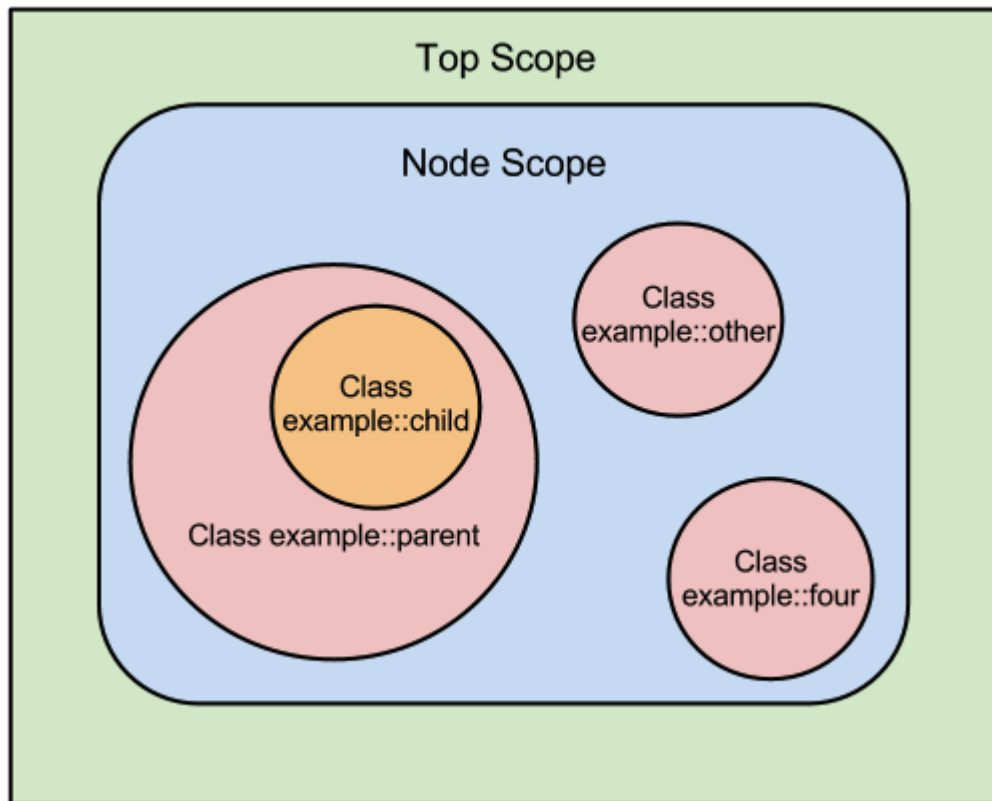
Notes:

- The use of hyphens in qualified variables should be avoided.
- The hyphen (-) invokes a subtraction operation.
- For more information please visit: <http://projects.puppetlabs.com/issues/5268>.

Scope

Partial isolation of areas of code

- Every `class`, `define`, or `node` introduces a new scope.
- Variables declared outside of classes, defines, or nodes are *top scope*.



Variables are constant

Within a single scope variables *cannot* be redeclared.

```
class scope {
  $somecontent = 'Content of /tmp/test_one.txt'
  file { ['/tmp/test_one.txt':
    content => $somecontent,
  ]

  $somecontent = 'Content of /tmp/test_two.txt'    # This will cause an error.
  file { ['/tmp/test_two.txt':
    content => $somecontent,
  ]
}
```

Variables in Non-overlapping Scopes

In non-overlapping scopes, variables are distinct.

```
class one {
  $somecontent = 'This came from scope one'
  file { ['/tmp/one.txt':
    content => $somecontent,
  ]
}

class two {
  $somecontent = 'This came from scope two'
  file { ['/tmp/two.txt':
    content => $somecontent,
  ]
}

notify { $one::somecontent: }
notify { $two::somecontent: }
```

Conditionals

Use case statements to alter the flow of logic.

```
case $::osfamily {  
  'redhat' : { $package_name = 'httpd' }  
  'ubuntu' : { $package_name = 'apache2' }  
  default  : { fail('Unsupported platform') }  
}  
  
package { $package_name:  
  ensure => present,  
}
```

Use selectors for returning a value conditionally.

```
# This will fail if an unknown parameter is encountered  
$package_name = $::osfamily ? {  
  'redhat' => 'httpd',  
  'ubuntu' => 'apache2',  
}  
  
package { 'apache':  
  name => $package_name,  
}
```


Boolean Logic

if/else/elsif

- The following values evaluate to false:
 - undef (an undefined variable)
 - ''
 - false

```
if $mailserver {  
  file { ['/etc/mail': ensure => present }  
} else {  
  file { ['/etc/mail': ensure => absent }  
}
```

Which messages will be displayed?

```
if 'false' {  
  notify {'test case one': }  
}  
if 0 {  
  notify {'test case two': }  
}  
if '' {  
  notify {'test case three': }  
}
```

Notes:

http://docs.puppetlabs.com/puppet/2.7/reference/lang_expressions.html

Regular Expressions

- Puppet supports regular expression matching in expressions.
- Capture groups can be used within a code block.

```
if $::environment =~ /^dev-(\d+)/ {  
  file { '/etc/environment':  
    ensure => file,  
    content => "Development environment number: $1",  
  }  
  
  # the $1 variable does not persist out of its block  
}
```

Lab 1.3: Add Conditional Logic



- **Objective:**

- Update your `basics` module to include custom `motd` messages.

- **Steps:**

- Use conditional logic to set the contents of a `$message` variable.
- Use the contents of that variable to include messages in the `motd` template.
- Experiment with different ways to calculate the variable.

Facts and Functions

Lesson 2: Facts and Functions

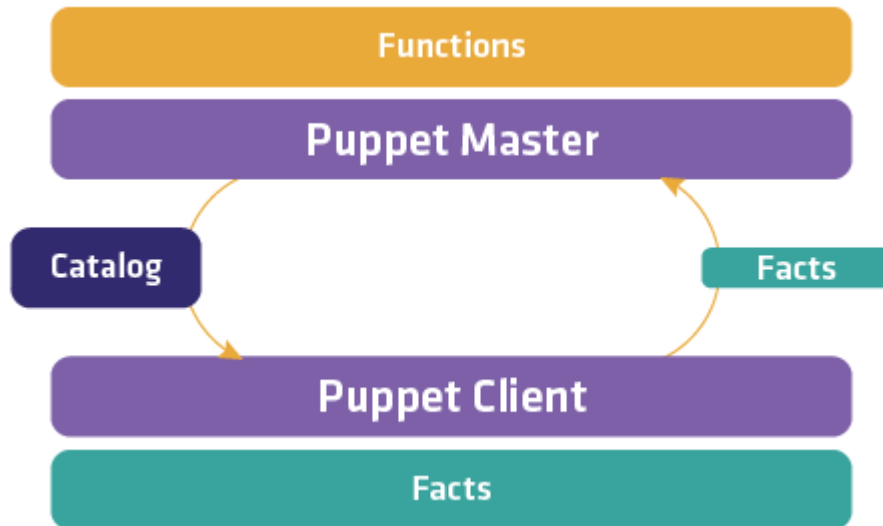
OBJECTIVES

At the end of this lesson, you will be able to:

- Identify the lifecycles of facts and functions.
- Access facts in your Puppet manifests.
- Execute functions in your Puppet manifests.
- Develop simple facts and functions.

Puppet Functions & Facts

Lifecycle of a Puppet Agent Run



Facts Expose Agent Information

- Each agent run starts by sending **facts** to the master:
 - the *only* thing the master knows about the agent
 - exposed as global variables during catalog compilation
- Custom facts can expose arbitrary agent information:
 - physical location of machine
 - JDK version installed
 - application role

Accessing Facts

Facts are exposed as top scope variables

```
class system {  
  $operatingsystem = 'MyOS'  
  notify { "The operating system is: ${operatingsystem}": }  
}
```

```
notice: The operating system is: MyOS
```

```
class truesystem {  
  $operatingsystem = 'MyOS'  
  notify { "The true operating system is: ${::operatingsystem}": }  
}
```

```
notice: The operating system is: CentOS
```

- Fact references should always contain a top scope reference.

Developing Facts

- By convention, the file name matches the name of the fact.
- Multiple facts can be distributed with a single module.

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/custom
├── lib
│   └── factor
│       └── role.rb
```

```
# role.rb
Facter.add('role') do
  setcode 'cat /etc/role'
end
```

Custom Fact

Simple fact to determine the role of a server.

```
# role.rb
Facter.add('role') do
  setcode do
    role = Facter::Util::Resolution.exec('cat /etc/role')
    role.gsub(/<.*?>/m, "")
  end
end
```

- The fact is set to the string value returned from the `setcode` block.
- If no other manipulation is needed you can pass a string to `setcode`.

```
# role.rb
Facter.add('role') do
  setcode 'cat /etc/role'
end
```

Notes:

`setcode` supports a ruby block, or a single command as a string.

Distributing Facts

- Facts are distributed automatically on an agent run with `pluginsync`.
- To access a `pluginsynced` fact on the command line, pass `-p`.

```
[root@training ~]# puppet agent -t
info: Retrieving plugin
notice: /File[/var/opt/lib/pe-puppet/lib/facter/role.rb]/content:
[...snipped diff...]
notice: /File[/var/opt/lib/pe-puppet/lib/facter/role.rb]/content: content changed
'{md5}ab0007a9a726cceeca26845aabb0778e' to '{md5}a7b8fa55d03238d4a6635cead77f5e37'
info: Loading downloaded plugin /var/opt/lib/pe-puppet/lib/facter/role.rb
[...]
info: Caching catalog for training.puppetlabs.vm
info: Applying configuration version '1355242783'
notice: Finished catalog run in 3.14 seconds
[root@training ~]# facter -p role
Desktop class machine
```

- Test facts locally by setting the `FACTERLIB` or `RUBYLIB` environment variables:

```
[root@training ~]# export FACTERLIB=/etc/puppetlabs/puppet/modules/custom/lib
[root@training ~]# facter role
Desktop class machine
```

Notes:

Notice that the Puppet agent run first downloads any new or changed facts and then loads them. This precedes the catalog request and application. The practical implications of this are that custom facts are available for use on the very first Puppet agent run after they are defined. You do not need to sync them on one Puppet run and then use them on the next unless they depend on resources that are managed by Puppet itself.

If a fact is synced via `pluginsync`, then the version of the fact that was synced will take precedence over a fact tested by setting `FACTERLIB`. For this reason, it is often useful to stop the Puppet agent before commencing development so that an incomplete fact doesn't get synced.

facter/facts.d

Facts exposed by stdlib module or Facter 1.7+

```
[root@training ~]# cat /etc/puppetlabs/facter/facts.d/datacenter.json
{
  "location": "portland",
  "cluster": "web"
}
```

```
[root@training ~]# cat /etc/puppetlabs/facter/facts.d/datacenter.yaml
---
location: portland
cluster: web
```

```
[root@training ~]# cat /etc/puppetlabs/facter/facts.d/datacenter.txt
location=portland
cluster=web
```

```
[root@training ~]# facter location
portland
```

Notes:

Puppet Open Source facts.d path is /etc/facter/facts.d

Exercise 2.1: Create a Custom Fact



- **Objective:**

- Create a new `kerberos` module that will contain a custom fact that reads the `default_realm` parameter from `/etc/krb5.conf`.

- **Steps:**

- Ensure you are in your module path.
- Create a new module that will contain your custom fact.
- Create a new ruby file called `default_realm.rb`.
- Install your fact using `pluginsync`.

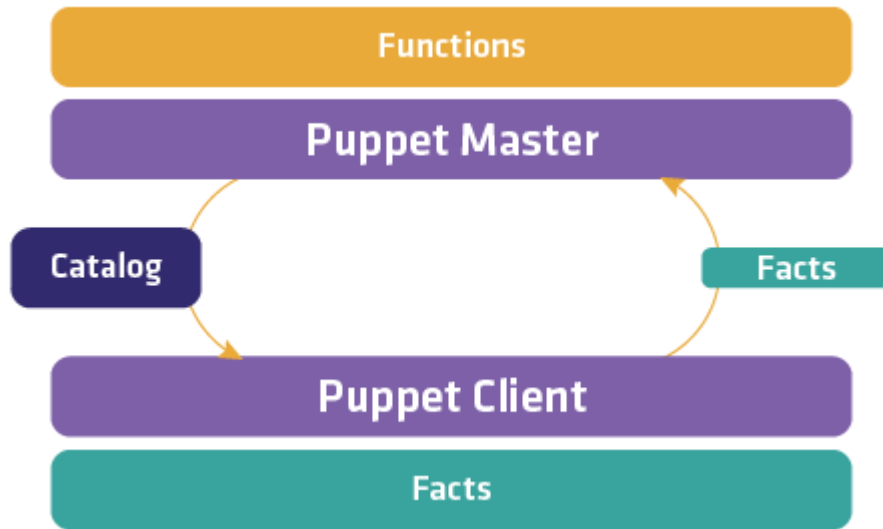
Notes:

Discussion questions:

- What sort of information might be appropriate to expose via custom facts?
- What factors might affect when the value of the fact is available?
- What sort of layout for your modules might you choose when developing custom facts?

Puppet Functions & Facts

Lifecycle of a Puppet Agent Run



Adding Functionality

Functions add functionality to the language.

- Functions run *during compilation*.
- Functions run on the Puppet Master.
- They are commonly used for:
 - interfacing with external tools.
 - providing additional functionality to the Puppet DSL.
- Example functions:
 - `include`
 - `template`
 - `regsubst`
 - `is_bool` (stdlib)

Notes:

- Functions are documented at <http://docs.puppetlabs.com/references/latest/function.html>
- More useful functions can be found at <http://forge.puppetlabs.com/puppetlabs/stdlib>
 - install on Puppet Open Source with `puppet module install puppetlabs/stdlib`
 - already installed on Puppet Enterprise

Types of Functions

A function is either a `:statement` or an `:rvalue`.

- `:statement`
 - Code that just executes.
 - May perform an action, such as raising a warning to be logged.
 - This is the default.
- `:rvalue`
 - Code that executes and returns a value.
 - Assigned to a variable.
 - Assigned to a resource parameter.
 - Used to choose a conditional branch.

Adding a Function

To create your own custom functions, add ruby files in:

`<modulepath>/<modulename>/lib/puppet/parser/functions`

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/custom/  
├── lib  
│   └── puppet  
│       └── parser  
│           └── functions  
│               └── mastername.rb ## $mastername = mastername()  
└──
```

- A custom `:statement` function:

```
Puppet::Parser::Functions.newfunction(:myfunc) do  
  # ...  
end
```

- A custom `:rvalue` function:

```
Puppet::Parser::Functions.newfunction(:myfunc2, :type => :rvalue) do  
  # ...  
end
```

Custom Function

Simple function that returns the hostname of the Puppet Master.

```
# mastername.rb
require 'socket'
module Puppet::Parser::Functions
  newfunction(:mastername, :type => :rvalue ) do
    Socket.gethostname.chomp
  end
end
```

The name of the function and the filename must match.

Handling Arguments

Arguments are passed as a single array

```
# basename.rb
require 'socket'
module Puppet::Parser::Functions
  newfunction(:basename, :type => :rvalue ) do |args|
    raise ArgumentError, 'Wrong number of arguments' if args.length != 1
    filename = args[0]
    File.basename filename
  end
end
```



Lab 2.2: Create a Custom Function

- **Objective:**

- Write a function to return the home directory of a user.

- **Steps:**

- Ensure you are in your module path.
- Create a new module that will contain your custom function.
- Create a new ruby file called `homedir.rb`.
- Install your function using `pluginsync`.
- Use your function in a manifest.

- **Hints:**

- Assume the typical Linux home directory layout.

Classification

Lesson 3: Classification

OBJECTIVES

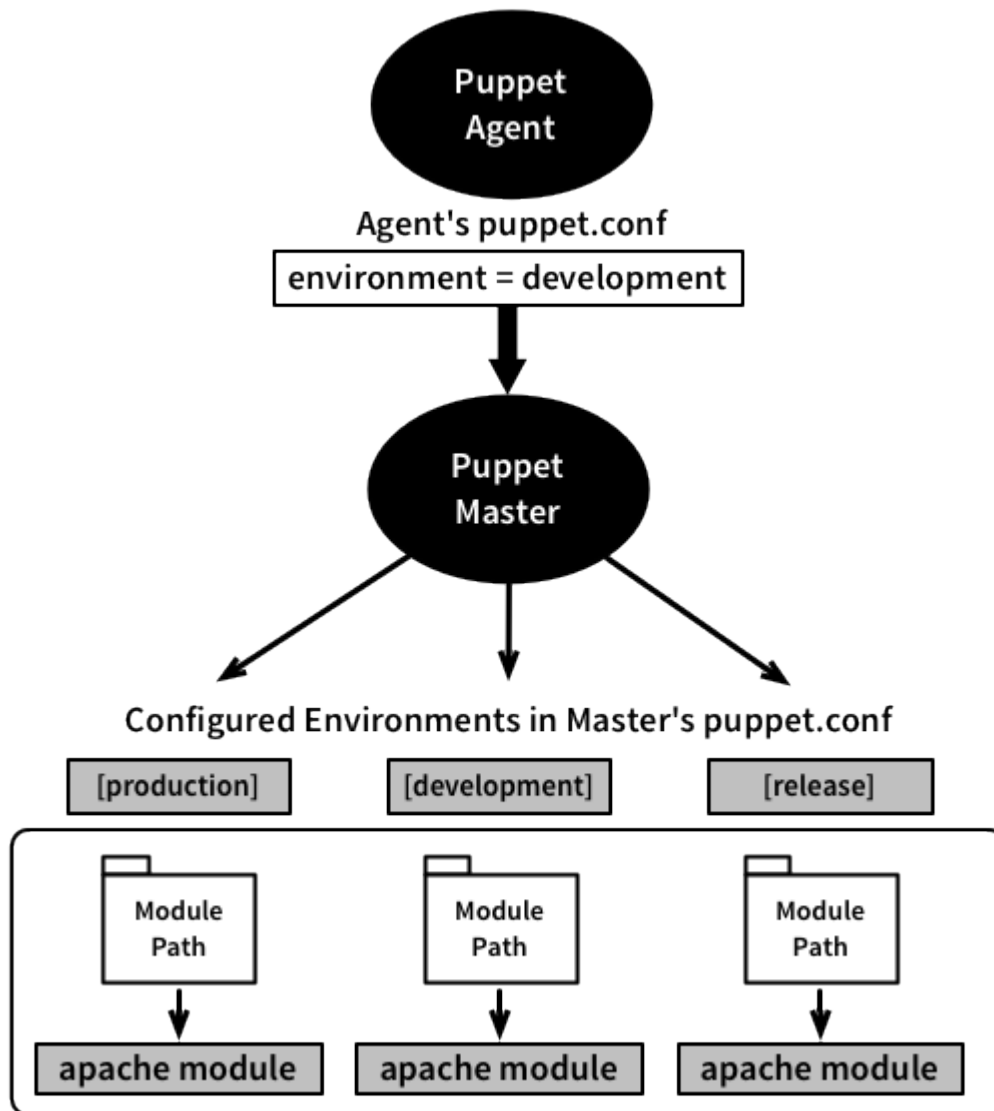
At the end of this lesson, you will be able to:

- Identify several classification methods.
- Create a new environment configuration on both master & agent.
- Specify a custom `modulepath` for each environment.
- Identify the role of an External Node Classifier.

Environments

- Allow you to specify multiple self contained zones on a single master
- Contains settings for:
 - `modulepath` -- where modules are located
 - `manifest` -- path to the `site.pp` main manifest.
- Environment naming is arbitrary
- Default environment is called Production
 - implicitly created
- Often set to track repository branches for `dev`, `test`, and `prod`

Environments



Environment Configuration

on the Puppet Master

```
#/etc/puppetlabs/puppet/puppet.conf
[production]
  manifest    = /etc/puppetlabs/puppet/environments/production/site.pp
  modulepath  = /etc/puppetlabs/puppet/environments/production/modules
[development]
  manifest    = /etc/puppetlabs/puppet/environments/development/site.pp
  modulepath  = /etc/puppetlabs/puppet/environments/development/modules
[test]
  manifest    = /etc/puppetlabs/puppet/environments/test/site.pp
  modulepath  = /etc/puppetlabs/puppet/environments/test/modules
```

- Catalogs compiled for each environment take these settings into account.

Notes:

For the future of puppet environment configuration: <http://projects.puppetlabs.com/issues/15770>.

Requesting an Environment

Agents request a catalog from a given environment

- Set the environment in `puppet.conf` on the agent

```
[agent]
environment = development
```

- Override the environment via the command line for a single agent run

```
[root@training ~]# puppet agent -t --environment test
```

Notes:

Declaring an environment on the command line will only apply to that particular puppet run, the environment will not be changed permanently.

Lab 3.1: Create a Development Environment



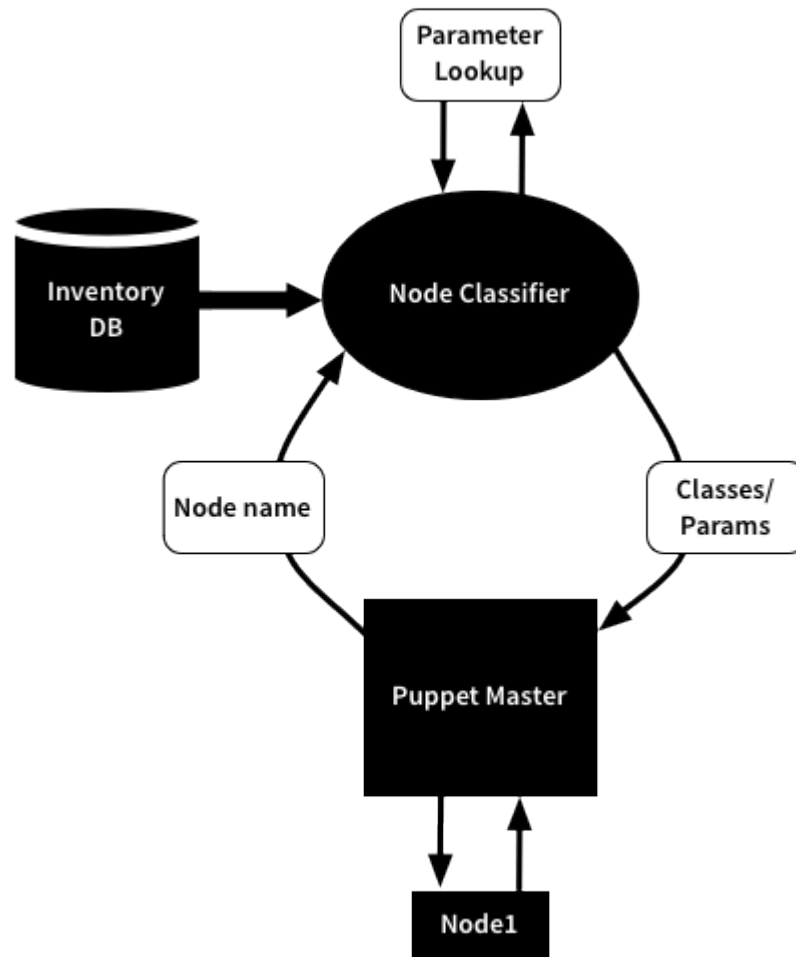
- **Objective:**

- Create a new 'development' environment for your master.

- **Steps:**

- Edit your `puppet.conf`.
- Add a new environment named `development` to the end of the file.
- Copy your standard `modulepath` to your development environment.
- Trigger a puppet run in your development environment.

External Node Classifier



exec node_terminus

```
# /etc/puppetlabs/puppet/puppet.conf on the puppetmaster
[master]
  node_terminus = exec
  external_nodes = /etc/puppetlabs/puppet-dashboard/external_node
```

- External Node Classifier scripts
 - executed with the agent's `$certname` as an argument
 - output YAML to `stdout`
 - can be written in any language

External Node Classifier Output

Example output

```
[root@training puppet-dashboard]# ./external_node somenode.puppetlabs.net
---
classes:
- common
- puppet
- dns
- ntp
parameters:
puppet_server: puppet.puppetlabs.net
dns_server: ns.puppetlabs.net
datacenter: slicehost
name: somenode.puppetlabs.net
```

equivalent to

```
$puppet_server = 'puppet.puppetlabs.net'
$dns_server    = 'ns.puppetlabs.net'
$datacenter    = 'slicehost'

node 'somenode.puppetlabs.net' {
  include common, puppet, dns, ntp
}
```

Advanced ENC Usage

- provide rich data types
- provide parameters for parameterized classes
- set agent environments

```
---
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
name: shells.example.com
```

In Puppet 3, ENC set environments became authoritative.

Notes:

For changes to ENC environment handling in puppet 3.0.X, visit <http://projects.puppetlabs.com/issues/3910>. ENCs override command line `--environment` arguments in puppet 3.0.X

ldap node_terminus

Use LDAP to classify your nodes

```
# /etc/puppetlabs/puppet/puppet.conf on the puppetmaster
[master]
  node_terminus = ldap
  ldapserver = ldap.example.com
  ldapbase = ou=hosts,dc=example,dc=com
```

ruby-ldap must be installed

```
[root@training ~]# /opt/puppet/bin/gem install ruby-ldap
```


LDAP Schema Extensions

```
dn: cn=web,ou=Hosts,dc=example,dc=com
objectClass: device
objectClass: ipHost
objectClass: puppetClient
objectClass: top
cn: web
environment: production
ipHostNumber: 192.168.0.1
description: Our webserver
puppetClass: common
puppetClass: lampstack
puppetVar: config_exim=true
puppetVar: trusted_users=lludwig,lak,joe
puppetVar: datacenter=slicehost
```

equivalent to

```
$config_exim    = 'true'
$trusted_users = [ 'lludwig', 'lak', 'joe' ]
$datacenter     = 'slicehost'

node 'web.example.com' {
  include common, lampstack
}
```

Exercise 3.2: Use the Puppet Enterprise ENC



- **Objective:**

- Review the Puppet Enterprise ENC.

