Linux Academy

Learning Puppet

Infrastructure Automation With Puppet

Linux Academy

Learning Puppet

What are common issues in traditional environments?

- Provisioning of new nodes (servers) when needed
- Configuration consistency among different nodes
- Custom scripts are written for specific operating systems and environments
- Managing packages and configuration across thousands of nodes
- Managing large amounts of nodes becomes expensive and unmanageable

Learning Puppet

What is Puppet?

- Infrastructure automation and configuration management tool
- Enforces the defined state of the infrastructure
- Puppet can automate tasks on thousands of machines, A.K.A “nodes”
- Puppet enables infrastructure as code and *DevOps*
- Puppet allows configuration consistency across nodes
- Puppet enables quick provisioning of new machines in an environment

Learning Puppet

What is Puppet?

- Puppet allows DevOp admins to write declarative instructions using the Puppet language
- DevOp admins write code using the Puppet DSL to express the desired state of a node
- Code is written inside of *classes* and classes are assigned to node
- Node classification is the process of assigning a class to a node for processing
- Puppet was created by Luke Kanies in 2003 written with the Ruby language

Learning Puppet

More advanced Puppet

- Puppet is advanced and there is a lot more to it such as:
  - Code testing (acceptance testing) tools such as *beaker* developed by puppet
  - Puppet forge
  - Puppet cloud provisioning
  - Writing custom puppet modules to extend puppet

Linux Academy.com

Learning Puppet

Example:

Puppet language DSL

```
{

class motd {
  file { "/etc/motd":
    ensure => 'file',
    source => "puppet:///modules/motd/motd",
  }
}
```

Simple class that defines a file resource

Linux Academy

Learning Puppet

About This Course

Linux Academy

Learning Puppet

About this course

- Instructor Anthony James, Puppet Certified
- Will focus on:
    - Core concepts as they relate to Puppet and Enterprise Puppet Server
    - Getting started building classes and implementing puppet in a production env
    - Will utilize LinuxAcademy.com lab servers during the course and learning the DSL
    - Will address installing Puppet Master and Agents in your own environment
    - Will cover fundamental requirements for the Puppet professional certification
    - After the concepts of the course we will learn puppet by building classes:
        - MOTD
        - Local users management
        - Apache configuration and virtual hosts

Linux Academy

Learning Puppet

Puppet Enterprise Stack

Puppet Master

- Compile and serve configuration catalogs to agent nodes
- Issue orchestration commands via MCollective
- Availability of puppet enterprise web interface console
- Collect data and compile reports from agent nodes
- Manage environments, assign node configurations and manage code with GIT

What is a catalog?

The catalog is a document downloaded from the puppet master to the agent that describes the desired state for each resource that should be managed. It may also specify dependency information for the resources that should be managed in a certain order. This is essentially a compiled version of the DSL and is compiled on the Puppet master and stored in PuppetDB if PuppetDB is used.

A Catalog is used to:

- Reduce the node agents resource consumption (it's compiled on master and downloaded to the node)
- Simulate changes on a node (locally compiled a catalog using puppet apply)
- If used with PuppetDB you can query the DB based off nodes managed resources (managed by Puppet

Puppet Enterprise Orchestration Engine

Puppet enterprise includes MCollective which is an orchestration system that allows the admin to invoke different commands in *parallel* across a select group of nodes.

Common Tasks With Orchestration:

- Control Puppet
- Browse and search resources
- Execute actions on specific nodes or all nodes

Note: Puppet Enterprise Agent is required for orchestration tasks

Learning Puppet

Puppet Process

- Install Puppet master and Puppet agents
- Create configurations (modules/classes) that declare a resources desired state
- Assign configurations to nodes (also known as declaring a class)

  Configurations can be assigned/declared in the console

  Configurations can be assigned/declared in /etc/puppetlabs/puppet/manifests/site.pp

  Note: All of this is covered in-depth as part of this course

Linux Academy

Learning Puppet

Nodes

Learning Puppet

Puppet nodes (agent)

Nodes are any virtual or physical system that is able to run Puppet Agent and is specifically supported by puppet agent (operating system support on next slide)

Required Ports:

- 8140 for puppet requests
- 61613 for orchestration requests

Currently supported operating systems (Fall 2014)

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

For latest: https://docs.puppetlabs.com/pe/latest/install_system_requirements.html