- **Steps:**

- Execute the built-in `exec node_terminus` script that ships with Puppet Enterprise.
- Login to the enterprise console using a web browser.
- Browse to your node and add the class *databases*.
- Execute the `exec node_terminus` script again.

Advanced Coding Techniques

Lesson 4: Advanced Coding Techniques

OBJECTIVES

At the end of this lesson, you will be able to:

- Use resource defaults to reduce duplication.
- Use complex data structures like arrays and hashes.
- Identify techniques for code compression.
- Utilize class inheritance.
- Identify alternate ordering strategies.

Resource Defaults

Prevent redundant attribute specification.

- reference all resources of a given type in current scope
- provides default or starting parameters
- can be overridden as necessary

```
File {  
  owner => 'root',  
  group => 'root',  
  mode  => '0644',  
}  
Exec {  
  logoutput => true,  
}  
  
file { '/etc/motd':  
  ensure => file,  
  content => template('motd/motd.erb'),  
}  
exec { '/path/to/script1.sh' : }  
exec { '/path/to/script2.sh' : }  
exec { '/path/to/script3.sh' :  
  logoutput => false,  
}
```

Resource Relationship Abbreviation

- Require and Before can be expressed using <- and ->.
- Subscribe and Notify can be expressed using <~ and ~>.

```
package { 'httpd':  
  ensure => present,  
}  
service { 'httpd':  
  ensure => running,  
  enable => true,  
}  
Package['httpd'] -> Service['httpd']
```

-or-

```
package { 'httpd':  
  ensure => present,  
} ->  
service { 'httpd':  
  ensure => running,  
  enable => true,  
}
```

Chaining Dependencies

```
package {'openssh-server':  
  ensure => present,  
}  
file { ['/etc/sshd_config':  
  content => template("${module_name}/sshd_config.erb"),  
  require => Package['openssh-server'],  
}  
service { 'sshd':  
  ensure    => running,  
  enable    => true,  
  subscribe => File['/etc/sshd_config'],  
}
```

The dependencies can be written as:

```
Package['openssh-server'] -> File['/etc/sshd_config'] ~> Service['sshd']
```

Or as:

```
Service['sshd'] <~ File['/etc/sshd_config'] <- Package['openssh-server']
```

Anchor Pattern

Use anchor resources to 'contain' classes

- Classes contain all of the resource included
 - relationships to the class transfer to included resources
 - relationships do *not* transfer to included *classes*
- Use `anchor` resources to bookend included classes
 - manually contains classes

```
class container {  
  anchor { 'container:begin':  
    before => Class['ssh', 'fail2ban', 'nagios'],  
  }  
  
  include ssh  
  include fail2ban  
  include nagios  
  
  anchor { 'container:end':  
    require => Class['ssh', 'fail2ban', 'nagios'],  
  }  
}
```


Using Arrays for Resource Declaration

- Creates unique resources for each element of the array
- All resources have the same attributes other than title & namevar

```
# mkdir -p pattern
file { [ '/var/www', '/var/www/html', '/var/www/html/docroot' ]:
  ensure => directory,
}

# notice that we require only a single file resource, not the array
service { 'httpd':
  ensure => running,
  enable => true,
  require => File['/var/www/html/docroot'],
}

# ensure resources based on a variable--can be array or string!
$openssh_packages = $::osfamily {
  'debian' => 'ssh',
  'redhat' => ['openssh-server', 'openssh-client'],
}
package { $openssh_packages:
  ensure => installed,
}
```

Lab 4.1: Class Refactor



- **Objective:**

- Using resource defaults and arrays, refactor a sample class.

- **Steps:**

- Create a `databases::admins` class in your `website` module.
- Refactor the sample code from your exercise guide using arrays and resource defaults.

Storing Data in Hashes

- Also known as
 - associative arrays
 - dictionaries

```
$home_paths = {  
  brad => '/home/brad',  
  ralph => '/home/ralph'  
}  
  
file { $home_paths['brad'] :  
  ensure => directory,  
}
```

```
notice: /Stage[main]/File[/home/brad]/ensure: created  
notice: Finished catalog run in 0.04 seconds
```

Notes:

Hashes are available in Puppet 2.6.x and higher.

Complex Data Structures

Hashes can store arbitrary data types

- Even other hashes
- Nested arrays and hashes can be accessed by chaining indexes:

```
$web_configs = {  
  'kermit.puppetlabs.vm' => {  
    port    => 80,  
    docroot => '/var/www/kermit',  
  },  
  'elmo.puppetlabs.vm'   => {  
    port    => 80,  
    docroot => '/var/www/elmo',  
  },  
}  
  
file { $web_configs['kermit.puppetlabs.vm']['docroot']:  
  ensure => directory,  
}
```

```
notice: /Stage[main]/File[/var/www/kermit]/ensure: created  
notice: Finished catalog run in 0.02 seconds
```

Create Resources Function

Instantiates all resources described by a hash

- Native resources
- Defined resources
- Classes

```
$user_hash = {  
  'ralph' => { uid    => '1330',  
               group  => 'allstaff',  
               groups => ['developers', 'operations', 'release'], }  
  'brad'  => { uid    => '1308',  
               group  => 'allstaff',  
               groups => ['developers', 'prosvc', 'release'], }  
}  
  
create_resources('user', $user_hash)
```

```
notice: /Stage[main]//User[brad]/ensure: created  
notice: /Stage[main]//User[ralph]/ensure: created  
notice: Finished catalog run in 0.16 seconds
```

Notes:

The `create_resources` function does not support virtual or exported resource creation in puppet 2.7.X.

Create Resources Function

Instantiates all resources described by a hash

- Native resources
- Defined resources
- Classes

Can also specify resource defaults

```
$resource_defaults = {  
  'ensure' => 'present',  
  'provider' => 'ldap',  
}  
  
create_resources(user, $user_hash, $resource_defaults)
```

Lab 4.2: Using data in hashes



- **Objective:**

- Instantiate user resources based on data stored in a hash.

- **Steps:**

- Refactor your `databases::admins` class to define users in a hash.
- You may customize parameters for each user.
- Use `create_resources()` to manage these users.

Defined Resource Types

Unlike classes, defined types can be declared multiple times.

```
define company::employee(  
  $location = 'corp',  
  $home_server = '/mnt/corp_server',)  
{  
  user { $name:  
    home => "${home_server}/${name}",  
    tag  => $location,  
  }  
}
```

```
company::employee { 'luke': }  
company::employee { 'james': }  
  
company::employee { 'ralph':  
  location    => 'Denver',  
  home_server => '/mnt/denver',  
}
```

Always use `$name` to derive resource titles within a defined type.

Explicitly Managing Resources

Using the `host` resource type you can specify host entries.

```
host { 'localhost':  
  ensure      => 'present',  
  host_aliases => 'localhost.localdomain',  
  ip          => '127.0.0.1',  
}  
host { 'kermit.puppetlabs.com':  
  ensure      => present,  
  host_aliases => 'kermit',  
  ip          => '172.16.238.131',  
}  
host { 'piggy.puppetlabs.com':  
  ensure      => present,  
  host_aliases => ['piggy', 'missy'],  
  ip          => '172.16.238.132',  
}  
host { 'oscar.puppetlabs.com':  
  ensure => absent,  
}
```

But what if you only want to have explicitly declared host entries?

Purge Example

This will purge all unspecified host resources.

```
resources { 'host':  
  purge => true,  
}
```

```
# HEADER: This file was autogenerated at Thu Apr 18 11:48:58 +0000 2013  
# HEADER: by puppet. While it can still be managed manually, it  
# HEADER: is definitely not recommended.  
127.0.0.1    localhost    localhost.localdomain  
172.16.238.131  kermit.puppetlabs.com kermit  
172.16.238.132  piggy.puppetlabs.com piggy missy  
172.16.238.135  animal.puppetlabs.com animal
```

```
notice: /Host[animal.puppetlabs.com]/ensure: removed  
notice: Finished catalog run in 0.26 seconds
```

Resources Resource

```
resources { 'user':  
  purge          => true,  
  unless_system_user => true,  
}
```

- `name`: the name of the resource type that is to be managed. (`namevar`)
- `purge`: true or false
- `unless_system_user`: true, false, or some upper uid limit specified as an integer.

Lab 4.3: Use the Resources Resource



- **Objective:**

- Use the `resources` resource to purge host records.

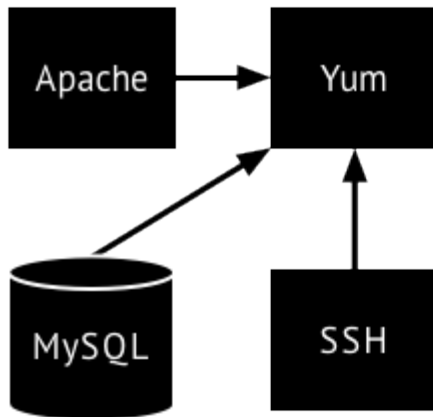
- **Steps:**

- Create a new module named `hosts`.
- Gather the current hosts using `puppet resource`.
- Add a `resources` resource to your code.
- Create and apply your smoke test.
- Re-run puppet to purge your change.

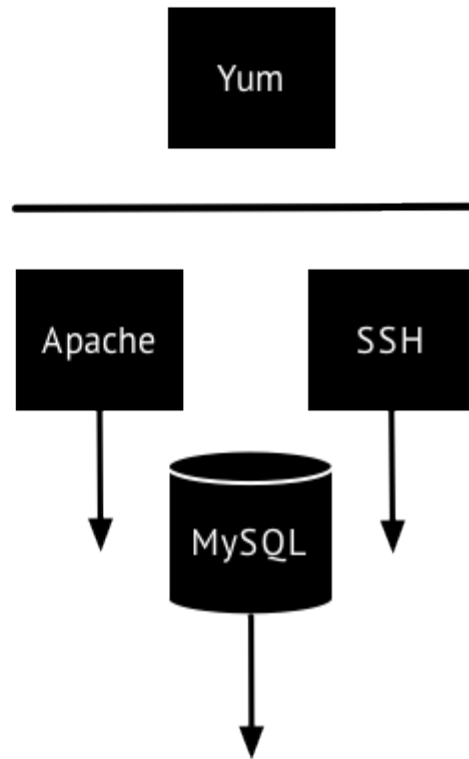
Run Stages Overview

- Allow for specific ordering of a class during runtime.
- Are declared as resources. Use existing relationship syntax.

WITHOUT RUN STAGES



WITH RUN STAGES



Declaring the Run Stages Resource

- You declare them like any other resource.
- There is always an implied stage called `main`.
- Stage `main` is the default stage for all classes.
- Can use the `before` and `require` metaparameters to create stage ordering. Their values are references to other stages.

```
class stages {  
  stage { 'before': before => Stage['main'] }  
  stage { 'after': require => Stage['main'] }  
}
```

Run Stages in Action

- You must use parameterized classes.
- Only entire classes can be put in a run stage.

```
class stages {  
  stage { [ 'before', 'after']: }  
  Stage['before'] -> Stage['main'] -> Stage['after']  
}
```

```
class webserver {  
  include stages  
  include packages # Gets Stage['main'] by default  
  
  class { 'yum': stage => before }  
  
  class { 'apache': stage => after }  
}
```

Lab 4.4: Run Stages



- **Objective:**

- Use a run stage to ensure ordering of classes.

- **Steps:**

- Create run stages titled `pre-run` and `post-run`
- Use dependencies to ensure that they are enforced before and after the `main` stage.
- Create a class to ensure that the `epel` Yum repository is enabled and place it in the `pre-run` stage.
- Create a class with an `updatedb` `exec` resource and place it in the `post-run` stage.

Parameterized Classes Review

An example parameterized class definition:

```
class elmo_app (
  $application_dir,
  $options = 'Indexes MultiViews',
  $port    = '80',
  $dbuser  = 'elmo',
  $dbpass  = 'password'
) {

  include apache
  include mysql

  apache::vhost { 'elmo':
    port      => $port,
    docroot   => $application_dir,
    options   => $options,
    notify    => Service['httpd'],
  }

  mysql::db { 'elmo':
    user       => $dbuser,
    password   => $dbpass,
    host       => $::hostname,
    grant      => ['all'],
  }

  #...
}
```

Declaring Parameterized Classes

Class Definition:

```
class elmo_app (  
  $application_dir,          # no default, so required parameter  
  $options = 'Indexes MultiViews',  
  $port     = '80',  
  $dbuser   = 'elmo',  
  $dbpass   = 'password'  
) {  
  #...
```

Declaring a class and specifying parameter(s):

```
class { 'elmo_app':  
  application_dir => '/opt/web',  
  port           => '80',  
}
```

If no parameters are required (all have defaults):

```
include apache  
# -- or --  
class { 'apache': }
```

Include Function vs Class Resource

Include Function

The `include` function is idempotent and can be used repeatedly:

```
include apache
include apache
# valid code and the class is only included once
```

Class Resource

As the title is not unique, this causes duplicate resource error:

```
class {'apache':}
class {'apache':}
# This will result in an error
```

Lab 4.5: Parameterized Classes



- **Objective:**

- Create a module that allows you to serve a Wordpress blog from a specified `docroot`.

- **Steps:**

- Install the `puppetlabs/apache` module.
- Install the `hunner/wordpress` module.
- Create a `website` module that accepts a `docroot` parameter.
- Use the `apache::vhost` type to create a default virtual host for your Wordpress instance.
- Instantiate a Wordpress instance in that `docroot`.

Inherited Classes

Class inheritance allows for specialization.

```
class editors {  
  package { 'vim-common':  
    ensure => present,  
  }  
}
```

```
class editors::old inherits editors {  
  package { 'emacs-common':  
    ensure => present,  
  }  
}
```

```
include editors          # installs vim-common package  
include editors::old    # installs both vim-common & emacs-common package
```

Overriding Attributes

Override resource attributes to provide alternate configuration:

```
class parent {
  $somecontent = 'Parent'
  file { ['/tmp/class_inheritance.txt']:
    content => $parent::somecontent,
  }
}

class child inherits parent {
  $somecontent = 'Child'
  File['/tmp/class_inheritance.txt'] {
    content => $child::somecontent,
  }
}

include child
```

What is the content of `/tmp/class_inheritance.txt`?

Class Inheritance

Adding resources and attributes

```
class ssh {  
  package { 'openssh':  
    ensure => present,  
  }  
  
  file { ['/etc/ssh/ssh_config':  
    owner   => 'root',  
    group   => 'root',  
    mode    => '0644',  
    source  => 'puppet:///modules/ssh/ssh_config',  
    require => Package['openssh-clients'],  
  ]  
  
  file { ['/etc/ssh/sshd_config':  
    owner   => 'root',  
    group   => 'root',  
    mode    => '0644',  
    source  => 'puppet:///modules/ssh/sshd_config',  
    require => Package['openssh-server'],  
  ]  
  
  service { 'sshd':  
    ensure => running,  
    enable => true,  
  }  
}
```

Class Inheritance

Adding resources and attributes

- Inherit all resources from the `ssh` class and add new resources
- Add a new `require` relationship to `Service['sshd']`
- Append a new `subscribe` relationship to `Service['sshd']`

```
class ssh::paranoid inherits ssh {  
  package { 'fail2ban':  
    ensure => present,  
  }  
  
  sshkey { 'trustedhost.example.com':  
    ensure => present,  
    key    => 'puppet:///modules/ssh/trustedhost.pub',  
  }  
  
  File['/etc/ssh/sshd_config'] {  
    source => 'puppet:///modules/ssh/sshd_config_paranoid',  
  }  
  
  Service['sshd'] {  
    require => Package['fail2ban'],  
    subscribe +> sshkey['trustedhost.example.com'],  
  }  
}
```


Class Inheritance

- Inheritance should be used within a module to reduce repetition

```
[root@training ~]# tree /etc/puppetlabs/puppet/modules/ssh
├── manifests
│   ├── init.pp          ## class ssh {}
│   ├── paranoid.pp      ## class ssh::paranoid inherits ssh {}
│   └── paranoid
│       └── solaris.pp   ## class ssh::paranoid::solaris inherits ssh::paranoid {}
└── tests
    ├── init.pp          ## include ssh
    ├── paranoid.pp      ## include ssh::paranoid
    └── paranoid
        └── solaris.pp   ## include ssh::paranoid::solaris
```

- Inheriting classes from other modules decreases modularity
 - increases coupling
 - increases chance of breakage

```
# bad practice
class wordpress inherits apache { ... }
```

Using \$module_name

```
define apache::vhost(
  $port ,
){
  if $caller_module_name {
    $conf_template = template("${caller_module_name}/vhost.erb")
  }
  else {
    $conf_template = template("${module_name}/vhost.erb")
  }
  file { ["/etc/http/conf.d/${title}.conf":
    ensure => file,
    content => $conf_template,
  ]
}
```

```
class wordpress {
  apache::vhost { 'kermit.puppetlabs.vm':
    port => '8080',
  }
}
# Pulls template from <modulepath>/wordpress/templates
```

```
class apache {
  apache::vhost { 'elmo.puppetlabs.vm':
    port => '8080',
  }
}
# Pulls template from <modulepath>/apache/templates
```

Notes:

Please see http://docs.puppetlabs.com/puppet/3/reference/lang_variables.html#facts-and-built-in-variables for more built in variables.

Lab 4.6: Inherited Classes



- **Objective:**

- Create a base `webapp` class and inherit more specific subclasses.

- **Steps:**

- Create a `webapp` module and class that manage common configurations.
- Create a `webapp::wordpress` class that inherits from `webapp`.
- Refactor your existing `wordpress` module to fit into this layered strategy.

Roles and Profiles

Lesson 5: Roles and Profiles

OBJECTIVES

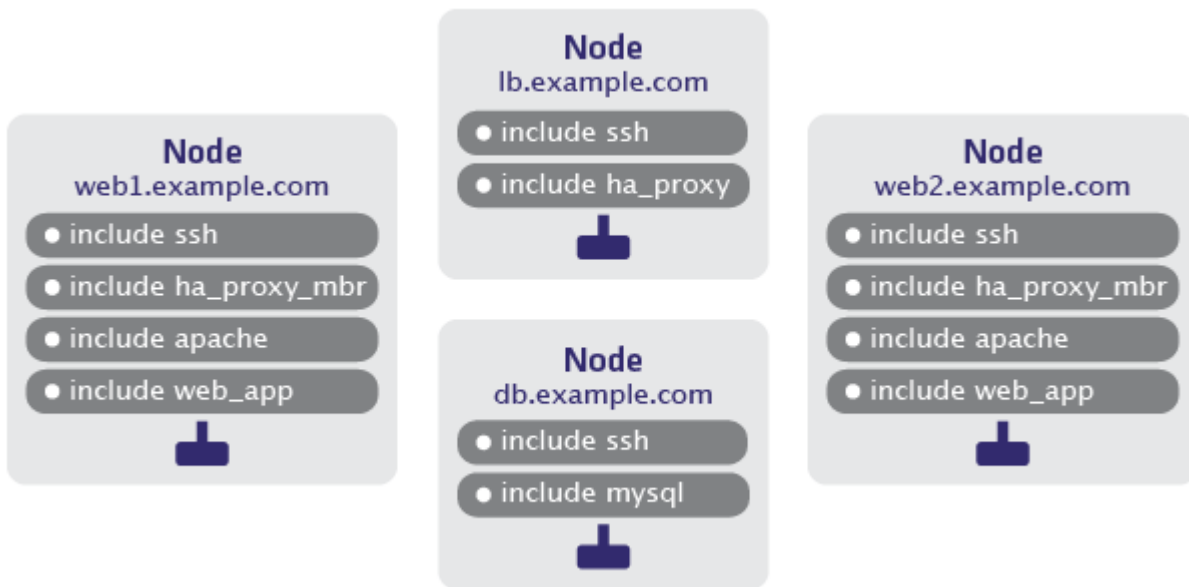
At the end of this lesson, you will be able to:

- Identify drawbacks to node level logic.
- Differentiate between business logic and implementation.
- Use Hiera and the ENC to efficiently abstract layers of the stack.
- Create appropriate role and profile classes.

Modular Design

Typical (abbreviated) workflow

1. Develop great modules.
2. Classify them to nodes.



Evolving Infrastructure

Business requirements always change

- Infrastructure grows
- Company pivots to serve differing markets
- Must respond quickly
- More and more edge cases
- Eventually code feels bulky and high maintenance

Time to refactor!

Danger Signs



- Resources declared in multiple modules
- You find yourself wondering where your implementation *fits*
- Copious amount of logic at a node level
- Repetition and duplication
- The `if` statement is your *go-to-guy*

Node Level Logic

Looks reasonable?

```
node 'basil.puppetlabs.vm' {  
  class { 'apache':  
    version => 'latest',  
  }  
  class { 'motd': }  
  class { 'ssh': }  
  if $::operatingsystem == 'solaris' {  
    class { 'users':  
      default_shell => '/bin/false',  
    }  
  }  
  else {  
    class { 'users': }  
  }  
  Class['ssh'] -> Class['users']  
}
```

UNTIL YOU HAVE TO MAINTAIN IT!

Node Level Logic

Looks reasonable?

Until you have to maintain it!

- What happens when you must manage 1,000 nodes?
 - or 10,000 nodes!!
- That's a lot of code and of node definitions.
- Judicious use of an ENC can alleviate some of the pain.
 - abstraction layers expose only coherent interfaces
 - code is versionable
- So where should you implement this?

Good Design

Appropriate Abstraction Layers

- Good modules should:
 - only manage own resources
 - be granular and portable
 - avoid implementation details
- Good architecture should:
 - provide business logic to classification
 - provide an abstraction layer for implementation of components
 - make code adaptable to complex requirements
 - reduce node-level logic
 - reduce functionality overlap

Implementation Layer

Business logic does not align with technology.

- Break everything down into components.
- Think about what things actually *are* instead of just what they look like.
- Look for overlap and similarities in application stacks.
- Reduce each application into granular Puppet modules.
- Create a code layer responsible for implementation.

Let's call these **profiles**.

Implementation Layer

Here's a start

```
class profiles::x {
  include tomcat
  include mysql
  include componenta
  include componentb
  componentb::resource { 'name':
    ensure => present,
  }
}
class profiles::y {
  include tomcat
  include mysql
  include componenta
  include componentc
  include componentd
}
class profiles::z {
  include tomcat
  include mysql
  include componenta
  include componentb
  include componentd
  include dependency
  Class['dependency'] -> Class['componentd']
}
```

Component Stack

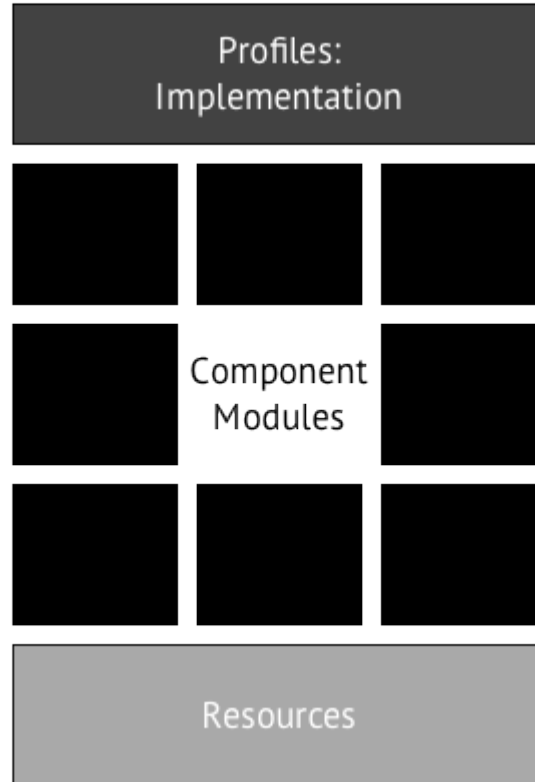
Use inheritance for abstraction

```
class profiles::application {
  include tomcat
  include mysql
  include componenta
}
class profiles::application::x inherits profiles::application {
  include componentb
  componentb::resource { 'name':
    ensure => present,
  }
}
class profiles::application::y inherits profiles::application {
  include componentc
  include componentd
}
class profiles::application::z inherits profiles::application {
  include componentb
  include componentd
  include dependency
  Class['dependency'] -> Class['componentd']
}
```

Notes:

Many people choose to use composition rather than inheritance. In that case, each application profile would simply include the `profiles::application` class.

Component Stack



Notes:

- Components should be named after what they manage (`apache`, `ssh`, `mysql`).
- Profiles should be named after the logical stack they implement (`database`, `bastion`, `email`).

Abstraction Layers

Puppet is all about abstraction.

- Data is abstracted by Hiera.
- Providers are abstracted by types.
- Resources are abstracted by classes.
- Classes are abstracted by modules.

Abstraction Layers

Puppet is all about abstraction.

- Data is abstracted by Hiera.
- Providers are abstracted by types.
- Resources are abstracted by classes.
- Classes are abstracted by modules.
- **Modules are abstracted by profiles.**

Lab 5.1: Designing Profiles



- **Objective:**

- Design profile classes for a WordPress stack, an email server, a nagios monitor, and a log monitor.

- **Steps:**

- Design component modules for `wordpress`, `mysql`, `apache`, `php`, `exim`, `nagios`, `splunk`, **etc.**
 - Don't write any real logic; just architect the layers.
- Write profile classes that include each component as needed.

Business Logic

Next layer of abstraction

- We've designed a technology stack
 - Components and implementation details
- Now we need a layer for business logic
 - Allow the business to manage how the infrastructure *looks* without defining *what it is*
 - Choose technology stacks (profiles) to include
 - Don't expose unnecessary details

Notes:

Remember to keep your audience in mind when developing abstraction layers.

Roles

Represent business logic, not technology

- Roles are classes that:
 - define a set of technology stacks (profiles) that make up a logical role
 - include as many profiles as required to define itself
 - abstract the business role from the implementation details
 - contain no logic at all

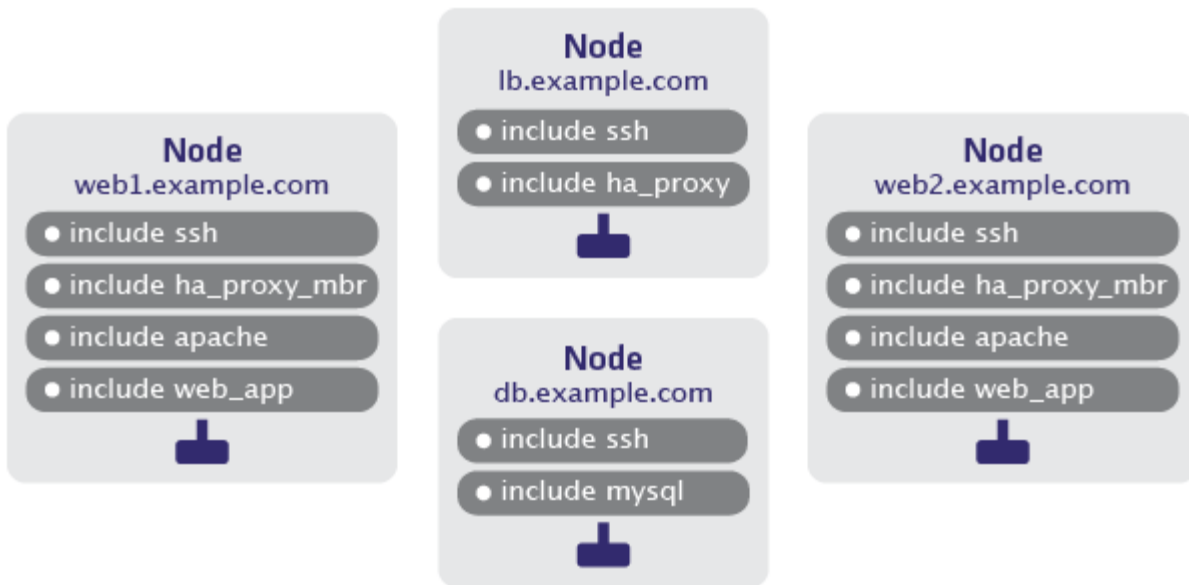
Roles *only* implement profiles.

```
class role::webapp {  
  include profiles::base  
  include profiles::customapp  
  include profiles::test_tools  
}
```

Mapping Nodes to Roles

A node can only have one role.

- If a node requires two roles, it has by definition become a new role.
- A single role can be applied to many nodes, however.

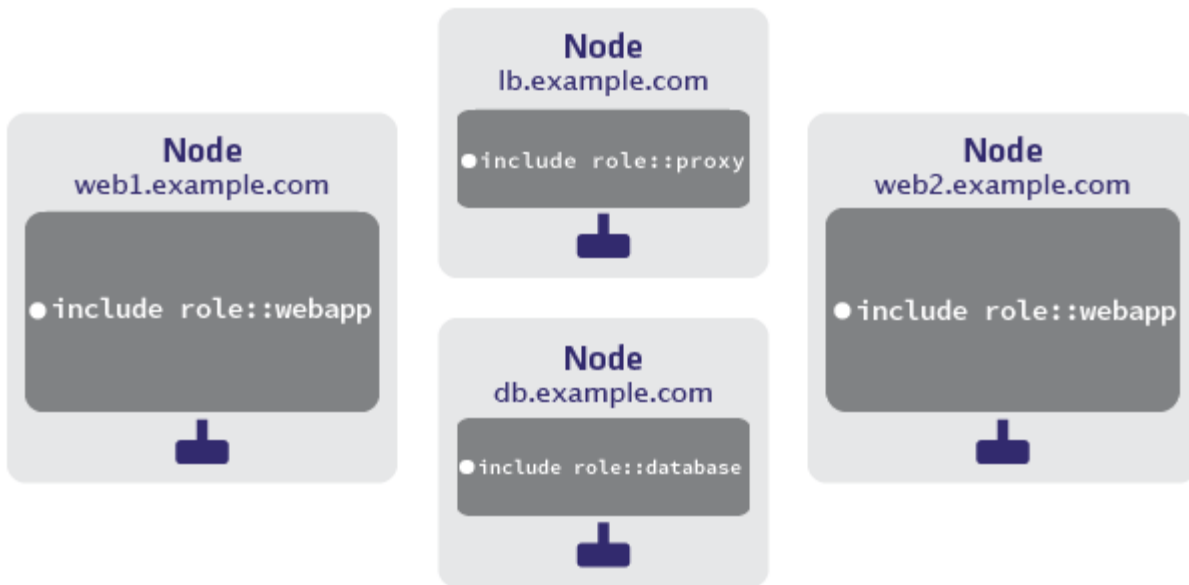


Legacy Granular Classification

Mapping Nodes to Roles

A node can only have one role.

- If a node requires two roles, it has by definition become a new role.
- A single role can be applied to many nodes, however.



Role Classification

Classifying Nodes

Simply assign a roles to each node

- Roles define the logical role a node should play
 - select profiles that make up a logical role
 - expose no implementation details

```
node 'craig.puppetlabs.vm' {  
  include role::uat_server  
}
```

Classifying Nodes

Simply assign a roles to each node

- Roles define the logical role a node should play
 - select profiles that make up a logical role
 - expose no implementation details

Edit node

Node
craig.puppetlabs.vm

Description
Enter a description for this node here...

Parameters

| key | value |
|-----|-------|
| | |

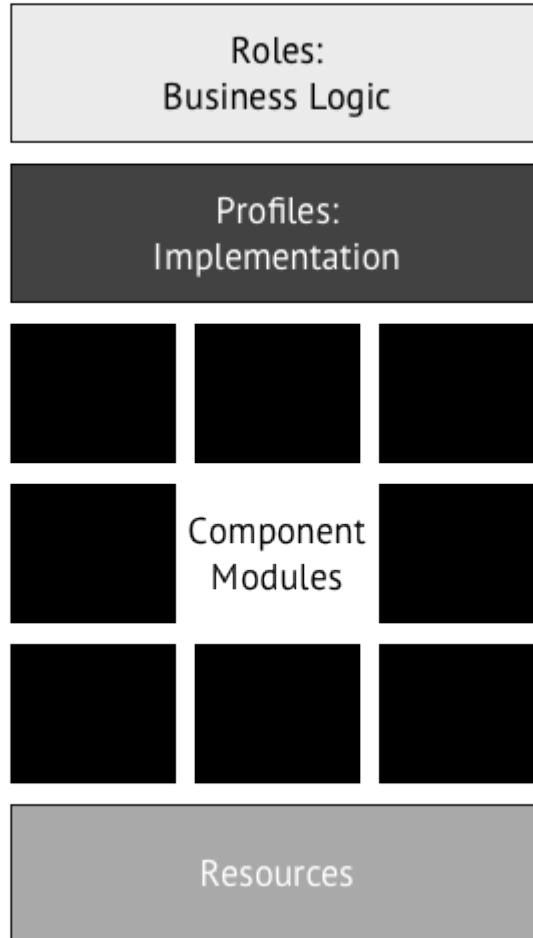
[Add parameter](#)

Classes
role::uat_server x

Groups

[Save changes](#) or [Cancel](#)

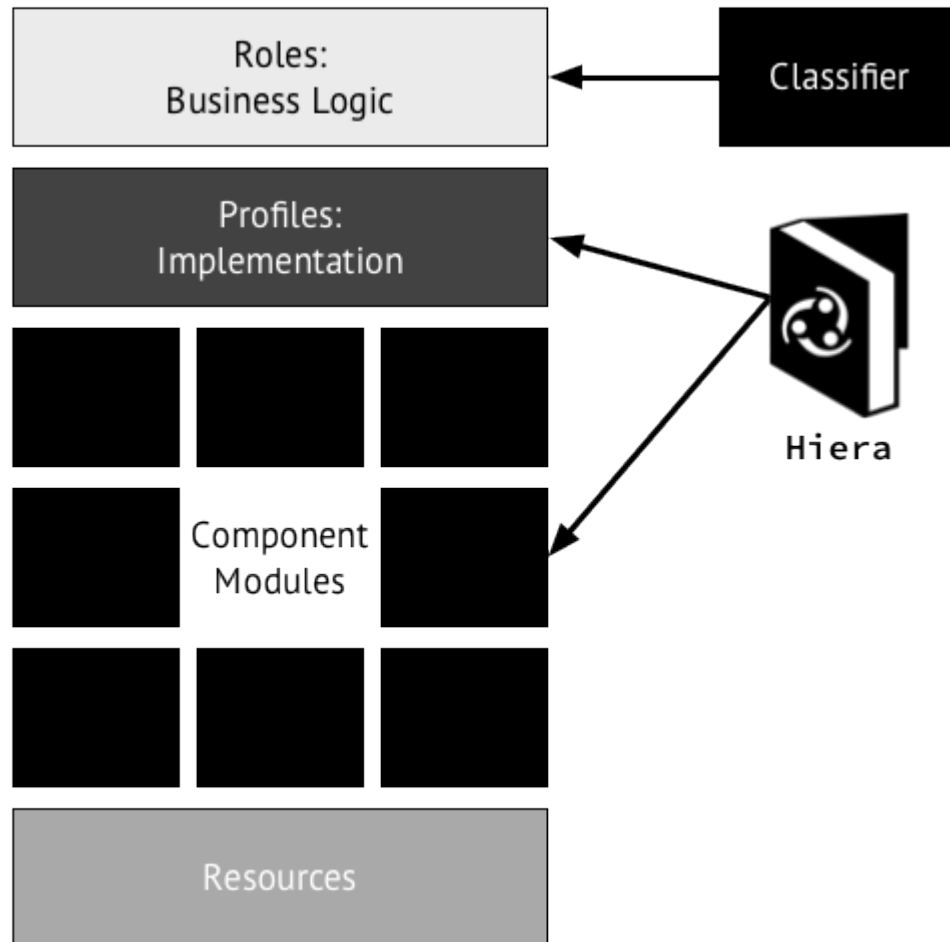
Business Logic Stack



Notes:

- Components should be named after what they manage (`apache`, `ssh`, `mysql`)
- Profiles should be named after the logical stack they implement (`database`, `bastion`, `email`)
- Roles should be named in business logic convention (`uat_server`, `web_cluster`, `application`, `archive`)

Business Logic Stack



Notes:

- Components should be named after what they manage (`apache`, `ssh`, `mysql`)
- Profiles should be named after the logical stack they implement (`database`, `bastion`, `email`)
- Roles should be named in business logic convention (`uat_server`, `web_cluster`, `application`, `archive`)

Key Benefits

Roles and Profiles

- Reduced node-level logic to a role.
 - Gain the ability to be flexible with implementation.
 - Business logic improves manageability by non-Puppet users.
 - Edge cases are now easy to solve.
-

Notes:

A more complete writeup can be found at <http://www.craigdunn.org/2012/05/239/>

Lab 5.2: Designing Roles



- **Objective:**

- Design roles for the profile classes from the previous lab.

- **Steps:**

- Remember that roles can contain no logic, merely include profiles.
- Design complete roles that contain all profiles required to fully define itself.

Best Practices

Lesson 6: Best Practices

OBJECTIVES

At the end of this lesson, you will be able to:

- Describe several best practices.
- Identify the Puppet Labs style guide.
- Apply appropriate syntax from the style guide.

Logical Grouping vs. Resource Grouping

Resources should be grouped by logical relationship to each other

```
# OpenSSH
package { 'openssh':
  ensure => present,
}
file { ['/etc/ssh/sshd_config':
  ensure  => file,
  require => Package['openssh'],
  #...
}
service { 'sshd':
  ensure  => running,
  require => File['/etc/ssh/sshd_config'],
}

# Apache2
package { 'httpd':
  ensure => present,
}
file { ['/etc/httpd/conf.d/httpd.conf':
  ensure  => file,
  require => Package['httpd'],
  #...
}
service { 'httpd':
  ensure  => running,
  require => File['/etc/httpd/conf.d/httpd.conf'],
}
```

Notes:

Best practices dictate that within a given manifest, resources should be grouped by logical relationship to each other, rather than by resource type. Use of the semicolon syntax to declare multiple resources within a single set of curly braces is not recommended, except in the rare cases where it would improve readability.

Explicitly Ensure Files

File resources should explicitly ensure their state

- Explicitly ensure a value such as `file` instead of `present`.

```
file { ['/etc/motd':  
  ensure => file,  
  content => 'This system is owned be Example Corp.',  
}  
  
file { ['/usr/local/foo':  
  ensure => directory,  
}  
  
file { ['/usr/local/foo/syslog':  
  ensure => link,  
  target => '/var/log/messages',  
}
```

Notes:

- The `present` state matches when a file is a `file`, a `directory`, or a `link`.
- You should never use the deprecated practice of specifying a `symlink` target as the `ensure` value.

Code Defensively

Catch unexpected conditions before they break things

- Better to fail compilation than to enforce unpredictable configuration
- Case statements and selectors should have default cases.

```
case $::operatingsystem {
  'centos' : { $version = '1.2.3' }
  'solaris': { $version = '3.2.1' }
  default : { fail("Unsupported platform ${::operatingsystem}") }
}
```

- Validate class or type parameters and fail if any are invalid

```
class myservice($ensure='running') {
  if ! member([ 'running', 'stopped' ], !ensure) {
    fail('ensure parameter must be running or stopped')
  }
  # ...
}
```

- Expect `exec` resources to fail and set dependencies appropriately

Notes:

- Default case should fail the catalog compilation if behavior cannot be predicted on the majority of platforms.
- Even if you want "do nothing" as the default case, explicitly include that.
- For selectors, default selections should always be included, unless catalog compilation needs to fail when no value matches.

Style Guide

Not just for code

An example you might find at a college or publishing house:

Style Guide: A style guide allows multiple content publishers to render their text, images, and colors consistently in order to minimize distractions to the reader. The two best-known styles for trade publications are Associated Press (AP) and Chicago Manual of Style. Like most institutions, internal preferences lead to a hybrid style that may take from both AP and Chicago. We expect submissions to follow the Chicago Manual of Style.

- Adhering to a style guide:
 - Increases communication between teams and members.
 - Makes errors more readily discoverable.
 - Makes complex code more consumable by others.
 - Makes it easier to immerse yourself in your own dormant code.

Best Practice Resources

- Puppet Labs Style Guide
 - full of useful concepts to keep your code intelligible
http://docs.puppetlabs.com/guides/style_guide.html
- Puppet Labs Documentation
 - credible information on everything Puppet
<http://docs.puppetlabs.com/>
- RodJek's puppet-lint
 - check that your Puppet manifest conform to the style guide
<http://puppet-lint.com>
- puppet parser validate automated tests
 - syntax checking verifying puppet code
<http://puppetlabs.com/blog/verifying-puppet-checking-syntax-and-writing-automated-tests/>

File Manipulation

Lesson 7: File Manipulation

OBJECTIVES

At the end of this lesson, you will be able to:

- Identify several techniques for managing parts of files as resources.
- Iteratively create configuration files out of component pieces.
- Interact with configuration settings contained in files.

file_line Resource

Included with puppetlabs/stdlib module v2.1.x +

```
# Configure tools to use local proxy server
file_line { ['/etc/bashrc':
  ensure => present,
  path    => '/etc/bashrc',
  line    => 'export HTTP_PROXY=http://squid.puppetlabs.vm:3128',
}
```

```
[root@training ~]# cat /etc/bashrc
# ...
# File truncated for slide
unset i
unset pathmunge
export HTTP_PROXY=http://squid.puppetlabs.vm:3128
```

Notes:

- Previously the `file_line` type was named `whole_line`

`file_line` Parameters

- `ensure` - The basic property that the resource should be in. Valid values are `present` or `absent`.
- `line` - The line to be appended to the file located by the `path` parameter.
- `name` - An arbitrary name used as the identity of the resource.
- `path` - The file Puppet will ensure contains the line specified by the `line` parameter.

Available in `stdlib` 3.0.0 and higher

- `match` - An optional regular expression to run against existing lines in the file. If a match is found, the line is replaced rather than adding a new line.

File Fragments Using Concat

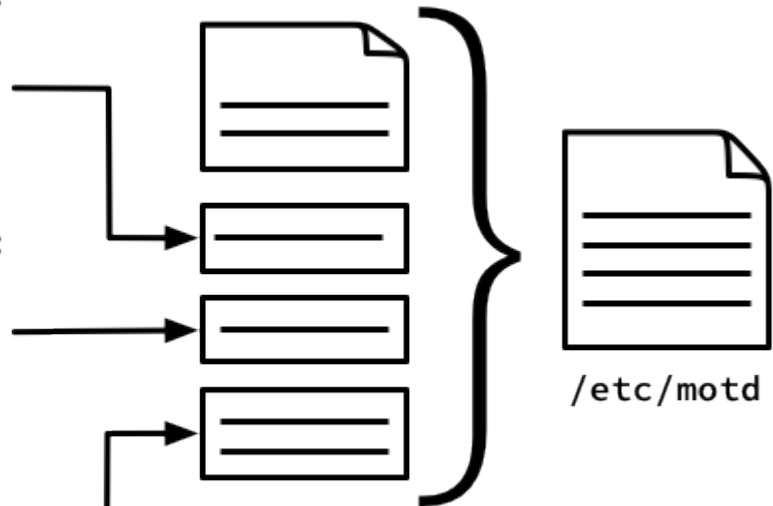
Order driven concatenation of lines to create a file.

```
$motd = '/etc/motd'
```

```
concat::fragment{'foo':  
  target => $motd,  
  content => 'foo',  
  order  => '01',  
}
```

```
concat::fragment{'bar':  
  target => $motd,  
  content => 'bar',  
  order  => '02',  
}
```

```
[ order => '03' ]  
[ order => '04' ]
```



ripienaar/puppet-concat Module

Order driven concatenation of lines to create a file.

```
$zone_file = '/var/named/puppetlabs.vm.zone'

concat{$zone_file:
  owner => root,
  group => root,
  mode  => 644
}

concat::fragment{'puppetlabs_zone_header':
  target => $zone_file,
  content => template('dns/zone_header.rb'),
  order  => 01,
}

concat::fragment{'www_a_record':
  target => $zone_file,
  content => 'www      IN  A      192.168.0.1',
  order  => 10,
}

concat::fragment{'ftp_a_record':
  target => $zone_file,
  content => 'ftp      IN  A      192.168.0.2',
  order  => 10,
}
```

Lab 7.1: Managing File Content



- **Objective:**

- Use the `file_line` resource to ensure that only `root` can run cron jobs.
- Use `concat` to iteratively build an `/etc/motd` file.

- **Steps:**

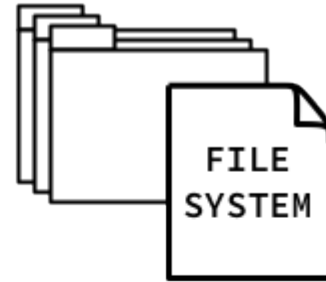
- Add entries to `/etc/cron.allow` and `/etc/cron.deny` to control cron access.
- Create a common header for `/etc/motd` using a `concat::fragment`
- Allow other classes to append information to your motd with fragments

- **Hint:**

- `/etc/cron.allow` does not exist by default

Augeas

Configuration Editing Tool



```

/files
/etc
  /puppetlabs
    /puppet
      /puppet.conf
        /main
          [main]
            user = pe-puppet
          /master
            [master]
              certname = master.puppetlabs.vm
          /agent
            [agent]
              server = master.puppetlabs.vm

```

```

/etc
  /puppetlabs
    /puppet
      /puppet.conf
        [main]
          user = pe-puppet
        [master]
          certname = master.puppetlabs.vm
        [agent]
          server = master.puppetlabs.vm

```

Augeas Resource Type

Configuration File Editing Tool

```
# /etc/yum.conf
...
metadata_expire=1h
installonly_limit = 5
# PUT YOUR REPOS HERE OR IN separate files named file.repo
# in /etc/yum.repos.d
```

Using the `augtool` command line utility to find node path:

```
[root@training ~]# /opt/puppet/bin/augtool
augtool> ls /files/etc/yum.conf/main
...
metadata_expire = 1h
installonly_limit = 5
```

Augeas Resource Type

Configuration File Editing Tool

Using the `augeas` resource type to change a value:

```
augeas { 'yum.conf':  
  context => '/files/etc/yum.conf/main',  
  changes => 'set proxy http://squid.puppetlabs.vm:3128'  
}
```

```
# /etc/yum.conf  
...  
metadata_expire=1h  
installonly_limit = 5  
# PUT YOUR REPOS HERE OR IN separate files named file.repo  
# in /etc/yum.repos.d  
proxy=http://squid.puppetlabs.vm:3128
```



Lab 7.2: Using Augeas

- **Objective:**

- Use the `augeas` resource to modify `krb5.conf`'s `default_realm` value.

- **Steps:**

- Modify the `kerberos` module you created previously.
- Create a definition for class `kerberos` in your new module.
- Add a new `augeas` resource type to your class.

- **Extra Credit:**

- Modify the custom fact you wrote to use the ruby `augeas` libraries to gather the fact

Data Separation

Lesson 8: Data Separation with Hieradata

OBJECTIVES

At the end of this lesson, you will be able to:

- Describe the single source of truth design pattern.
- Identify the data abstraction capabilities of Hieradata.
- Identify available Hieradata functionality in Puppet.
- Configure and use Hieradata.
- Describe the role of automatic data bindings in Puppet 3.0+.

The Problem

- Modification of code should not be required to use a module.
- Separation allows better integration with 3rd party systems.
- Overriding data based on different criteria within Puppet code is cumbersome.
- Embedding configuration data increases complexity of data management.
- Propagating change across your infrastructure is error prone.

Single Source of Truth

See also: *Don't Repeat Yourself*

- Practice of structuring information storage:
 - each data element is stored exactly once
 - can reference this from multiple places
 - updates automatically propagate across the infrastructure
- Benefits of a *single source of truth* architecture:
 - minimizes risk of outdated & incorrect information
 - minimizes work when updating configuration
 - ensures that disparate infrastructure components are configured harmoniously
 - makes identification of data more discoverable
- Puppet uses Hieradata as its *single source of truth* data abstraction layer.

Single Source of Truth

As implemented by Hieradata:

- Simplify management of data in Puppet.
- Puppet classes can request whatever data they need, *when they need it*.
- Hieradata works as a data lookup broker.
- Authoritative source of configuration data.
- Benefits of retrieving configuration data from Hieradata:
 - Easier to configure your own nodes
 - Easier to reuse public Puppet modules
 - Easier to design your modules for reuse
 - Easier to publish your own modules for collaboration
 - Easier to ensure that *all nodes* affected by changes in configuration data are updated

Design and Capabilities

- Hiera uses an ordered hierarchy to look up data:
 - large amount of common data that apply to all nodes
 - override smaller amounts of it wherever necessary
 - as many override levels as needed
- Multiple backend resolution:
 - collect information from multiple data stores
 - single source as far as Puppet is concerned
- Many data sources arranged in hierarchy:
 - static data source: same value for all nodes
 - dynamic data source: value is customized per node via facts
- Backends and data sources are resolved in order until a value is found.

Hierarchical Lookups

- Facts and other variables in scope are used to determine a hierarchy.
- Top down hierarchy for overriding configuration values based on:
 - roles
 - environments
 - locations
 - or anything else
- No coding required!

Pluggable Backends

- Retrieve data from multiple sources
 - integrate with existing CMDB
- Develop custom backends
- Some existing custom backends
 - `hiera-gpg`
 - `hiera-json`
 - `hiera-redis`
 - `hiera-mysql`

Notes:

Learn how to develop your own Hiera backends in the *Extending Puppet Using Ruby* course.

Hiera: sample configuration

- Facts are used to determine the lookup hierarchy.

```
---
:backends:
  - yaml

:yaml:
  :datadir: /etc/puppetlabs/puppet/hieradata

:hierarchy:
  - hosts/{fqdn}
  - env/{environment}
  - common
```

sample /etc/puppet/hieradata/env/dev.yaml

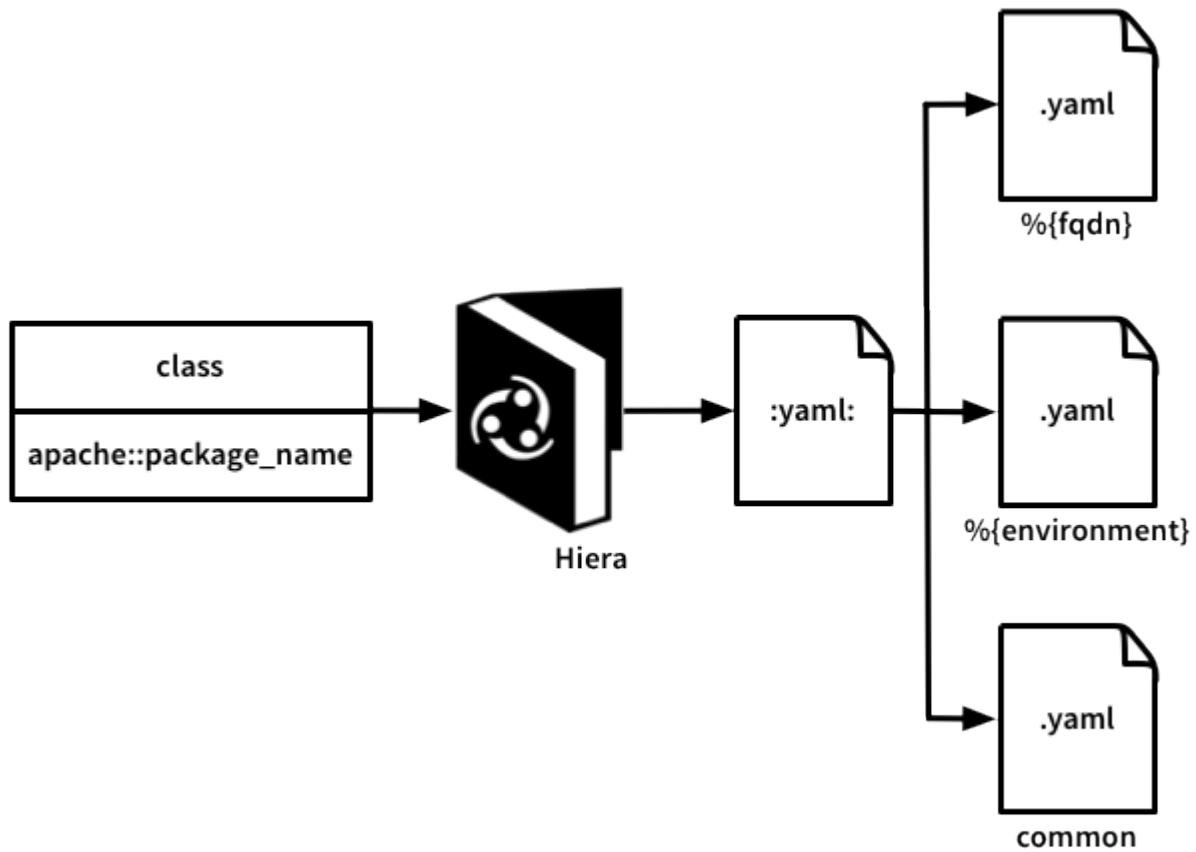
```
---
ntpserver: 212.22.1.3
dnsservers: [ "10.0.0.1", "10.0.0.2" ]
```

Notes:

Hiera will search for key value pairs in the following order:

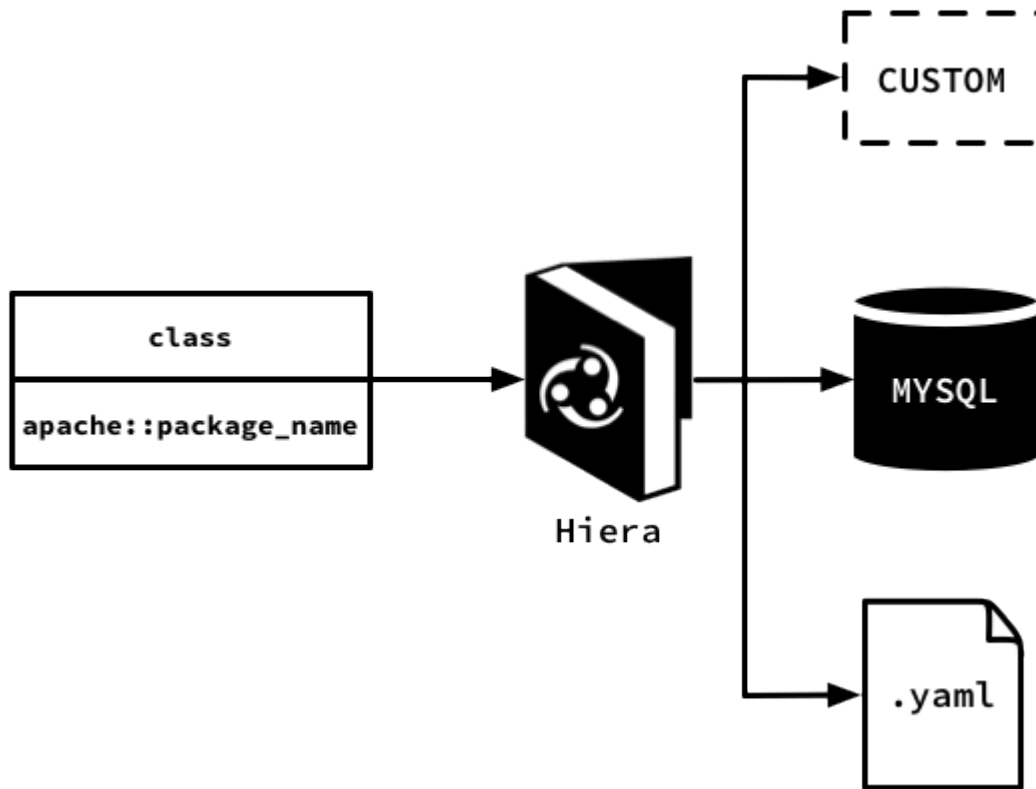
1. /etc/puppet/hieradata/hosts/agent.puppetlabs.vm.yaml
2. /etc/puppet/hieradata/env/dev.yaml
3. /etc/puppet/hieradata/common.yaml

Resolution Hierarchy



1. `${hieradata}/hosts/agent.puppetlabs.vm.yaml`
2. `${hieradata}/env/development.yaml`
3. `${hieradata}/common.yaml`

Hierarchical Database



Life with Hiera

Without Hiera

```
if ( $::environment == 'dev' ) {  
  $ntpserver = '192.168.2.1'  
} else {  
  if ( $::fqdn == 'host4.mycorp.com' ) {  
    $ntpserver = '127.0.0.1'  
  } else {  
    $ntpserver = '213.21.6.4'  
  }  
}
```

With Hiera

```
$ntpserver = hiera('ntpserver')
```

- The appropriate value for this node is looked up automatically.
- No complex logic.
- No out of date configuration settings.
- Automatically propagated.

Hiera

Functions for interacting with Hiera

- `hiera`
 - returns the first value found for a key
- `hiera_array`
 - returns an array of *all matching values* for a key
- `hiera_hash`
 - returns a merged hash of *all matching values* for a key
 - note: all returned values must be hashes
 - useful when returning resources to be instantiated with `create_resources`
- `hiera_include`
 - includes each of an array of class names returned
 - can be used as a lightweight ENC

Using the `hiera` function

Returns the first value found for a key

```
$ cat /etc/puppetlabs/puppet/hieradata/kermit.puppetlabs.vm.yaml
---
mail::smtp: 'smtp.puppetlabs.vm'
users::users_array: [ 'brad', 'ralph' ]
```

```
class users (
  $users_array = hiera('users::users_array')
) {

  user { $users_array
    ensure => present,
  }

}
```

```
include users
# This will automatically look up the users_array using hiera
```

- The `class::param` naming convention future proofs code for Puppet 3.x
- Data returned can be any type Puppet understands

Using the `hiera_array` function

Returns an array of *all matching values* for a key

```
# /etc/puppetlabs/puppet/hiera.yaml
---
[...]
```

`:hierarchy:`

- `%{fqdn}`
- `%{enviroment}`
- `common`

```
$ cat /etc/puppetlabs/puppet/hieradata/common.yaml
---
packages::install_list: [ 'ruby', 'php', 'mysql' ]

$ cat /etc/puppetlabs/puppet/hieradata/development.yaml
---
packages::install_list: [ 'ruby-devel', 'php-devel', 'mysql-devel' ]
```

```
$package_array = hiera_array('packages::install_list')

package { $package_array:
  ensure => installed,
}
```

- In development, packages from both datasources will be installed
- In production, only packages from `common` will be installed

Using the `hier_hash` function

Returns a merged hash of *all matching values* for a key

```
$ cat /etc/puppetlabs/puppet/hieradata/common.yaml
---
users::sysadmins:
  gary:
    shell: /bin/bash
    uid: 501

$ cat /etc/puppetlabs/puppet/hieradata/development.yaml
---
users::sysadmins:
  craig:
    shell: /bin/bash
    uid: 502
```

```
$users = hiera_hash('users::sysadmins')
create_resources('user', $users)
```

- In development, users from both datasources will be managed
- In production, only users from `common` will be managed

Using the `hier_include` function

Includes each of an array of class names returned

```
$ cat /etc/puppetlabs/puppet/hieradata/kermit.puppetlabs.vm.yaml
---
purchased::services: [ 'mysql-server', 'httpd', 'wordpress' ]

$ cat /etc/puppetlabs/puppet/hieradata/oscar.puppetlabs.vm.yaml
---
purchased::services: [ 'pgsql-server', 'nginx', 'drupal' ]
```

```
class purchased {
  # Enables services that customers have purchased
  hier_include('purchased::services')
}
```

```
node 'kermit.puppetlabs.vm' {
  include purchased
}
```

- Each class named in the array returned by `hier_include` will be included in the catalog.

Lab 8.1: Configure Hiera



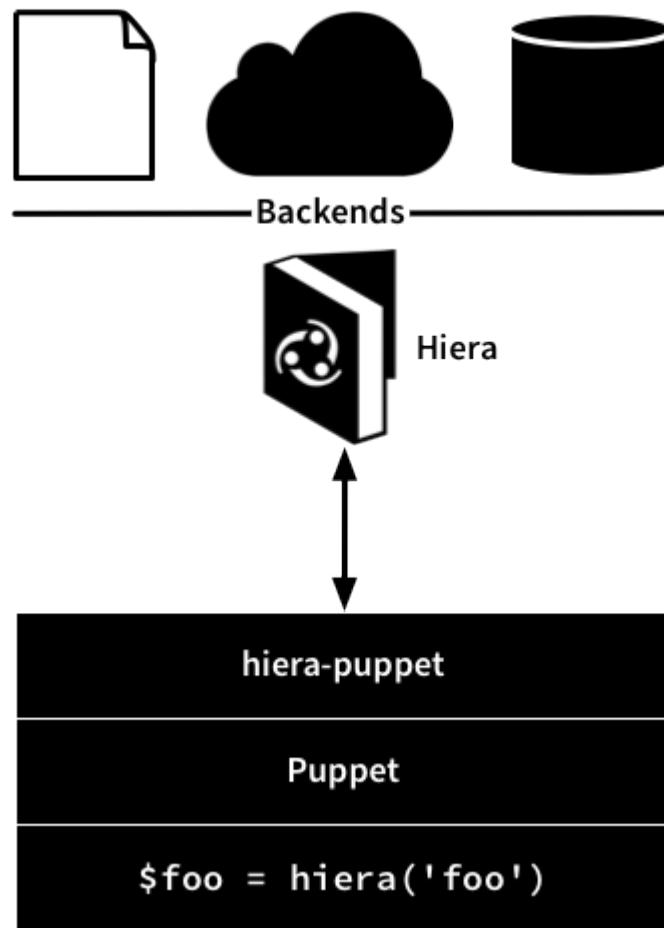
- **Objective:**

- Configure and test Hiera on your Puppet Master.

- **Steps:**

- Create the new Hiera configuration file.
- Create your Hiera datasource hierarchy.
- Use Puppet to test your Hiera data.

Integrating Hiera before Puppet 3.x



Automatic Data Bindings

Automatic parameter lookup in Puppet 3.x

- Class parameters are automatically looked up from Hieradata
- Hieradata keys queried are `class::param`

```
# /etc/puppetlabs/puppet/hieradata/dc1.yaml
---
ntp::time_server: time.puppetlabs.com
```

```
class ntp (
  $time_server,    # automatically uses hiera('ntp::time_server') as default
  $crypto = false, # automatically uses hiera('ntp::crypto', false) as default
) {
  file { ['/etc/ntp.conf']
    content => template('ntp/ntp.conf.erb')
  }
}
```

Simply include the class:

```
include ntp
```

Notes:

The resolution order of class parameters with Automatic Data Bindings is:

1. Passed in parameters
2. Values looked up from Hieradata
3. Defaults expressed in the class signature

Note: This does not replace the hiera functions, but merely augments them.

Getting Ready for Data Bindings

Emulating Data Bindings in Puppet 2.x

- Look up parameter defaults with a manual hiera function call
- Use the same key naming conventions

```
# /etc/puppetlabs/puppet/hieradata/dc1.yaml
---
ntp::time_server: time.puppetlabs.com
```

```
class ntp (
  $time_server = hiera('ntp::time_server'),
  $crypto      = hiera('ntp::crypto', false),
) {
  file { ['/etc/ntp.conf'
    content => template('ntp/ntp.conf.erb')
  ]
}
```

Simply include the class:

```
include ntp
```

Notes:

After upgrading to 3.x the hiera function calls can be removed at your convenience.

extlookup

- Function for looking up data external to Puppet manifests
- Pre-cursor to Hiera (now deprecated in favor of Hiera)
- Has a csv backend (which Hiera does not have)

Data files:

```
# domain_myclient.com.csv:
snmp_contact,John Doe <john@myclient.com>
root_contact,support@%{domain}
client_trusted_ips,192.168.1.130,192.168.10.0/24
```

```
# common.csv:
snmp_contact,My Support <support@my.com>
root_contact,support@my.com
```

Accessing data:

```
$snmp_contact = extlookup('snmp_contact')
```

Notes:

- Documentation: <http://docs.puppetlabs.com/references/latest/function.html#extlookup>
- Project Site: <https://github.com/ripienaar/puppet-extlookup>

Lab 8.2: Centralizing Data



- **Objective:**

- Share data between your modules in multiple environments.

- **Steps:**

- Refactor your `databases::admins::users` class to retrieve the user data from Hiera.
- Move your user data from the hardcoded hash to the `common` datasource in Hiera.
- Add a new user definition in the `development` datasource.

Virtual Resources

Lesson 9: Virtual Resources

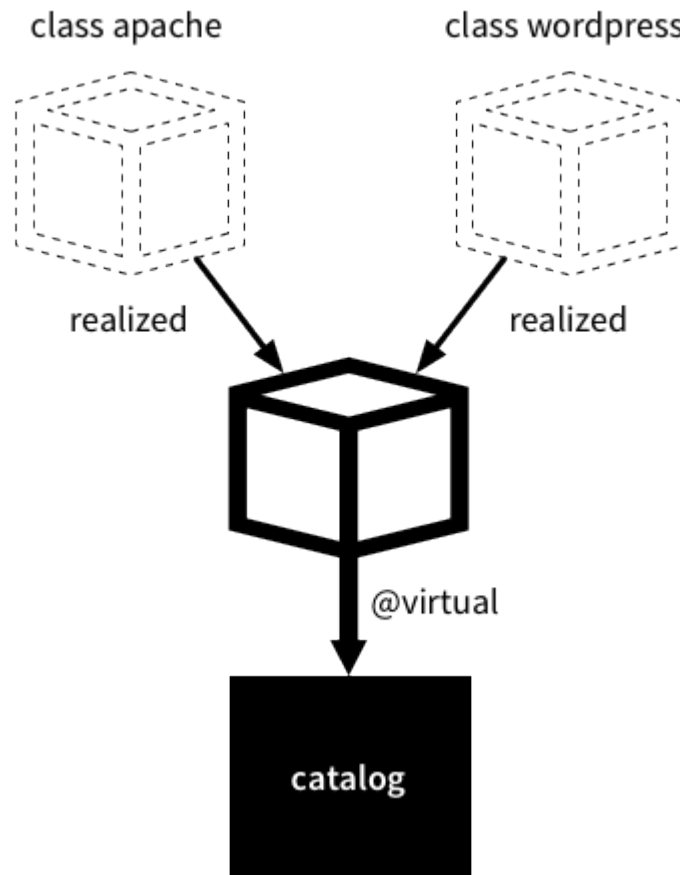
OBJECTIVES

At the end of this lesson, you will be able to:

- Identify the purpose of virtual resources.
- Describe how to declare and realize virtual resources.

Declaring Virtual Resources

Virtual Resources can be declared once but used throughout your configuration.



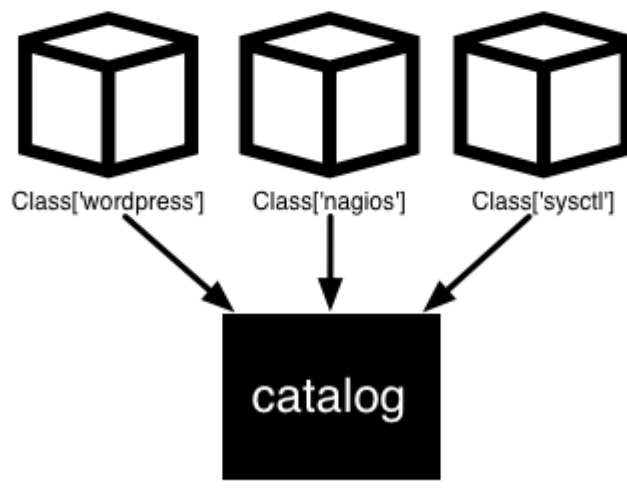
Notes:

Learn more about Virtual Resources in the docs:

- http://docs.puppetlabs.com/puppet/3/reference/lang_virtual.html
- http://docs.puppetlabs.com/guides/virtual_resources.html

Use Case

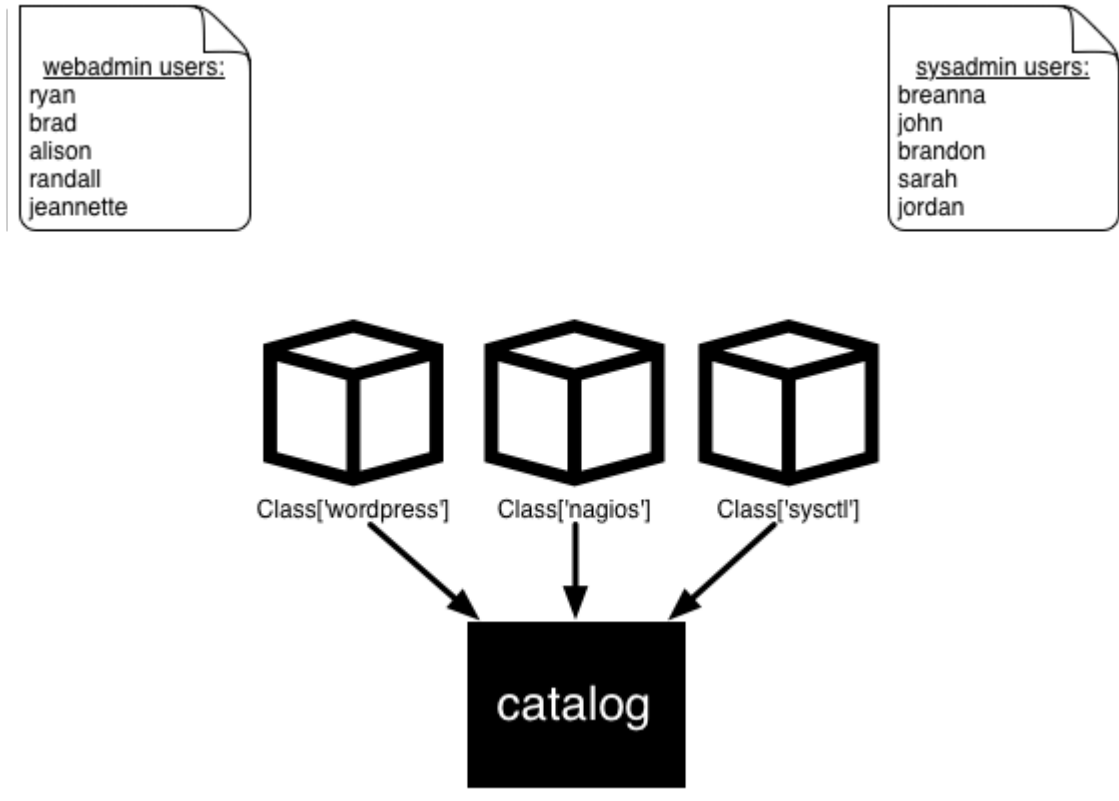
Allows resources to be declared in multiple places without conflict



Imagine a catalog with three classes included.

Use Case

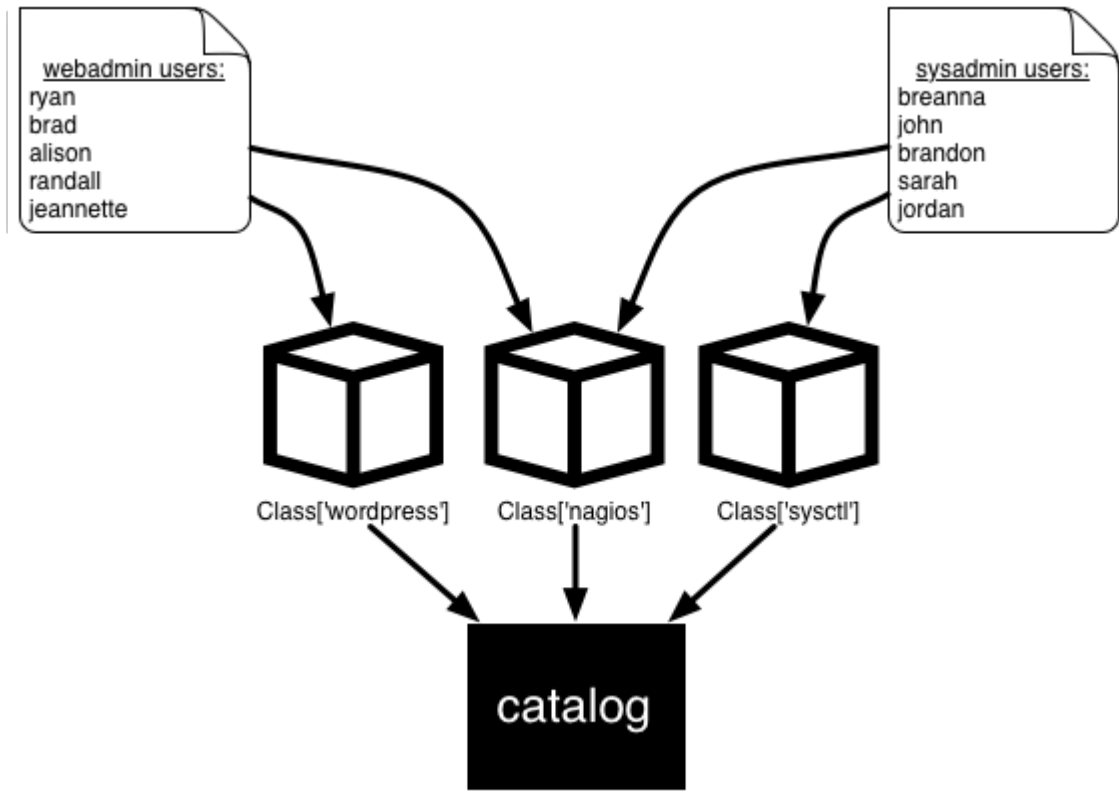
Allows resources to be declared in multiple places without conflict



We also have two lists of users with different responsibilities.

Use Case

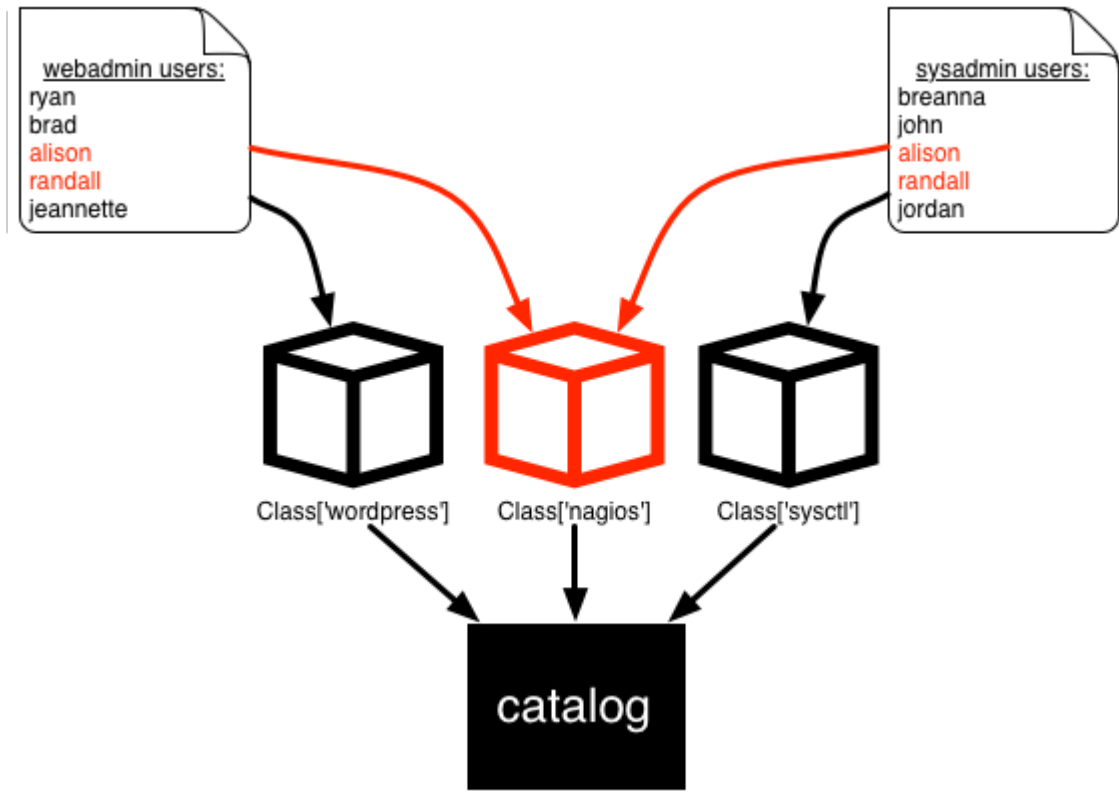
Allows resources to be declared in multiple places without conflict



We want to include the appropriate users in each class.

Use Case

Allows resources to be declared in multiple places without conflict



But what happens when there's a conflict?

Creating a Virtual Resource

Mark resources in the catalog as virtual:

```
@package { 'gcc': ensure => present }
```

```
notice: Finished catalog run in 0.04 seconds
```

Until the virtual resource is realized or collected it will not be added to the catalog.

```
realize Package['gcc']
```

```
notice: /Stage[main]//Package[gcc]/ensure: created  
notice: Finished catalog run in 18.02 seconds
```

Virtual resources can be realized as often as is convenient

```
realize Package['gcc']  
realize Package['gcc'] # no duplicate definitions!  
realize Package['gcc']
```

```
notice: /Stage[main]//Package[gcc]/ensure: created  
notice: Finished catalog run in 18.02 seconds
```

Realizing a Defined Resource Type

Defined resources may also be made virtual.

```
define apache::vhost ( $vhost_name ) {  
  file { [ "/etc/httpd/0_${name}_httpd.conf":  
    content => template('apache/vhost.erb'),  
  ]  
}
```

```
@apache::vhost { 'kermit.puppetlabs.vm': }  
@apache::vhost { 'oscar.puppetlabs.vm': }  
@apache::vhost { 'piggy.puppetlabs.vm': }  
@apache::vhost { 'bigbird.puppetlabs.vm': }  
@apache::vhost { 'animal.puppetlabs.vm': }
```

```
realize Apache::Vhost['kermit.puppetlabs.vm']  
realize Apache::Vhost['piggy.puppetlabs.vm']
```

Exercise 9.1: Realizing a Virtual Resource



- **Objective:**

- Use the `realize` function to add your virtual resource to the catalog.

- **Steps:**

- Create a new `nix_admins` class with virtual resources.
- Create a smoke test for your class.
- Run your smoke test.
- Add the `realize` function to your smoke test.
- Run your smoke test again.

Resource Collectors

```
User <| |> # realize all virtual user resources
User <| group == 'sysadmin' |> # realize all system administrators
User <| tag == 'portland' |> # realize all users tagged with Portland
```

The general form of a resource collector:

```
Type <| [search expression] |>
```

Notably, collectors cannot be used as the value of a resource attribute, the argument of a function, or the operand of an expression.

Realizing Resources Using Collectors

```
@user { 'luke': ensure => present, }

@user { 'james':
  ensure => present,
  group  => 'dba',
}
@user { 'jeff':
  ensure => present,
  group  => 'sysadmin',
}
@user { 'brad':
  ensure => present,
  group  => 'webadmins',
}
```

```
User <| (group == dba or group == sysadmin) or title == luke |>
```

The users `luke`, `james`, `jeff` will be created, but `brad` will not.

Resource Collectors Conditionals

- `==` - (equality search)
 - When applied to an array, match any members
- `!=` - (non-equality search)
 - When applied to an array, match any members
- `and` - Both operands must be valid search expressions.
- `or` - Both operands must be valid search expressions.

Resource Collectors Conditional Example

```
@user { 'ralph': ensure => present, tag => ['webadmin','winadmin'] }
@user { 'bradh': ensure => present, tag => ['webadmin','sysadmin'] }
@user { 'zacks': ensure => present, tag => ['webadmin','sysadmin'] }
@user { 'james': ensure => present, tag => ['webadmin','netadmin'] }
@user { 'ryanc': ensure => present, tag => ['webadmin','winadmin'] }
@user { 'lukek': ensure => present, tag => ['webadmin','netadmin'] }

User <| tag == 'webadmin' and tag != 'sysadmin' |>
```

Which users will be collected ?

Resource Collectors Conditional Example

```
@user { 'ralph': ensure => present, tag => ['webadmin','winadmin'] }
@user { 'bradh': ensure => present, tag => ['webadmin','sysadmin'] }
@user { 'zacks': ensure => present, tag => ['webadmin','sysadmin'] }
@user { 'james': ensure => present, tag => ['webadmin','netadmin'] }
@user { 'ryanc': ensure => present, tag => ['webadmin','winadmin'] }
@user { 'lukek': ensure => present, tag => ['webadmin','netadmin'] }

User <| tag == 'webadmin' and tag != 'sysadmin' |>
```

Which users will be collected ?

All the users will be collected, since all of them have a `tag` that is not `sysadmin`!

Resource Collectors

Use collections to set dependencies on many resources at once

```
yumrepo { 'internal_repo':  
  baseurl => 'http://localrepo.example.com/x86_64',  
  descr   => 'All internally built packages',  
  enabled => 1,  
}  
  
yumrepo { 'custom_packages':  
  baseurl => 'http://localrepo.example.com/custom/x86_64',  
  descr   => 'All custom built packages',  
  enabled => 1,  
}  
  
# All packages require the internal repo to be configured  
Package<| |> {  
  require => Yumrepo['internal_repo'],  
}  
  
# And those tagged with custom also require the custom repo  
Yumrepo['custom_packages'] -> Package <| tag == 'custom' |>
```

All package resources (not just virtual ones) are selected.

Notes:

Remember that a collector will realize all virtual resources that it matches.

Lab 9.2: Resource Collectors



- **Objective:**

- Use a resource collector to realize your resources.

- **Steps:**

- Edit your `nix_admins` class.
- Create a `wordpress` collector for the 'wordpress' tag.
- Apply your puppet module configuration.

Exported Resources

Lesson 10: Exported Resources

OBJECTIVES

At the end of this lesson, you will be able to:

- Configure store configs for exported resources.
- Configure PuppetDB as a replacement backend for storeconfigs.

Sharing Information

Sometimes information from one node is needed to configure another node.

- Example use cases
 - Distribute host records when you can't use DNS
 - Distribute SSH host keys among machines in a lab and only allow login from known hosts
 - Inform an application about each database available in a cluster

Exporting Resources

Puppet has the ability to export resources to a database so that they can be collected and used on other hosts.

```
class hosts {
  # create a virtual host resource based on known information
  # and export it back to the Puppet Master
  @@host { $::hostname:
    ip          => $::ipaddress,
    host_aliases => $::fqdn,
  }

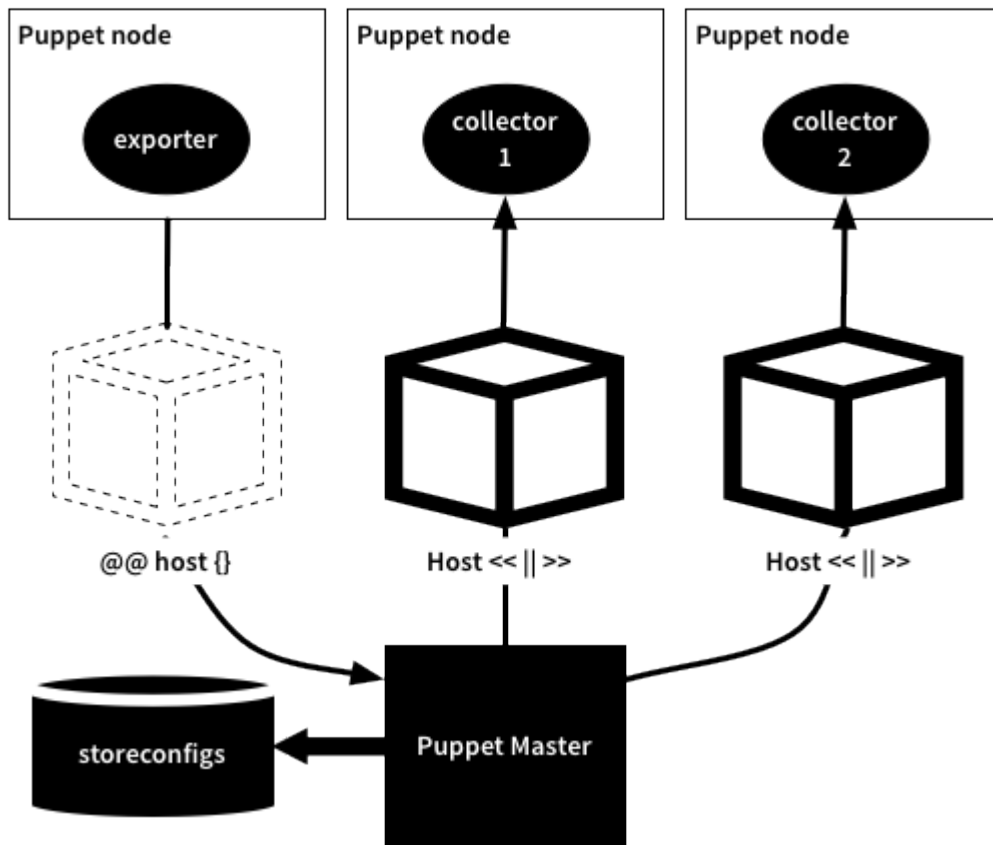
  # collect all exported resources and realize them on this host
  Host <<||>>

  # ensure that we have no host entries that aren't explicitly configured
  resources { 'host':
    purge => true,
  }
}
```

- Exported resources build on the concept of virtual resources
- Double @@ sign marks the virtual resource as exportable
- Double << and >> brackets realize exported virtual resources

Exporting Resources

This allows nodes to collectively share information.



Configuration of storeconfigs is required.

Enabling storeconfigs

```
# /etc/puppetlabs/puppet/puppet.conf
[master]
  storeconfigs = true
```

Enable `thin_storeconfigs` to only store exported resources.

```
# /etc/puppetlabs/puppet/puppet.conf
[master]
  storeconfigs = true
  thin_storeconfigs = true
```

This setting is ignored when using puppetdb.

Lab 10.1: Export a Resource



- **Objective:**

- Export a host record for your VM to the rest of the class.

- **Steps:**

- Refactor your hosts class
- Export a host record for yourself
- Import host records from the other students.
- Create a node declaration that includes this class in your `site.pp` file.

Storeconfig vs PuppetDB

Active record backed store configs:

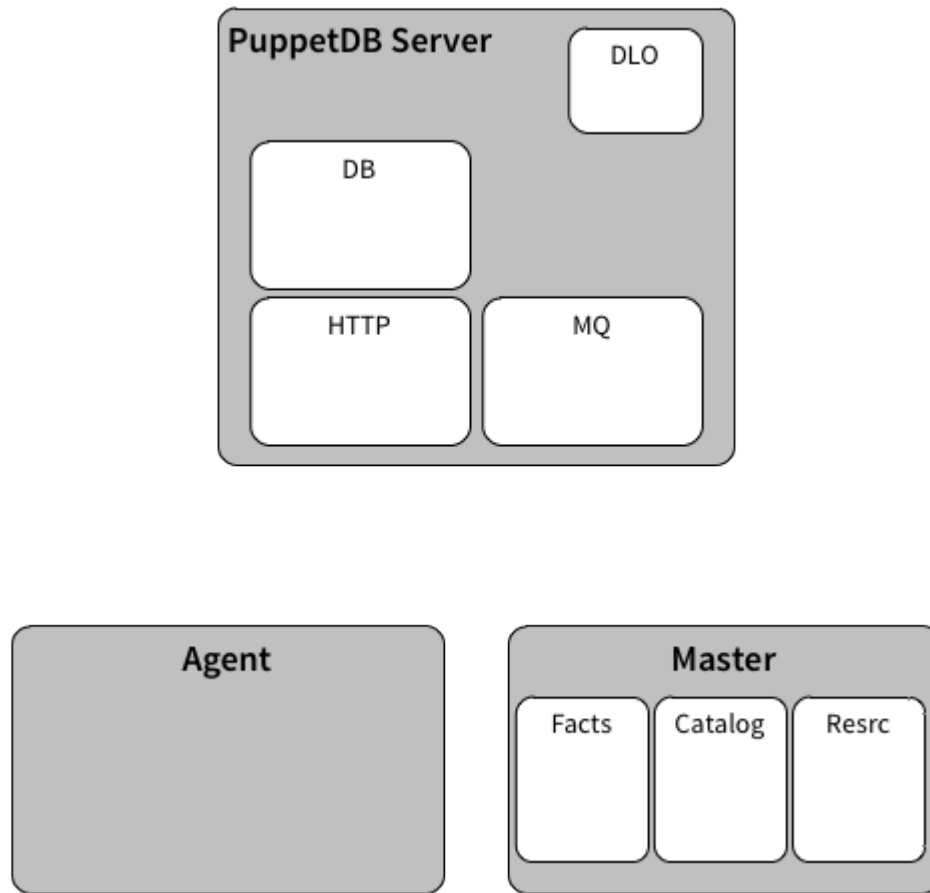
- Default implementation does not scale well.
- Storeconfig makes one transaction per resource.

PuppetDB backed store configs:

- Processes one transaction per catalog.
- Processes the catalog in the background.

PuppetDB

Puppet Data Warehouse



Notes:

- Stores all of its data asynchronously
- Frees up the master to compile more catalogs
- Documented REST API for Resource Fact retrieval
- Drop-in replacement for stock storeconfigs functionality
- Compilation times are much reduced compared to traditional storeconfigs

PuppetDB System Requirements

- nix Server with JDK 1.6+
- Red Hat Enterprise Linux 5 or 6 or any distro derived from it (including CentOS)
- Debian Squeeze, Lenny, Wheezy, or Sid
- Ubuntu 12.04 LTS, 10.04 LTS, 8.04 LTS, 11.10, or 11.04
- Fedora 15 or 16
- Puppet 2.7.12+ or later

Installing PuppetDB

```
[root@training ~]# puppet module install puppetlabs/puppetdb
```

```
node puppetmaster {  
  include puppetdb  
  include puppetdb::master::config  
}
```

- Install and manage the PuppetDB server and database.
- Configure the Puppet Master to use PuppetDB.

PuppetDB

`$confdir/puppetdb.conf`

```
# /etc/puppetlabs/puppet/puppetdb.conf
[main]
  server = puppetdb.example.com
  port = 8081
```

The `routes.yaml` file is managed by the `puppetlabs/puppetdb` module.

```
# /etc/puppetlabs/puppet/routes.yaml
---
master:
  facts:
    terminus: puppetdb
    cache: yaml
```

Metrics

PuppetDB has a built-in dashboard.

<http://{puppetdbhost}:8080/dashboard/index.html>

- By default, this listens only on `localhost`
- Pass a parameter to the `puppetdb` class to enable external access:

```
class { 'puppetdb':  
  listen_address => 'example.foo.com'  
}
```

PuppetDB DLO

Dead Letter Office

When commands fail irrecoverably or over a long period of time, they are written to disk in what is called the dead-letter office (or DLO).

- Command is retried up to 16 times
- Byte for byte copy of the original request
- Directory may grow in size if many requests fail

Puppet Node Clean

Clean out the stored configuration database for a given node.

```
[root@training ~]# puppet node clean kermit.puppetlabs.vm --unexport
```

Want to know more? `puppet man node`

- Currently not supported by PuppetDB

Notes:

See bug report: <https://projects.puppetlabs.com/issues/14608>

Puppet Node Deactivate

Exclude a node from storeconfigs in PuppetDB

```
[root@training ~]# puppet node deactivate firebox.puppetlabs.vm --mode master
```

- Retains the current storeconfig resources for that node.
- Does not remove exported resources from managed nodes.
 - only stops managing those resources
 - use `resources resources` to purge unwanted resources

Want to know more? `puppet man node`

node-ttl-days

Configured in puppetdb.conf

- `gc-interval` This controls how often, in minutes, to compact the database. The compaction process reclaims space and deletes unnecessary rows. If not supplied, the default is every 60 minutes.
- `node-ttl` Time with no activity before a node will be auto-deactivated. Nodes will be checked for staleness every `gc-interval` minutes.
- `node-purge-ttl` The length of time a node can be deactivated before it's deleted from the database.

Some settings may also be managed via the `puppetdb` wrapper class

```
class { 'puppetdb':  
  listen_address => 'example.foo.com',  
  node_ttl       => '14d',  
  node_purge_ttl => '7d',  
}
```



Lab 10.2: Export a Resource

- **Objective:**

- Export a resource to load balance our Wordpress sites behind the classroom proxy.

- **Steps:**

- Install the `puppetlabs/haproxy` module.
- Export a `haproxy::balancermember` resource.
- Run the Puppet Agent.
- Create a blog post with something interesting to share.
- Once the proxy has collected resources, browse to <http://proxy.puppetlabs.vm>

Scaling Puppet

Lesson 11: Scaling Puppet

OBJECTIVES

At the end of this lesson, you will be able to:

- Configure a Puppet Master with a Shared Certificate Authority.
- Add puppet dynamically to a load balancer.
- Review best practices for large cluster systems.

Recommended Baseline

Puppet Master running under Passenger

- Roughly 1,000 nodes
- Check-in time set to 30 mins with a 5 min splay (default)
- Hardware requirements:
 - 8 core processor
 - 8 GBs of Ram

Thundering Herd

When many nodes check at once

- Caused by unevenly distributed node checkin times
- Large groups of nodes using a random splay may converge on certain times
 - side effect of using a pseudo random number
- Sometimes alleviated by running Agent as a cron job
 - disable the Puppet Agent daemon
 - use `fqdn_rand()` to generate a predictable random checkin time

One Shared CA for Multiple Puppet Masters

One Puppet Master is designated as the `ca_server`.

- Certificate signing requests are sent to this server.
- Allows agents to request a catalog from any master with a valid certificate
- One CA signs all certificates so all agents and all masters trust each other
- Secondary masters should have their CA functionality disabled

```
[master]  
ca = false
```

- Each node can be configured to point to one CA

```
[main]  
ca_server = classroom.puppetlabs.vm
```

- Alternatively, the load balancer can proxy all CSR requests to the CA Server

Certificate Revocation List Replication

CRL lists nodes no longer authorized

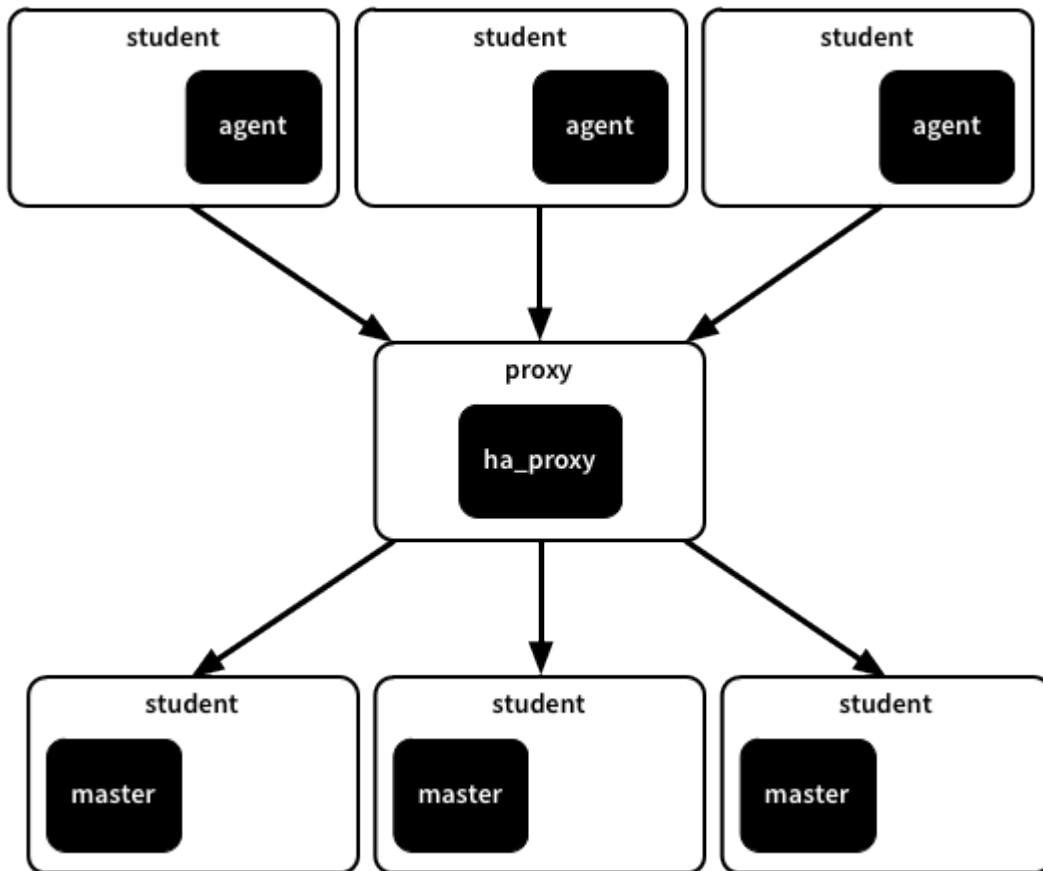
- Puppet does not automatically distribute this file when changes occur
- You should distribute it yourself
- Apache must be restarted to reload the CRL

```
class sharedca::secondary {  
  file { ['/etc/puppetlabs/puppet/ca/ca_crl.pem':  
    ensure => file,  
    source => 'puppet:///modules/sharedca/crl.pem',  
    notify => Service['pe-httpd'],  
  }  
}
```

Notes:

`$ssldir/crl.pem` is the list that the Puppet Agent uses. It is generated automatically from the CA Server's `$ssldir/ca/ca_crl.pem`. A node that is both a master and an agent will have both files.

Classroom Infrastructure





Lab 11.1: Configure a Load Balance Member

- **Objective:**

- Configure your Puppet Master to work with the classroom proxy.

- **Steps:**

- Install the `puppetlabs/haproxy` module.
- Create a new `proxy` module.
- Export an `haproxy::balancermember` resource for your node.
- Regenerate & sign certificates with a `dns_alt_name`

Splitting off the Fileserver Role

Using the Puppet File Server URI

- Recursive file serving can be a heavy load.
- Useful to separate roles and centralize fileserving
 - one master can compile catalogs and another can serve files
- The Puppet URI format has an implicit hostname built in
 - defaults to master serving the current catalog
 - almost never used
 - if you set the hostname, the file is requested from the specified host

```
source => "puppet://${hostname}/${mountpoint}/${path}"
```

```
$local_server = hiera('example::local_server')
file { ['/var/db/schema.sql']:
  ensure => file,
  source => 'puppet://$local_server/modules/mysql/schema.sql',
}
```

Lab 11.2: Distributing File Servers



- **Objective:**

- Create a new Puppet File Server mount point.

- **Steps:**

- Edit your `fileserver.conf`.
- Add a new mount point `[local]`.
- Share a sample file via the proxy to other students.

Three Common Architectures

Choice of layout depends greatly on catalog complexity

- Small scale
 - Up to 500-800 nodes
- Medium scale
 - Up to 1500-2000 nodes
- Large scale
 - Up to 6000 nodes
- Extra large scale
 - More than 6000 nodes

Small Scale Architecture

Choice of layout depends greatly on catalog complexity

- Small scale
 - Single server
 - Puppet Enterprise default install
 - Up to 500-800 nodes

Small Scale Architecture

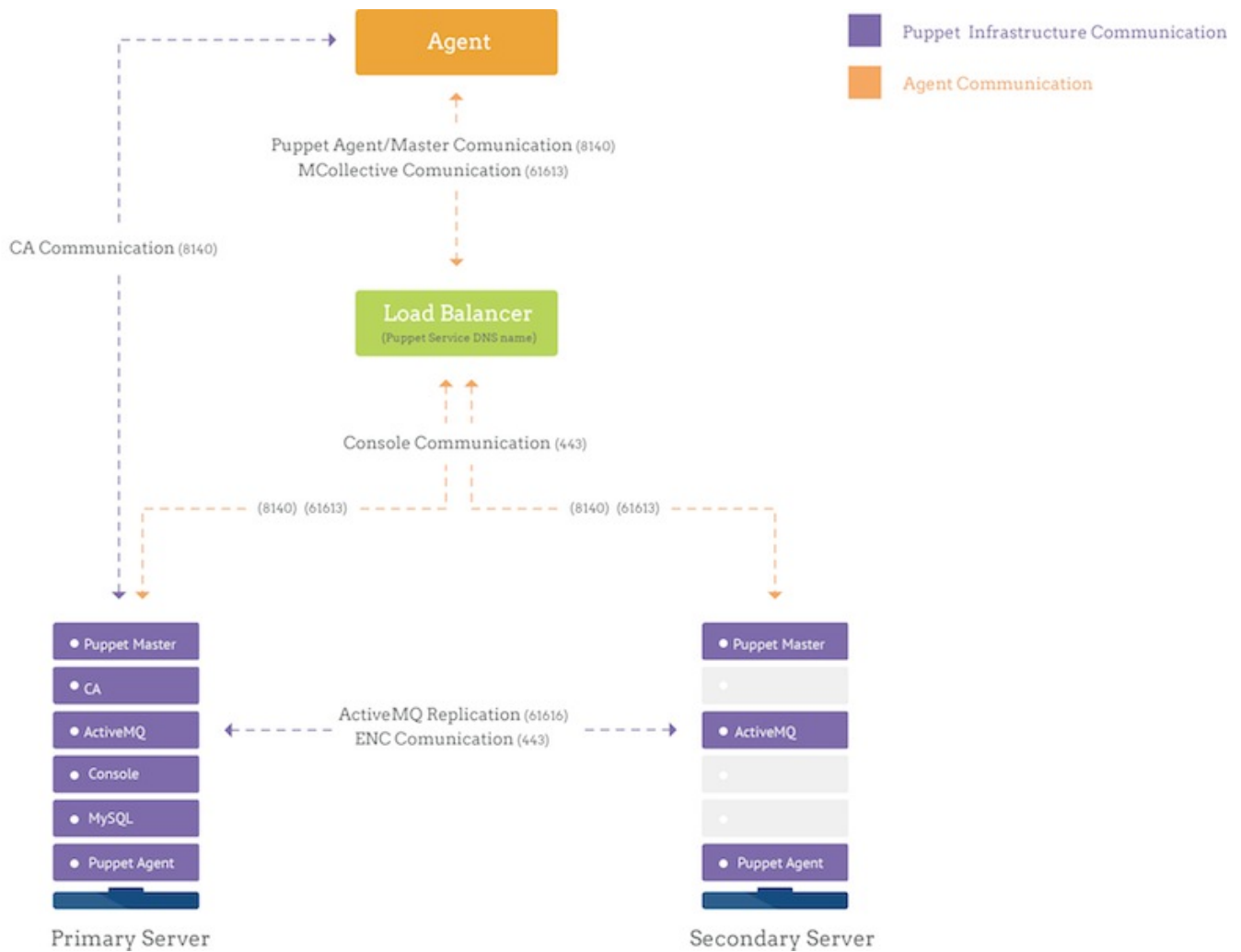


Medium Scale Architecture

Choice of layout depends greatly on catalog complexity

- Medium scale
 - Two or more masters behind a load balancer
 - Similar to the classroom layout
 - Up to 1500-2000 nodes

Medium Scale Architecture

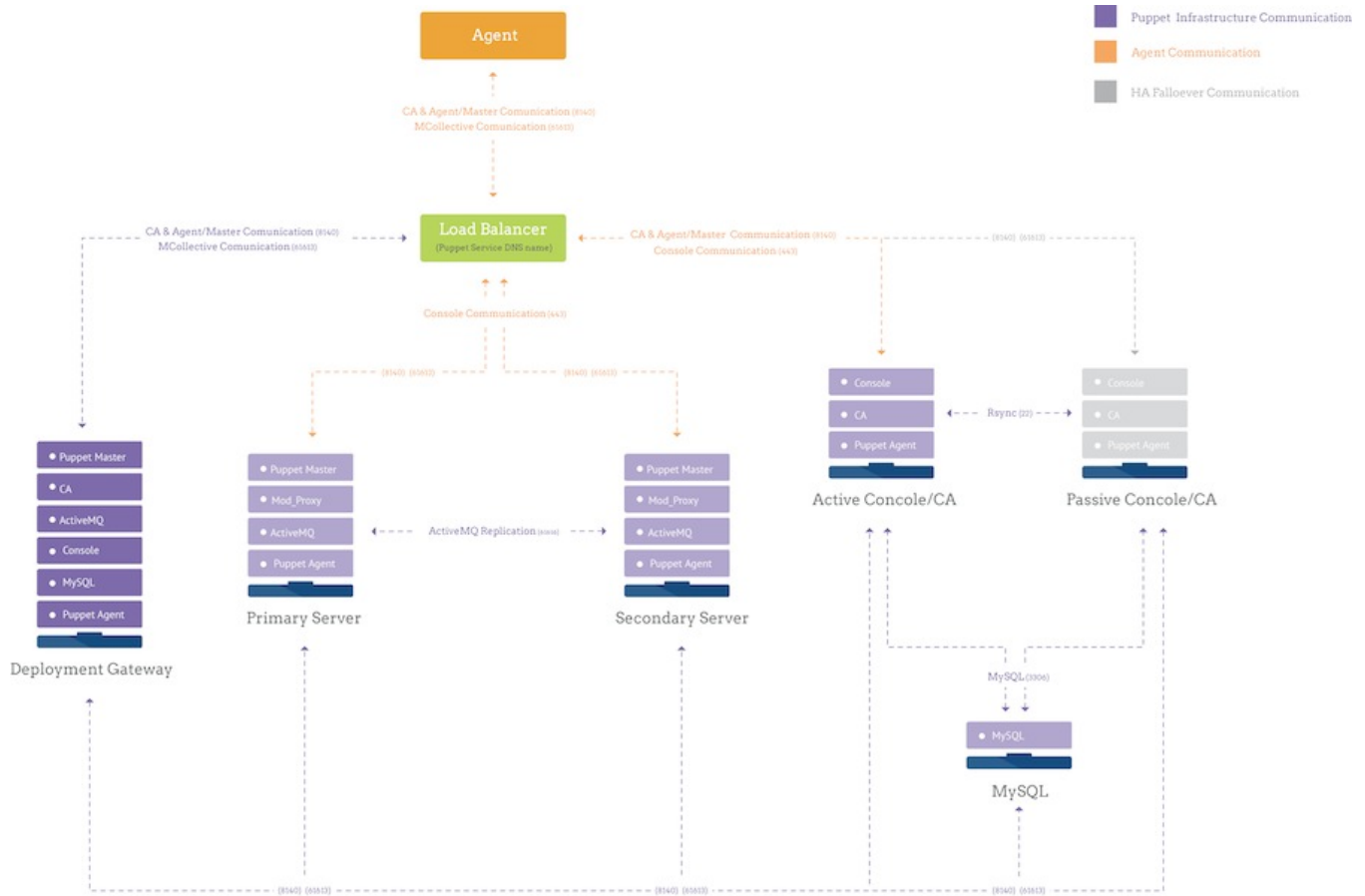


Large Scale Architecture

Choice of layout depends greatly on catalog complexity

- Large scale
 - Multiple levels of load distribution & redundancy
 - Up to 6000 nodes

Large Scale Architecture



Extra Large Scale Architecture

Choice of layout depends greatly on catalog complexity

- Extra large scale
 - Greater than 6000 nodes
 - No standard layout
 - Expand and test as required

Estimating Requirements

Based on catalog compile time

$$m = \frac{ns}{cr}$$

```
m: number of masters  
n: number of nodes  
s: catalog compile time in seconds  
c: cores per master  
r: run interval in seconds
```

- assumes even temporal load
- no replacement for actual benchmarking

Notes:

This is a rough estimate only and should be taken as a starting point. Round up to the next whole number and benchmark some puppet runs. Make sure to account for variability and spikes in load.

Exercise 11.3: Estimating Capacity



- **Objective:**

- Estimate how many Puppet Masters you will need to support your infrastructure.

- **Steps:**

- Use the number of nodes running in your production infrastructure.
- Inspect syslog on your production master (if available) to estimate the average catalog compile time.
- Adjust run interval times and number of cores and observe the effects on the number of masters needed.

Advanced Reporting

Lesson 12: Advanced Reporting

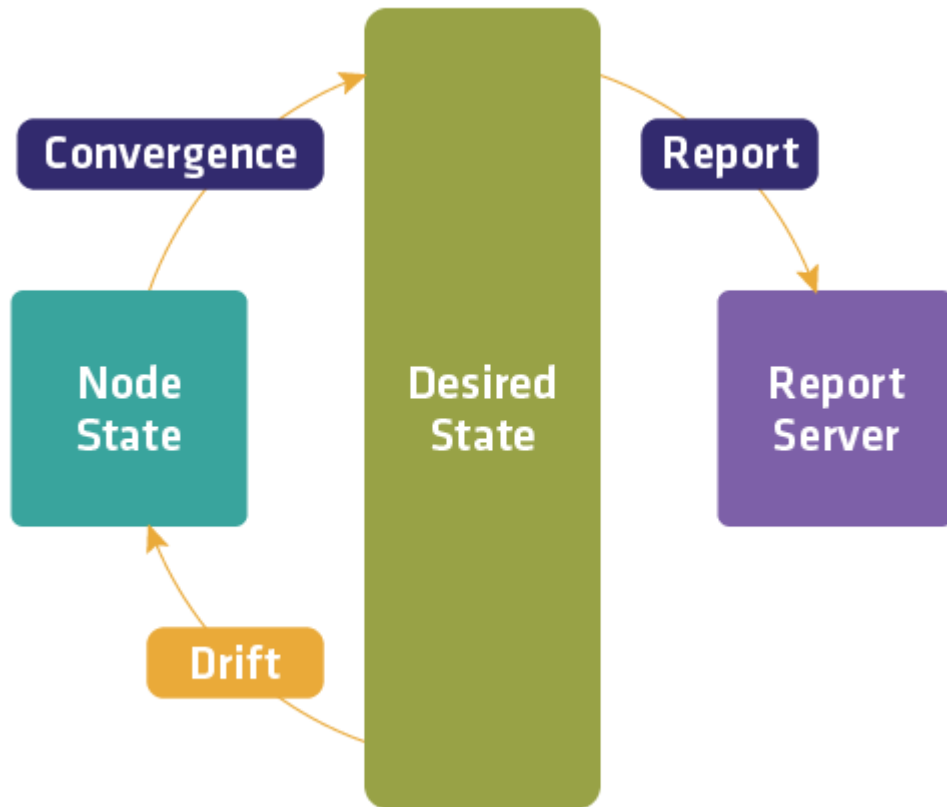
OBJECTIVES

At the end of this lesson, you will be able to:

- Identify required components of a report handler.
- Install a new report handler.
- Enable and Disable report handlers.

Agent Reports

The Puppet Agent can send a report to the Puppet Master after every puppet run.



What's in a Report?

Basic Example

```
info: Applying configuration version '1328975856'  
notice: Hello World!  
notice: /Notify[example]/message: defined 'message' as 'Hello World!'  
notice: Finished catalog run in 0.03 seconds
```

Transaction Data

```
notice: /Notify[example]/message: defined 'message' as 'Hello World!'
```

Metric Data

```
notice: Finished catalog run in 0.03 seconds
```


Configuring Report Handlers

- On the Puppet Agent

```
#  
# /etc/puppetlabs/puppet/puppet.conf  
#  
[agent]  
  report = true
```

- On the Puppet Master

```
#  
# /etc/puppetlabs/puppet/puppet.conf  
#  
[master]  
  reports = https,tagmail,store,log
```

- Puppet Enterprise defaults:
 - reporting is enabled
 - the `https` report handler is enabled and POSTs reports to the Console

Log

- Contains every log message in a transaction.
- Can be used to centralize client logs into syslog.

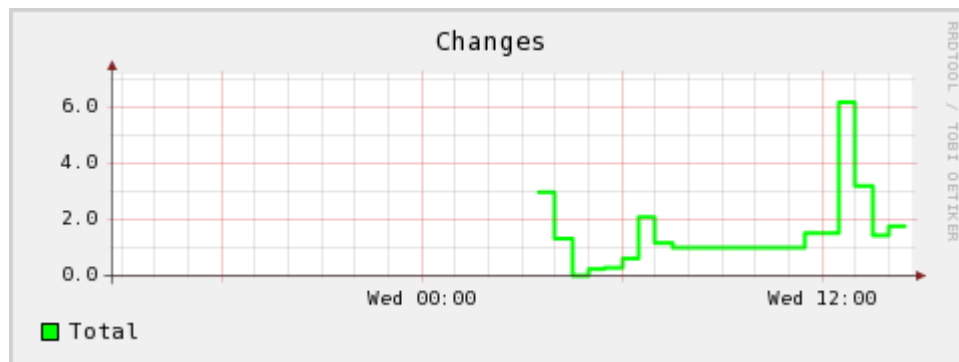
Example (text trimmed for slide):

```
#/var/log/messages  
Compiled catalog for training.puppetlabs.vm in 0.86 seconds  
Caching catalog for training.puppetlabs.vm  
Applying configuration version '1328977795'  
Hello World!  
(/Notify[example]/message) defined 'message' as 'Hello World!'  
Finished catalog run in 0.69 seconds
```

Uses the system syslog calls.

RRD Graph

- Generates RRD graphs from transaction report data.
- Requires Ruby's rrdtool library and the rrd binary.
 - Install the system packages `ruby-rrd` or `rrdtool-ruby`
 - Install the `RubyRRDtool` gem for Ruby bindings



Configuration Options

- `rrddir` - specify where RRD files and graphs get created.
- `rrdinterval` - Configure the rrd interval defaults to `runinterval`.

Store

Stores report data on the Puppet Master as YAML.

Example:

```
- !ruby/object:Puppet::Util::Log
  file: &id007 /etc/puppetlabs/puppet/modules/helloworld/manifests/init.pp
  level: !ruby/sym notice
  line: 4
  message: &id006 defined 'message' as 'Hello World!'
  source: "/Stage[main]/Helloworld/Notify[example]/message"
  tags:
    - notice
    - notify
    - example
    - class
    - helloworld
    - node
    - default
  time: 2012-02-11 16:45:11.057322 +00:00
```

Exercise 12.1: Configure Reporting



- **Objective:**

- Configure `log` and `store` report handlers on your Puppetmaster and clients.

- **Steps:**

- Enable the `log` and `store` report handlers.
- Verify that the logs are being sent to the Puppetmaster and that transaction reports are being stored.
- Examine syslog messages and the stored reports.
- Write a quick Ruby script using the sample code provided to process stored reports.

Tagmail

Delivers log reports via email based on tags.

```
#  
# /etc/puppetlabs/puppet/tagmail.conf  
#  
  
all: admins@puppetlabs.com  
webserver: webadmin@puppetlabs.com  
  
From: report@training.puppetlabs.vm  
Subject: Puppet Report for training.puppetlabs.vm  
To: admins@puppetlabs.com  
  
/Notify[9]/message (notice): defined 'message' as 'Hello World!'
```

Set the `reportfrom` email address and either the `smtpserver` or `sendmail` setting on the Puppet Master.

Tagmail Supported Tags

A comma-separated list of tags and !negated tags:

- all
- Explicit tags
- Class names

Tagmail supports puppet log levels:

- debug, info, notice, warning, err, alert, emerg, crit, verbose

Config Version in Reports

- `config_version` can be customized.
- Correlates logs in reports to code that resulted in them.

```
# /etc/puppetlabs/puppet/puppet.conf
[main]
  config_version = /bin/date
```

```
info: Caching catalog for agent.puppetlabs.vm
info: Applying configuration version 'Tue Oct 9 11:35:20 PDT 2012'
notice: Finished catalog run in 2.73 seconds
```

- Can be integrated with version control to display latest commit.

```
# /etc/puppetlabs/puppet/puppet.conf
[main]
  config_version = /usr/local/bin/return_git_version
```

```
info: Caching catalog for agent.puppetlabs.vm
info: Applying configuration version '5d4bdb7'
notice: Finished catalog run in 2.95 seconds
```


Lab 12.2: Create a Custom `config_version`



- **Objective:**

- Modify puppet to use the date command for the configuration version.

- **Steps:**

- Edit `puppet.conf` and add a `config_version` setting.
- Trigger a puppet run.

Using the Audit Metaparameter

When you want visibility instead of control

```
file { ['/etc/hosts':  
  audit => [ owner, group, mode ],  
}
```

- Normally Puppet resources describe desired state
- `audit` metaparameter instructs Puppet to monitor instead
- Provide a list of attributes to monitor
 - `all` instructs Puppet to monitor all attributes
- Puppet will inform you when changes occur
- Can monitor any resource

Using the Audit Metaparameter

```
file { ['/etc/motd']:
  ensure => present,
  audit  => all,
}
```

```
[root@training ~]# puppet apply audit.pp
notice: /Stage[main]//File[/etc/motd]/ensure: audit change: newly-recorded va ...
notice: /Stage[main]//File[/etc/motd]/content: audit change: newly-recorded v ...
notice: /Stage[main]//File[/etc/motd]/owner: audit change: newly-recorded val ...
notice: /Stage[main]//File[/etc/motd]/group: audit change: newly-recorded val ...
notice: /Stage[main]//File[/etc/motd]/mode: audit change: newly-recorded valu ...
[...]
```

```
[root@training ~]# puppet apply audit.pp
notice: Finished catalog run in 0.02 seconds
```

```
[root@training ~]# puppet apply audit.pp
notice: Finished catalog run in 0.02 seconds
```

```
[root@training ~]# echo "** Externally Modified **" >> /etc/motd
```

```
[root@training ~]# puppet apply audit.pp
notice: /Stage[main]//File[/etc/motd]/content: audit change: previously recor ...
notice: /Stage[main]//File[/etc/motd]/ctime: audit change: previously recorde ...
notice: /Stage[main]//File[/etc/motd]/mtime: audit change: previously recorde ...
notice: Finished catalog run in 0.02 seconds
```

```
[root@training ~]# cat /etc/motd
This is managed by Puppet
** Externally Modified **
```

Notes:

Notice that this resource instructs Puppet to not enforce any attributes of the file other than its presence. It will merely record attributes and report when they change.

Lab 12.3: Using Report Handlers



- **Objective:**

- Configure your server to send reports to an irc chat server.

- **Steps:**

- Create a (second) ssh connection to your virtual machine.
- Open a connection to the classroom internet relay chat channel.
- Install irc report processor.
- Create a deliberate failure and confirm irc notification.

Troubleshooting

Lesson 13: Troubleshooting

OBJECTIVES

At the end of this lesson, you will be able to:

- Troubleshoot common certificate errors.
- Identify files & directories that contain log data.
- Identify configuration files used with puppet.

auth.conf

Control access to Puppet's REST API endpoints

```
# /etc/puppetlabs/puppet/auth.conf
path /facts
method find, search
auth yes
allow custominventory.site.net, devworkstation.site.net

# A more complicated rule
path ~ ^/file_(metadata|content)/user_files/
auth yes
allow /^(.+\.)?example.com$/
allow_ip 192.168.100.0/24

# An exception allowing one authenticated workstation to access any endpoint
path /
auth any
allow devworkstation.site.net
```

Learn more about auth.conf ACLs at

http://docs.puppetlabs.com/guides/rest_auth_conf.html

autosign.conf

Control which CSRs will be signed automatically

```
# /etc/puppetlabs/puppet/autosign.conf
rebuilt.example.com
*.scratch.example.com
*.local
```

- Certificate requests to sign automatically.
- List of certnames or certname globs (one per line).

fileserver.conf

Configure fileserver mountpoints

```
# /etc/puppetlabs/puppet/fileserver.conf
[mount_point]
  path /path/to/files
  allow *.example.com
  deny *.wireless.example.com
```

- Automagic mountpoint at `/modules` that maps to the `modulepath(s)`.
- Add arbitrary mountpoints to serve files via Puppet URI.

Certificate - Subject Name

```
[root@training ~]# puppet cert print classroom.puppetlabs.vm
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 2 (0x2)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: CN=Puppet CA generated on classroom.puppetlabs.vm at Wed May 01...
    Validity
      Not Before: Apr 30 19:30:01 2013 GMT
      Not After : Apr 30 19:30:01 2018 GMT
    Subject: CN=classroom.puppetlabs.vm
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
  [...]
    X509v3 Subject Alternative Name:
      DNS:classroom, DNS:classroom.puppetlabs.vm, DNS:puppet, DNS:pup...
  [...]
```

```
warning: Not using cache on failed catalog
err: Could not retrieve catalog; skipping run
err: Could not send report: Server hostname 'incorrect.puppetlabs.vm' did not
      match server certificate; expected one of classroom.puppetlabs.vm,
      DNS:classroom, DNS:classroom.puppetlabs.vm, DNS:proxy.puppetlabs.vm
```

Certificate Time

- Certificate creation is based on Master's clock.
- Default validity period is 5 years from creation.
- Puppet 3.X and higher shows certificate expiration warnings.

```
[root@training ~]# cd /etc/puppetlabs/puppet/ssl/certs
[root@training certs]# openssl x509 -in classroom.puppetlabs.vm.pem -noout -dates
notBefore=Oct 15 15:19:53 2012 GMT
notAfter=Oct 15 15:19:53 2017 GMT
```

```
info: Caching certificate for tardis.company.com
err: Could not retrieve catalog from remote server: SSL_connect returned=1
      errno=0 state=SSLv3 read server certificate B: certificate verify failed.
      This is often because the time is out of sync on the server or client
warning: Not using cache on failed catalog
err: Could not retrieve catalog; skipping run
err: Could not send report: SSL_connect returned=1 errno=0 state=SSLv3 read
      server certificate B: certificate verify failed. This is often because
      the time is out of sync on the server or client
```

puppet cert revoke

```
[root@training ~]# puppet cert revoke bill.puppetlabs.vm
notice: Revoked certificate with serial 11
```

```
[root@training ~]# puppet cert revoke zack.puppetlabs.vm
notice: Revoked certificate with serial 10
```

```
[root@training ~]# cd $(puppet agent --configprint ssldir)
[root@training ~]# openssl crl -in ca/ca_crl.pem -noout -text
Serial Number: 0A
  Revocation Date: Dec  4 19:51:07 2012 GMT
  CRL entry extensions:
    X509v3 CRL Reason Code:
      Key Compromise
Serial Number: 0B
  Revocation Date: Dec  4 19:48:12 2012 GMT
  CRL entry extensions:
    X509v3 CRL Reason Code:
      Key Compromise
```

Apache does not reload the certificate revocation list dynamically.

```
[root@training ~]# service pe-httpd restart
Stopping pe-httpd: [ OK ]
Starting pe-httpd: [ OK ]
```

puppet cert clean

```
[root@training ~]# puppet cert clean gary.puppetlabs.vm
notice: Revoked certificate with serial 29
notice: Removing file Puppet::SSL::Certificate gary.puppetlabs.vm at
'/etc/puppetlabs/puppet/ssl/ca/signed/gary.puppetlabs.vm.pem'
notice: Removing file Puppet::SSL::Certificate gary.puppetlabs.vm at
'/etc/puppetlabs/puppet/ssl/certs/gary.puppetlabs.vm.pem'
```

Puppet's Vardir

Contains many files useful for debugging

```
[root@classroom pe-puppet]# ls $(puppet agent --configprint vardir)
bucket      client_data  facts        reports      state
classes.txt client_yaml  lib          rrd          yaml
clientbucket concat      puppet-module server_data
```

- `statedir`
 - Statistics and reports stored by agent
 - Defaults to `$vardir/state`
- `graphdir`
 - Catalog dependency graphs stored by master
 - Defaults to `$vardir/graphs`
- `yamldir`
 - Agent catalogs stored by master
 - Defaults to `$vardir/yaml`

Notes:

- `vardir` defaults to `/var/opt/lib/pe-puppet` on Puppet Enterprise
- `vardir` defaults to `/var/lib/puppet` on Puppet Open Source

Agent Classes

`classes.txt`

Contains all classes applied in plain text from the last run.

```
[root@training ~]# cat $(puppet agent --configprint vardir)/classes.txt
pe_compliance
pe_accounts
settings
classroom.puppetlabs.vm
hosts
puppetdb::params
puppetdb
puppetdb::server
puppetdb::server::firewall
puppetdb::server::database_ini
puppetdb::server::validate_db
puppetdb::server::jetty_ini
puppetdb::database::postgresql
[...]
```

Agent Last Run Summary

last_run_summary.yaml

Last run summary shows abbreviated metrics for the last run.

```
[root@training ~]# tree $(puppet agent --configprint statedir)
├── last_run_report.yaml
├── last_run_summary.yaml
├── resources.txt
└── state.yaml
```

```
[root@training state]# cat last_run_summary.yaml
---
changes:
  total: 1
events:
  total: 1
  success: 1
  failure: 0
[...]
```


Agent Last Run Report

`last_run_report.yaml`

Last run summary shows complete report for the last run.

```
[root@training ~]# tree $(puppet agent --configprint statedir)
├── last_run_report.yaml
├── last_run_summary.yaml
├── resources.txt
└── state.yaml
```

```
[root@training state]# cat last_run_report.yaml
--- !ruby/object:Puppet::Transaction::Report
  configuration_version: 1351925203
  environment: production
  host: classroom.puppetlabs.vm
  kind: apply
  logs:
    - !ruby/object:Puppet::Util::Log
  [...]
```

Agent Resources

resources.txt

Contains all resources titles in plain text from the last run.

```
[root@training ~]# tree $(puppet agent --configprint statedir)
├── last_run_report.yaml
├── last_run_summary.yaml
├── resources.txt
└── state.yaml
```

```
[root@training state]# cat resources.txt
service[postgresql]
file[/etc/puppetlabs/puppet/routes.yaml]
ini_setting[puppetdbserver]
service[pe-httpd]
firewall[5432 accept - postgres]
ini_setting[puppetdb_sslport]
ini_setting[puppetdb_psdatabase_password]
puppetdb_conn_validator[puppetdb_conn]
package[postgresql-server]
exec[/sbin/iptables-save > /etc/sysconfig/iptables]
[...]
```

Catalog Graphs

`relationships.dot`

Graphical representation of catalog dependencies.

```
[root@training ~]# tree $(puppet agent --configprint graphdir)
├── expanded_relationships.dot
├── relationships.dot
└── resources.dot
```

- Graphviz or OmniGraffle are common `dot` file viewers.
- Puppet saves graphs each time a catalog is compiled when `graph` is set to `true`.
- Puppet can also generate a graph by rendering a catalog as `dot` format.

```
[root@training ~]# puppet catalog --render-as dot find localhost > catalog.dot
```

Master's Yaml Node Directory

node/\$certname.yaml

The most recently compiled catalog for all nodes.

```
[root@training ~]# tree $(puppet agent --configprint yamldir)
├── node
│   ├── classroom.puppetlabs.vm.yaml
│   ├── dj.puppetlabs.vm.yaml
│   ├── gary.puppetlabs.vm.yaml
│   ├── instructor.puppetlabs.vm.yaml
│   ├── nan.puppetlabs.vm.yaml
│   ├── proxy.puppetlabs.vm.yaml
│   ├── ralph.puppetlabs.vm.yaml
│   └── ryan.puppetlabs.vm.yaml
```

Network Ports

Puppet uses the following ports:

- 443 - Used for the Puppet Enterprise console
- 8140 - Used for Puppet Master and Puppet Agent configuration
- 8139 - Used for internal communication (deprecated)
- 61613 - Used by Active MQ

Provisioning

Lesson 14: Provisioning

OBJECTIVES

At the end of this lesson, you will be able to:

- Identify common agent provisioning strategies
- Identify REST APIs useful for provisioning
- Set up a Kickstart server to automatically provision agents
- Remotely install Puppet using Cloud Provisioner

Types

Two basic types of provisioning systems

- Bare Metal Provisioning
 - Installs and configures physical hardware
- Cloud Provisioning
 - Installs and configures virtual machines

Bare Metal

Installing and configuring physical hardware

- Almost always local machines
 - often chosen by the security conscious
 - allows more control over network topology
 - high bandwidth or low latency to key servers
 - good choice for machine types that don't virtualize well
 - database servers that require raw disk I/O performance
 - virtual machine host servers
- Requires physical maintenance
- Manual machine setup process

Bare Metal

Kickstart Server

- Installer configuration file(s) requested via HTTP over the network
 - settings for most installation options
 - network information
 - disk partitioning and mounts
 - include post-install script blocks
 - update configuration files
 - download and run installers
 - register with inventory services
- Not cross platform
- Must pass kickstart URI to installer

Exercise 14.1: Configure a Kickstart server



- **Objective:**

- Install a new virtual machine instance using the classroom Kickstart service.

- **Steps:**

- Boot a new virtual machine from an install ISO and enter the Kickstart URI.
- Verify the new agent is installed and configured.

Bare Metal PXE Boot

Automating the install from power on

- Sys admins often set up PXE boot to avoid manual steps
- Network card requests boot image via SFTP
 - can boot installer with parameters directly
 - customizing boot order allows multiple installation schemes
 - fall back to PXE and only install new machines
 - PXE first and let the server decide whether to install or direct the system to boot from the local drive
- Only one PXE server per subnet
- Several projects exist to automate end to end provisioning with PXE

Provisioning Solutions

Lifetime node management solutions

- Cobbler
 - provisioning
 - managing DNS and DHCP
 - package updates
 - orchestration
- Razor
 - provisioning and reclaiming decommissioned hardware
 - rules based classification engine
 - hands off node to configuration management solution
 - still in early development

Virtual Provisioning

Managing virtual machines

- Can scale up to match capacity needs quickly
- Spin up new servers as single purpose virtual machines
 - local VMware ESX cluster
 - OpenStack infrastructure
 - cloud compute services: Amazon EC2, Windows Azure, etc
- Requires little or no physical maintenance
- Usually spin up VMs from templates instead of installing from scratch

Cloud Provisioner

Interface with cloud compute services

- Instantiate instances on cloud compute services
- Fully hosted and self hosted services
 - Amazon EC2
 - VMware ESX
 - OpenStack
 - etc.
- Can remotely install Puppet Enterprise
 - installs using SSH
 - works for local physical hardware too
- Configuration requirements
 - cloud service credentials
 - ssh public key

Exercise 14.2: Remotely Install Puppet



- **Objective:**

- Spin up a new virtual machine and use Cloud Provisioner to install Puppet Enterprise.

- **Steps:**

- Make a copy of the training VM and boot it up.
- Run Cloud Provisioner on your Puppet Master to remotely install Puppet Enterprise on the new VM.

REST API

Issue commands to your Puppet Master

`https://<master>:8140/<environment>/<action>/<certname>`

- Retrieve information with a `GET` request
 - Request a catalog
 - `GET /production/catalog/<certname>`
 - Request node information, including facts
 - `GET /production/node/<certname>`

REST API

Issue commands to your Puppet Master

`https://<master>:8140/<environment>/<action>/<certname>`

- Set information with a PUT or DELETE request
 - Sign agent certificates
 - PUT /production/certificate_status/<certname>
 - `payload: {"desired_state": "signed"}`
 - Decommission a node:
 - PUT /production/certificate_status/<certname>
 - `payload: {"desired_state": "revoked"}`
 - DELETE /production/certificate_status/<certname>

REST API

Call API from the command line

- Setting up the cURL connection

```
curl --cert /etc/puppetlabs/puppet/ssl/certs/<certname>.pem \
      --key  /etc/puppetlabs/puppet/ssl/private_keys/<certname>.pem \
      --cacert /etc/puppetlabs/puppet/ssl/ca/ca.crt.pem -H 'Accept: yaml' \
```

- Can reuse agent certificates
- Can accept `yaml` or `json` as return types
- Defaults to `GET` request
- Make a `PUT` request and send data
 - `-X PUT --data '{"desired_state":"revoked"}'`
- Make a `DELETE` request
 - `-X DELETE`

REST API

Call API from Ruby scripts

Use the `net/https` library to make secure REST calls

```
require 'yaml'
require "net/https"

http = Net::HTTP.new(@server, 8140)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_PEER

store = OpenSSL::X509::Store.new
store.add_cert(OpenSSL::X509::Certificate.new(File.read("#{@confdir}/ca.crt.pem")))
http.cert_store = store

http.key = OpenSSL::PKey::RSA.new(File.read("#{@confdir}/#{certname}.key"))
http.cert = OpenSSL::X509::Certificate.new(File.read("#{@confdir}/#{certname}.pem"))

request = Net::HTTP::Get.new("/production/catalog/#{agent}")
request["Accept"] = 'yaml'

puts http.request(request).body
```

auth.conf

Allow access to REST endpoints

- Allow provisioner to sign new agent certificates

```
# /etc/puppetlabs/puppet/auth.conf
path /certificate_status
method find, search, save, destroy
auth yes
allow pe-internal-dashboard,provisioner.example.com
```

Notes:

The `auth.conf` file is managed in Puppet Enterprise 2.7.0, so you will need to add an entry with the following Puppet code:

```
auth_conf::acl { 'certificate<em>status</em>override':
  path      => '/certificate<em>status',
  auth      => 'yes',
  acl~~~CONTENT~~~lt;/em>method => ['find','search', 'save', 'des
  allow     => ['pe-internal-dashboard', 'provisioner'],
  order     => 083,
}
```

When managing the `auth.conf` entry via Puppet code, the `order` attribute is critical. We are adding an entry to the file to come before the entry inserted by the standard modules. The file is parsed on a first-match basis, so we have to ensure that our entry comes first.

Exercise 14.3: Request facts with `curl`



- **Objective:**

- Request a list of facts from the Puppet Master using the `curl` command line binary.

- **Steps:**

- Locate your public certificate, private key, and CA certificate.
- Construct the appropriate `curl` command to request your facts from the classroom master.

MCollective

Lesson 15: MCollective

OBJECTIVES

At the end of this lesson, you will be able to:

- Identify core technologies in MCollective.
- Review configuration storage locations.
- Recall Puppet MCollective terminology.

Functionality

Benefits you can achieve with MCollective

- A framework to build custom orchestration tools
- Eliminate complex naming conventions for hostnames as a means of identity
- Simple to use command line tools to call remote agents
- Produce custom reports about your infrastructure
- Reusable agents to manage packages, services, Puppet and other common components
- Built in Authentication, Auditing, and Authorization
- Extremely pluggable and adaptable to local needs

Pluggable Core

Extensible Framework

- Develop custom MCollective Agents
- Add sources of data
- Leverage MCollective as transport to provide:
 - central inventory of hardware & services
 - central infrastructure status report
- Replace our choice of middleware
 - STOMP compliant

Notes:

STOMP Protocol

- Streaming Text Oriented Messaging Protocol
- Simplified protocol, quite human readable
- Language and Transport agnostic

Technologies

- There are numerous technologies often used with with MCollective:
 - ActiveMQ
 - RabbitMQ
 - stompserver
- And other variations that use MoM but are not specifically supported yet:
 - ZeroMQ
 - HornetQ
 - StormMQ
 - JBoss Messaging
 - IBM Websphere MQ

MCollective Middleware

- MCollective depends on middleware to broker messages between Client and Servers.
 - MCollective Servers subscribe to middleware topic and respond to RPC requests when appropriate.
 - MCollective Clients publish messages to the queue to issue actions to Servers.
- Default configuration requires Stomp protocol support.
- Middleware Software:
 - ActiveMQ
 - RabbitMQ
 - Stompserver

Middleware

- Wiki Definition: Middleware is computer software that connects software components or some people and their applications.
- MCollective uses Message-oriented Middleware (MoM) specifically.
- It is used for transporting messages between clients and servers.
- It has no programming logic itself; it's more like an SMTP or IRC server.

Message-oriented Middleware Advantages

- Transport is separated from components
 - Simple semantics: send, subscribe, publish often abstracted by API
- Good at broadcast messages and addressing groups of peers
 - Topics can broadcast
 - Queues can load-balance
- Asynchronous:
 - Messages can queue in middleware if components are down
 - Good support for long running jobs
 - Short jobs can also be handled

Middleware Terminology

- Middleware
 - A publish subscribe based system like Apache ActiveMQ
 - Middleware Broker
 - A message queuing bus that transmits messages between peers
 - Middleware Topic
 - messages sent to a topic are received by all subscribers
 - *subscriber* consumes messages from a topic
 - *publisher* writes messages to a topic
-

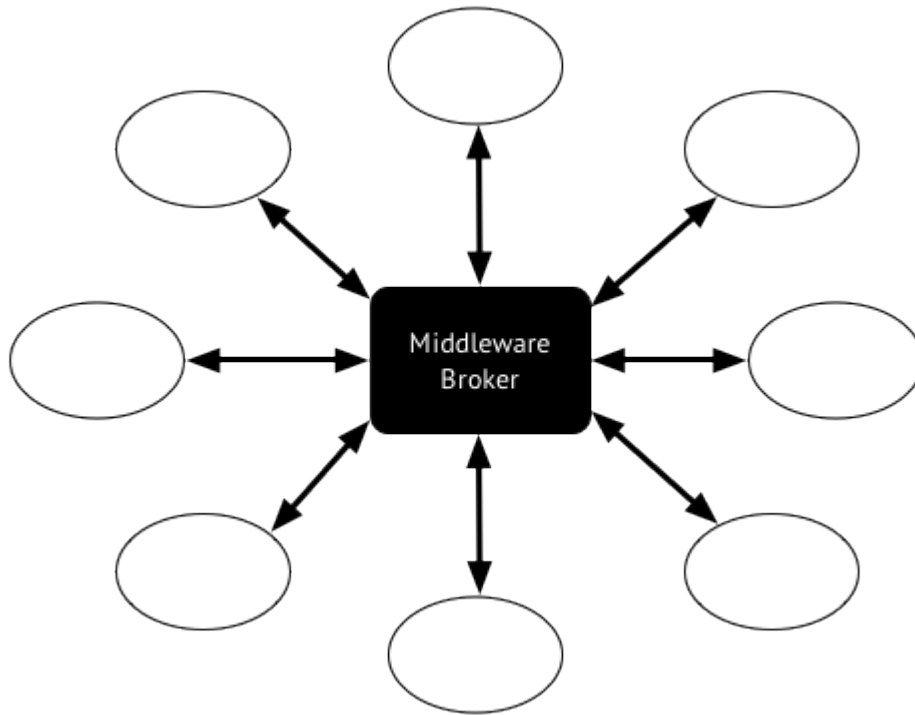
Notes:

- Messages are sent to the middleware directed at topics named
 - `/topic/mcollective.package.command`
- Replies come back on
 - `/topic/mcollective.package.reply.`

It is possible to add namespaces to MCollective to run multiple collectives, but that is beyond the scope of this class.

Architecture

Message Oriented Middleware

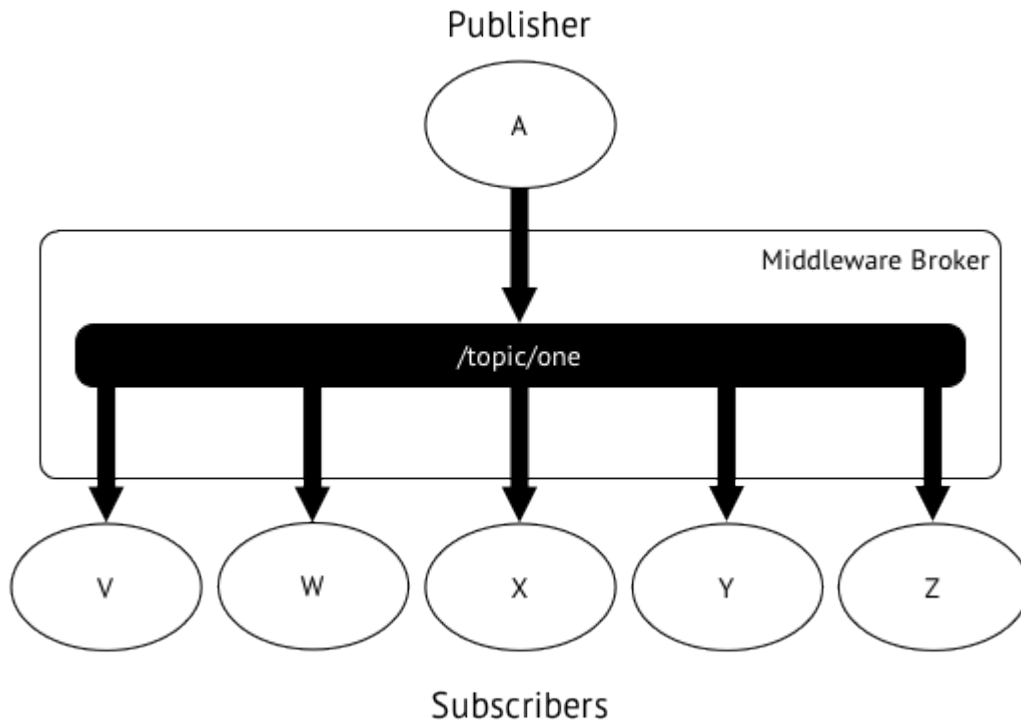


Notes:

- The default middleware broker is ActiveMQ
- Each agent, including the Puppet Master, is both a publisher and a subscriber

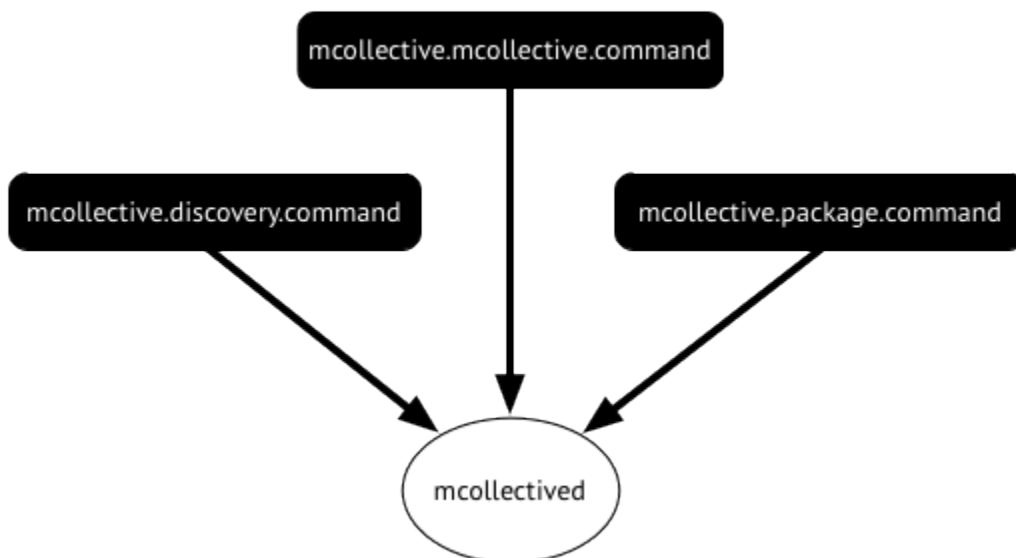
Topics

With topics, peers can become subscribers, publishers or both.



Notes:

`mcollectived` subscribes to a topic for each plugin it has, plus a control queue 'mcollective'.



MCollective Messages

- Agent Request Message

```
{ :filter      => {  
  "cf_class"   => [ "common::linux" ],  
  "fact"       => [ { :fact => "country", :value => "uk" } ],  
  "agent"      => [ "package" ]  
},  
:senderid      => "devel.your.com",  
:msgtarget     => "/topic/mcollective.discovery.command",  
:body          => body,  
:hash          => "2d437f2904980ac32d4ebb7ca1fd740b",  
:msgtime       => 1258406924,  
:requestid     => "0b54253cb5d04eb8b26ea75bbf468cbc"  
}
```

- Agent Reply

```
{ :senderid      => "devel.your.com",  
  :senderagent   => "package",  
  :msgtarget     => "/topic/mcollective.package.reply",  
  :body          => body,  
  :hash          => "2d437f2904980ac32d4ebb7ca1fd740b",  
  :msgtime       => 1258406924,  
  :requestid     => "0b54253cb5d04eb8b26ea75bbf468cbc"  
}
```

Component Terminology

- Server:
 - app server for hosting Agents and managing middleware connection
 - runs on *every* peer, Puppet Master and Agent both
- Agent:
 - block of Ruby code that performs a specific set of tasks, or Actions
 - runs as a plugin of the server
- Action:
 - tasks exposed by an Agent
 - an exim queue agent might expose actions like `mailq`, `rm`, `retry`, etc.
- Application:
 - MCollective command plugin that sends messages to agents

Using MCollective

- Primary interaction is via commandline
- `mco` executable brokers your commands to application plugins

```
peadmin@training:~$ mco ping
dev8                time=126.19 ms
dev6                time=132.79 ms
dev10               time=133.57 ms
.
.

---- ping statistics ----
25 replies max: 305.58 min: 57.50 avg: 113.16
```

- `ping` is the application called by the `mco` executable
- `mco help` will generate a list of application plugins

The Puppet Enterprise Console can automatically discover agent plugins and present a graphical interface for invoking them.

Notes:

Puppet Enterprise is configured with a `peadmin` user for running MCollective commands. To open a shell as the `peadmin` user, run: `sudo -i -u peadmin`

Structure of a Typical Command

`mco` [APPLICATION] [ACTION] [ARGUMENTS] [OPTIONS]

- APPLICATION
 - the name of the application you intend to run
- ACTION
 - The action that should be passed to listening agents
- ARGUMENTS
 - Arguments that should be passed to the action
- OPTIONS
 - Options to the `mco` command

Example of a Command

```
peadmin@training:~$ mco inventory agent.example.com
Inventory for agent.example.com:
Server Statistics:
  Version: 1.2.1
  Start Time: Mon Nov 14 15:25:33 -0800 2011
  Config File: /etc/puppetlabs/mcollective/server.cfg
  Collectives: mcollective
  Main Collective: mcollective
  Process ID: 5553
  Total Messages: 86
  Messages Passed Filters: 74
  Messages Filtered: 12
  Replies Sent: 73
  Total Processor Time: 1.34 seconds
  System Time: 0.59 seconds
Agents:
  discovery      package      puppetd
  puppetral      rpcutil     service
Configuration Management Classes:
  default        helloworld
  helloworld     pe_accounts
  pe_accounts    pe_accounts::data
  ...
```

- Can you identify the command components?

Notes:

The application is `inventory` and the argument is `agent.example.com`. The `inventory` application implies the action. Some applications will require you to specify an action.

Filtering Nodes

- Perform actions on a subset of your infrastructure
- Filter based on rich metadata, not complex hostname rules
 - Facter facts (`--with-fact` or `-F`)
 - Puppet classes (`--with-class` or `-C`)
 - *more...*
- Filters are case sensitive and support regular expressions.

```
peadmin@training:~$ mco ping --with-fact osfamily=RedHat
puppetmaster.example.com      time=135.94 ms
agent.example.com             time=136.55 ms
---- ping statistics ----
2 replies max: 136.55 min: 135.94 avg: 136.25
```

MCollective Documentation

Built in help subsystem for Applications

- `mco help`: List applications available on a system
- `mco <application> -help`: Display application documentation

```
peadmin@training:~$ mco rpc --help
Generic RPC agent client application

Usage: mco rpc [options] [filters] --agent <agent> --action <action> [--argument <key=val> ...]
Usage: mco rpc [options] [filters] <agent> <action> [<key=val> <key=val> ...]

    --no-results, --nr          Do not process results, just send request
-a, --agent AGENT              Agent to call
    --action ACTION            Action to call
    --arg, --argument ARGUMENT Arguments to pass to agent

    --np, --no-progress        Do not show the progress bar
-1, --one                      Send request to only one discovered nodes
    --batch SIZE               Do requests in batches
    --batch-sleep SECONDS      Sleep time between batches
...
```

MCollective Documentation

Built in help subsystem for Agents

- `mco help <agent>`: Display agent documentation

```
peadmin@training:~$ mco help service
Service Agent
=====

Start and stop system services

    Author: R.I.Pienaar
    Version: 1.2
    License: ASL2
    Timeout: 60
    Home Page: https://github.com/puppetlabs/mcollective-plugins

ACTIONS:
=====
    restart, start, status, stop
    ...
```


Puppet Integration

MCollective applications that leverage Puppet functionality

- `puppetd`
 - Manages the on-demand running of your Puppet Agents
- `package`
 - Uses Puppet RAL to manage package state
- `service`
 - Uses Puppet RAL to manage service state

Puppet Integration

Example `puppetd` Usage

- Run Puppet on 5 Debian nodes at a time:
 - `mco puppetd runall -F osfamily=Debian 5`
- Temporarily disable Puppet on `dev` classified nodes:
 - `mco puppetd disable -C dev`
- Retrieve the last run summary from a node:
 - `mco puppetd summary -I proxy.puppetlabs.vm`

Puppet Integration

Example package Usage

- Install a package on classified nodes:
 - `mco package install httpd -C backend`
- Update a package on a named node:
 - `mco package update httpd -I alphonse.puppetlabs.vm`
- Retrieve package versions from all RedHat machines:
 - `mco package status httpd -F osfamily=RedHat`

Puppet Integration

Example `service` Usage

- Restart a service on all `travis` classified nodes
 - `mco service travis-ci restart -C travis`
- Stop a service on a named node
 - `mco service exim stop -I compromised.puppetlabs.vm`
- Discover how many mailservers are running:
 - `mco service exim status`

Lab 15.1: Control Puppet via MCollective



- **Objective:**

- Using MCollective, manage the state of your Puppet Agent

- **Steps:**

- Remotely enable and disable your Puppet Agent
- Trigger a Puppet Agent run using MCollective
- Filter commands based on facts and classes

MCollective Plugins

- MCollective supports a variety of plugins including
 - Agents
 - Fact Sources
 - Auditing
 - Authorization
 - Discovery

SimpleRPC Agents

- Abstracts away complexity
- Provides convention and standards
 - *favoring convention over custom design*
- Very easy to write agents
 - input validation
 - sensible feedback mechanism
 - audit logging
 - authorization
 - etc...

Agent Files

- SimpleRPC agents can consist of
 - The agent code which gets executed
 - A metadata file to document the agent and provide validation
 - An optional application to provide subcommand to `mco`

```
peadmin@training:~$ tree /opt/puppet/libexec/mcollective
└─ mcollective
   └─ agent
      ├── service.ddl
      └─ service.rb
   └─ application
      └─ service.rb
```

Notes:

MCollective agents are not distributed via `pluginsync`, but you can use a Puppet module to install the agent where required

```
class custom::mcollective::plugins(
  $dir='/opt/puppet/libexec/mcollective/mcollective'
) {
  file { ["${dir}/agent/myagent.ddl":
    source => 'puppet:///modules/custom/myagent.ddl',
  ]
  file { ["${dir}/agent/puppetd.rb":
    source => 'puppet:///modules/custom/myagent.rb',
  ]
  file { ["${dir}/application/myapplication.rb":
    source => 'puppet:///modules/custom/myapplication.rb',
  ]
}
```


Configuration

- MCollective
 - `server: /etc/puppetlabs/mcollective/server.cfg`
 - `client: /etc/puppetlabs/mcollective/client.cfg`
- ActiveMQ: `/etc/puppetlabs/activemq/activemq.xml`
 - Persistence
 - Queue configuration
 - Security
 - SSL
- ActiveMQ Web Console: `/etc/puppetlabs/activemq/jetty.xml`
 - Management Interface

Inventory Reports

- Producing live inventory reports of your entire infrastructure is robust and scriptable.
- Styles
 - printf
 - perl
- Running inventory scripts
 - `mco inventory --script inventory.mc`

Custom Inventory Scripts

- printf style
 - Basic and quick to create
 - Needs no other dependencies.
 - Needs just a display format and fields to include.

```
# inventory.mc
inventory do
  format "%s:\t\t%s\t\t%s"

  fields { [ identity, facts["serialnumber"], facts["productname"] ] }
end
```

```
peadmin@training:~$ mco inventory --script inventory.mc
classroom.puppetlabs.vm:    VMware-56 ...      VMware Virtual Platform
alphonse.puppetlabs.vm:    VMware-56 ...      VMware Virtual Platform
proxy.puppetlabs.vm:       VMware-56 ...      VMware Virtual Platform
```

Custom Inventory Scripts

perl

- Requires the `formatr` gem.
- Can produce very readable and printable reports.
- Need to know `perlform` syntax.
 - <http://perldoc.perl.org/perlform.html>

```
peadmin@training:~$ mco inventory --script inventory.mc

Node Report Sun Jun 02 14:00:26 +0000

Hostname:      Customer:      Distribution:
-----

alphonse.puppetla
                                Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz

proxy.puppetlabs.
                                Ubuntu 12.04.1 LTS
                                Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz

classroom.puppetl
                                Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
```




Lab 15.2: Inventory Reports

- **Goal:**

- Use an inventory script to gather inventory information about the classroom.

- **Steps:**

- Build a basic `printf` style report
- Display `ipaddress`, `fqdn`, `memorysize` and `memoryfree`
- Run report against all nodes in the classroom

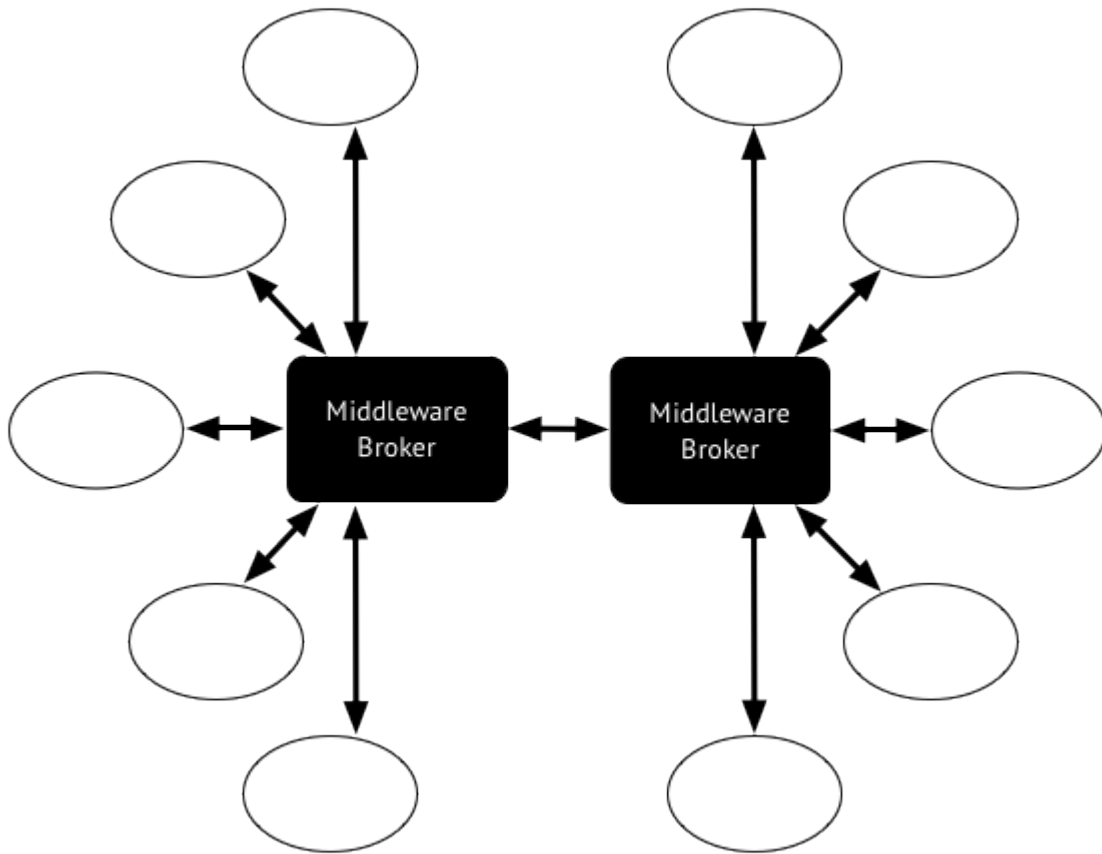
- **Extra Credit**

- Write a nicely formatted `perlform` report to do the same

Network of Brokers

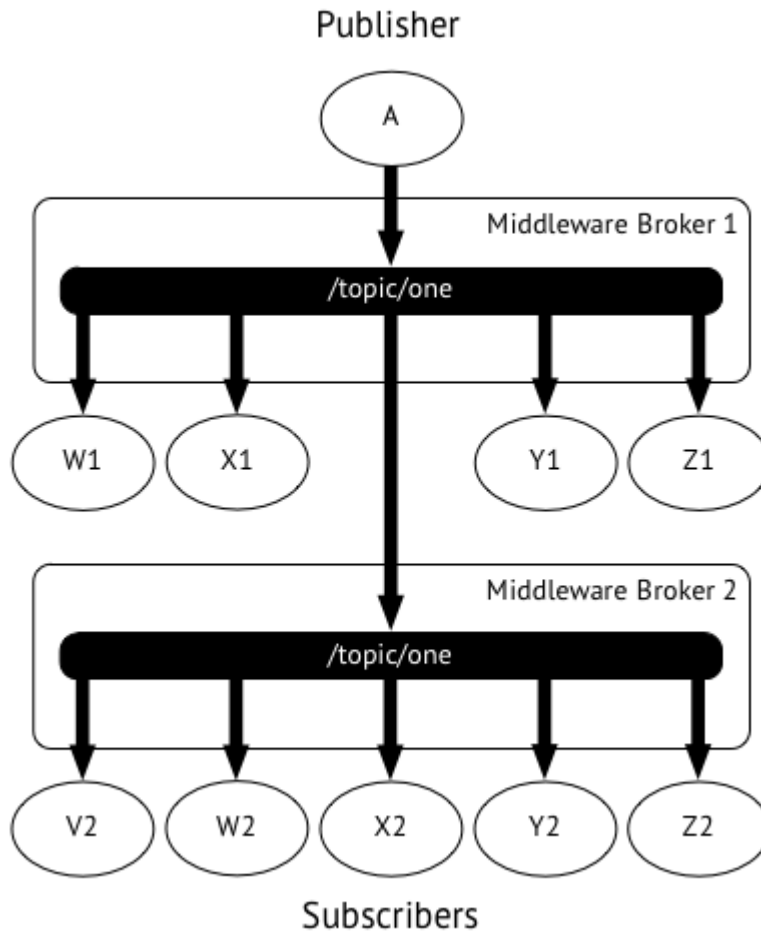
- Works by providing a scalable network or mesh of multiple brokers
- Brokers in the mesh are active
- Messages are intelligently passed between brokers to clients subscribed to topics or queues globally
- Implementation of routing and forwarding is transparent to the user

Network Diagram



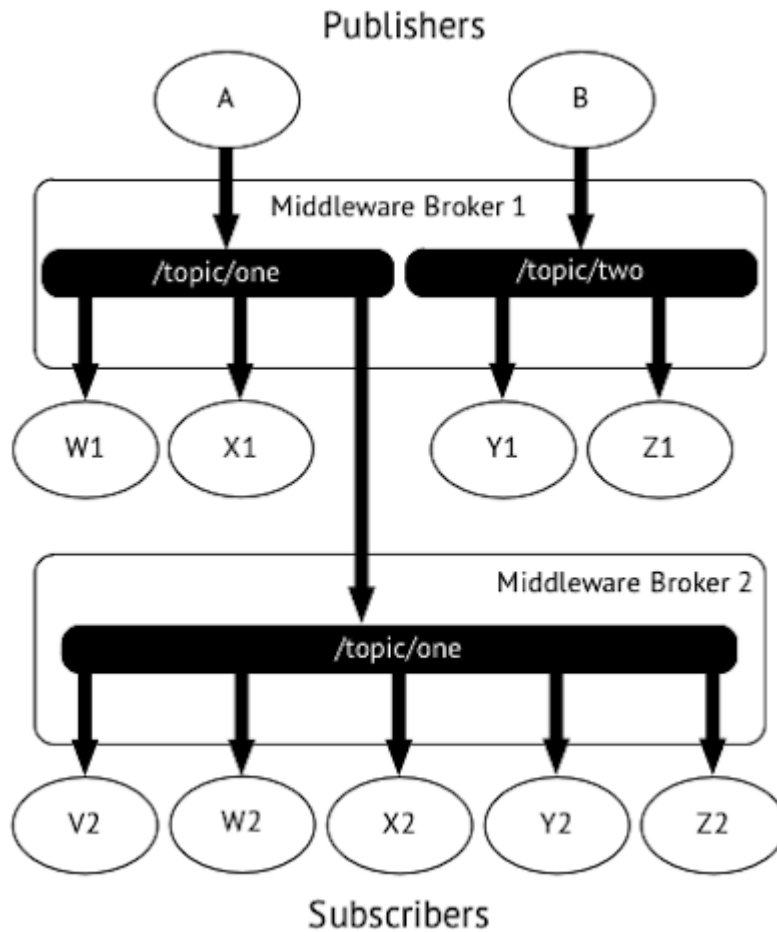
Message Forwarding

- With topics, messages are forwarded to all brokers where there are subscribers.



Message Forwarding

- Messages don't get forwarded to brokers where there are no subscriptions for the topic.



Dynamic Configuration

- Dynamic configuration uses multicast to discover other nodes.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://activemq.org/config/1.0">
  <broker>
    ...

    <networkConnectors>
      <networkConnector uri="multicast://default"/>
    </networkConnectors>

    <transportConnectors>
      <transportConnector name="openwire" uri="tcp://0.0.0.0:6166"/>
      <transportConnector name="stomp" uri="stomp://0.0.0.0:6163"/>
      <transportConnector
        userName="admin"
        password="secret"
        uri="tcp://<hostname_of_box>:0"
        discoveryUri="multicast://default"
      />
    </transportConnectors>

    ...
  </broker>
</beans>
```

Static Configuration

- Static configuration allows you to specify other brokers manually

```
<beans>
  <broker>
    ...
    <networkConnectors>
      <networkConnector
        name="broker1-broker2"
        uri="static: (tcp://broker2.your.net:6166) "
        userName="amq" password="secret" duplex="true"
        decreaseNetworkConsumerPriority="true"
        networkTTL="2" dynamicOnly="true"
      />
      <networkConnector
        name="broker1-broker3"
        uri="static: (tcp://broker3.your.net:6166) "
        userName="amq" password="secret" duplex="true"
        decreaseNetworkConsumerPriority="true"
        networkTTL="2" dynamicOnly="true"
      />
    </networkConnectors>

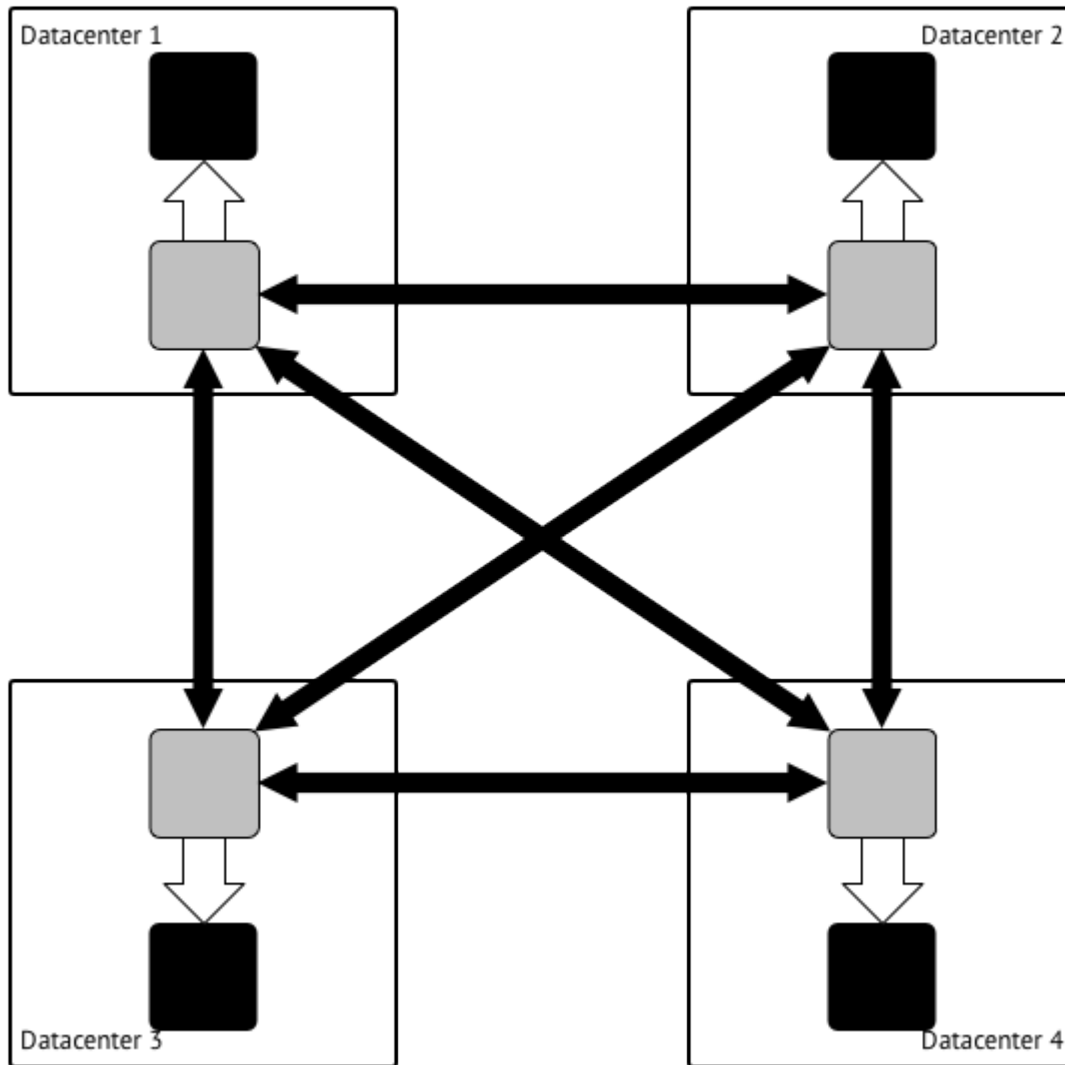
    <transportConnectors>
      <transportConnector name="openwire" uri="tcp://0.0.0.0:6166"/>
      <transportConnector name="stomp" uri="stomp://0.0.0.0:6163"/>
    </transportConnectors>

    ...
  </broker>
</beans>
```

Notes:

- Each node requires configuration to specify its peers.

MCollective Mesh



Recommendations

- Place an ActiveMQ broker in each WAN location
 - localized queues
 - isolate requests to only a chosen WAN location or subnet
 - helps when network segments become isolated
- Network of broker layout can work around firewalls, but...
 - avoid overly complex configurations
 - too many brokers in a network can cause more problems then it solves
- KahaDB or Pure Master/Slave replication in each WAN location for resiliency
- Other Master Slave configurations are complex and require more components

Capstone Lab

Course Satisfaction Survey

Advanced Puppet for Puppet Masters



At this time, we would appreciate any feedback you can share.
Please complete our course survey to get a free Puppet T-shirt!!

<http://www.puppetlabs.com/classfeedback>

or

<http://puppetlabs.classfeedback.sgizmo.com/s3/>

Capstone Lab: Advanced Puppet



There are two Capstone labs to choose from for this course. Details for each are in your exercise guide.

- **Objectives**

- A: Automate adding a Puppet Master to a group Load Balancer
- B: Use modules from the Forge to automate a vhost environment

- **Steps**

- A: Add a new master/agent virtual machine to the HA Proxy pool
- B: Using Hiera create virtualhosts and monitor them using Nagios

- **Extra Credit**

- Add functionality to the new servers as desired
- Tackle the second Capstone lab selection as time permits

Discuss with your instructor and your group which one to complete.

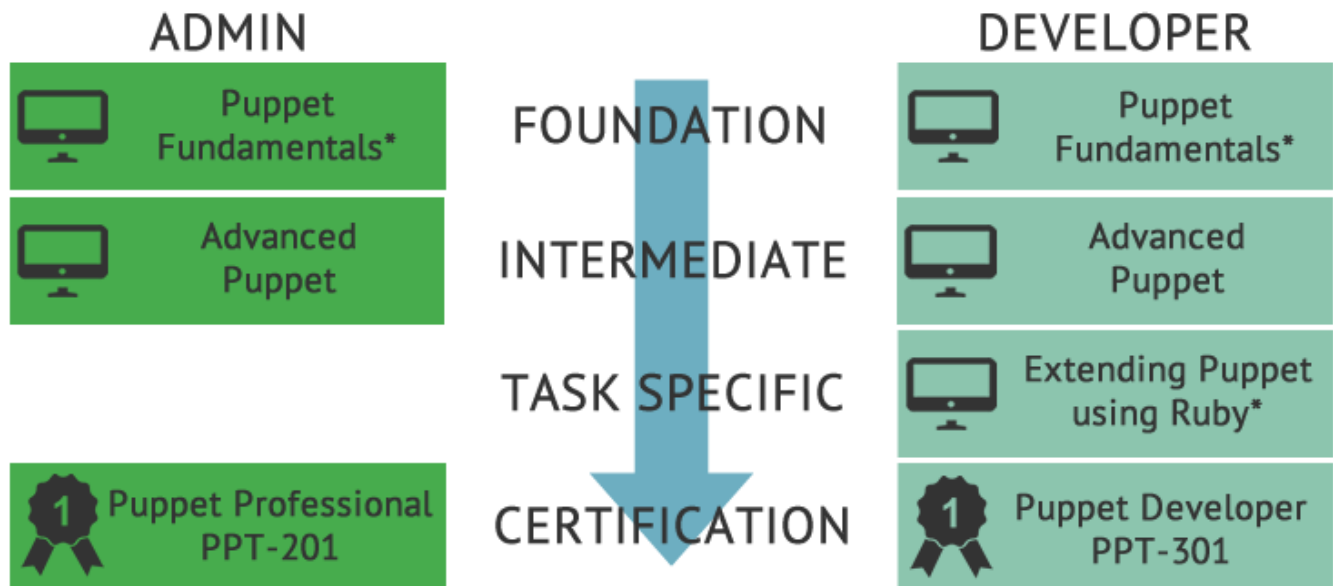
Course Conclusion

Course Summary

During this class, we built a production style environment with:

- Hiera as a backend data source.
- PuppetDB as a replacement for storeconfigs.
- Multiple masters using a shared Certificate authority.
- A series of test modules based on core principles.

Puppet Education Roadmap



* recommended preparation for Certification Exam

Help shape Puppet ...



Puppet Test Pilot Program

PARTICIPATE IN USER RESEARCH.
INFLUENCE PRODUCTS.
GET T-SHIRTS AND GIFT CARDS.

Sign up at puppetlabs.com/ptp

Thank You



Appendix

Glossary

define

To specify the contents and behavior of a class or a defined resource type. Defining a class or type doesn't automatically include it in a configuration; it simply makes it available to be declared.

declare

To direct Puppet to include a given class or resource in a given configuration. To declare resources, use the lowercase file `{"/tmp/bar":}` syntax. To declare classes, use the `include` keyword or the class `{"foo":}` syntax.

idempotent

Able to be applied multiple times with the same outcome.

Fact

A piece of information about a node, such as its operating system, hostname, or IP address. You can extend the list of available facts with custom facts, which can expose site-specific details of your systems to your Puppet manifests.

Facter

Puppet's system inventory tool. Facter reads facts about a node (such as its hostname, IP address, operating system, etc.) and makes them available to Puppet.

Function

A statement in a manifest which returns a value or makes a change to the catalog. You can choose from the list of built-in functions, use functions from public modules (like `puppetlabs-stdlib`), or write our own custom functions.

Hiera

A key/value lookup tool for configuration data, built to make Puppet better and let you set node-specific data without repeating yourself.

Appendix

Provider

Providers implement resource types on a specific type of system, using the system's own tools. The division between types and providers allows a single resource type (like `package`) to manage packages on many different systems (using, for example, `yum` on Red Hat systems, `dpkg` and `apt` on Debian-based systems, and `ports` on BSD systems).

Type

A kind of resource that Puppet is able to manage; for example, `file`, `cron`, and `service` are all resource types. A type specifies the set of attributes that a resource of that type may use, and models the behavior of that kind of resource on the target system. You can declare many resources of a given type.

Puppet Module Cheat Sheet

- Modules are directories with a predictable structure.
- Puppet can automatically load manifests, files, and plugins from modules in its modulepath.
- Use `puppet --configprint modulepath` to see where Puppet expects to find modules on your system.

Example Module: `/etc/puppetlabs/puppet/modules/apache`

manifests

This directory holds the module's Puppet code.

- Each `.pp` file should contain one and only one class or defined type.
- Filenames and class/defined type names are related; see the examples below.
- Within a module, the special `$module_name` variable always contains the module's name.

`apache/manifests/init.pp`

```
class apache {  
  ...  
}
```

`Init.pp` is special; it should contain a class (or define) with the same name as the module.

`apache/manifests/vhost.pp`

```
define apache::vhost ($port, $docroot) {  
  ...  
}
```

Other classes (and defines) should be named `modulename::filename` (without the `.pp` extension).

`apache/manifests/config/ssl.pp`

```
class apache::config::ssl {  
  ...  
}
```

Subdirectories add intermediate namespaces.

lib

This directory holds Ruby plugins, which can add features to Puppet and Facter.

`apache/lib/puppet/type/apache_setting.rb`

A custom type.

`apache/lib/puppet/parser/functions/htpasswd.rb`

A custom function.

`apache/lib/facter/apache_confdir.rb`

A custom fact.

files

Nodes can download any files in this directory from Puppet's built-in file server.

- Use the `source` attribute to download file contents from the server.
- Use `puppet:///` URIs to specify which file to fetch.
- Files in this directory are served at `puppet:///modules/modulename/filename`.

`apache/files/httpd.conf`

To fetch this file:

```
file {'/etc/apache2/httpd.conf':  
  ensure => file,  
  source => 'puppet:///modules/apache/httpd.conf',  
}
```

`apache/files/extra/ssl`

Puppet's file server can navigate any subdirectories:

```
file {'/etc/apache2/httpd-ssl.conf':  
  ensure => file,  
  source => 'puppet:///modules/apache/extra/ssl',  
}
```

templates

This directory holds ERB templates.

- Use the `template` function to create a string by rendering a template.
- Use the `content` attribute to fill file contents with a string.
- Template files are referenced as `modulename/filename.erb`.

`apache/templates/vhost.erb`

To use this template:

```
file {'/etc/apache2/sites-enabled/wordpress.conf':  
  ensure => file,  
  content => template('apache/vhost.erb'),  
}
```

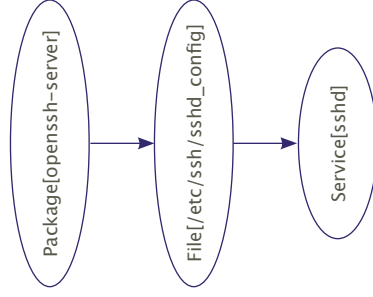

THE TRIFECTA

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, allowedrom, 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.

HELLO WORLD

notify

Sends an arbitrary message to the agent run-time log.

```
notify { "This message is getting logged
on the agent node.": }

notify { "Mac warning":
  message => $operatingsystem ? {
    'Darwin' => "This seems to be a
Mac.",
    default => "And I'm a PC.",
  },
}
```

ATTRIBUTES

- message — **Defaults to title.**
- See also: withpath

GRAB BAG

exec

Executes an arbitrary command on the agent node. When using execs, make sure the command can be safely run multiple times or specify that it should only run under certain conditions.

ATTRIBUTES

- command — The command to run; **defaults to title**. If this isn't a fully-qualified path, use the path attribute.
- path — A search path for executables; colon-separated list or an array. This is most useful as a resource default, e.g.:

```
Exec {
  path => [
    '/usr/local/bin',
    '/opt/local/bin',
    '/usr/bin',
    '/usr/sbin',
    '/bin',
    '/sbin'],
  logoutput => true,
}

exec { 'pwd': }
exec { 'whoami': }
```

- creates — A file created by this command; if the file exists, the command won't run.
- refreshonly — If true, the exec will only run if a resource it subscribes to (or a resource which notifies it) has changed.
- true or false
- onlyif — A command or array of commands; if any have a non-zero return value, the command won't run.
- unless — The opposite of onlyif.
- environment — An array of environment variables to set (e.g. ['MYVAR=somevalue', 'OTHERVAR=othervalue']).
- See also: cwd, group, logoutput, returns, timeout, tries, try_sleep, user.

cron

Manages cron jobs. Largely self-explanatory.

```
cron { logrotate:
  command => "/usr/sbin/logrotate",
  user => root,
  hour => 2,
  minute => 0
}
```

ATTRIBUTES

- command — The command to execute.
- ensure — Whether the job should exist.
- present
- absent
- hour, minute, month, mthday, and weekday — The timing of the cron job.
- See also: environment, name, provider, special, target, user.

user

Manages user accounts; mostly used for system users.

```
user { "dave":
  ensure => present,
  uid => '507',
  gid => 'admin',
  shell => '/bin/zsh',
  home => '/home/dave',
  managehome => true,
}
```

ATTRIBUTES

- name (**defaults to title**)
- uid — The user ID. Must be specified numerically; chosen automatically if omitted.
- ensure — Whether the user should exist.
- present
- absent
- role
- gid — The user's primary group. Can be specified numerically or by name.
- groups — An array of secondary groups to which the user belongs. (Don't include the group specified as the GID.)
- home — The user's home directory.
- managehome — Whether to manage the home directory when managing the user; if you don't set this to true, you'll need to create the user's home directory manually.
- true or false
- shell — The user's login shell.
- See also: allowdupe, auths, comment, expiry, key_membership, keys, membership, password, password_max_age, password_min_age, profile_membership, profiles, project, provider, role_membership, roles.

group

Manages groups.

ATTRIBUTES

- name (**defaults to title**)
- gid — The group ID; must be specified numerically, and will be chosen automatically if omitted.
- ensure — Whether the group should exist.
- present
- absent
- See also: allowdupe, auth_membership, members, provider.

EVERYTHING ELSE

You are ready. Go check the types reference at <http://docs.puppetlabs.com/references/latest/type.html>

Certificate of Course Completion



_____ attended _____ hours of

Puppet Labs training and completed the

Advanced Puppet course.

(Instructor Signature)

(Date)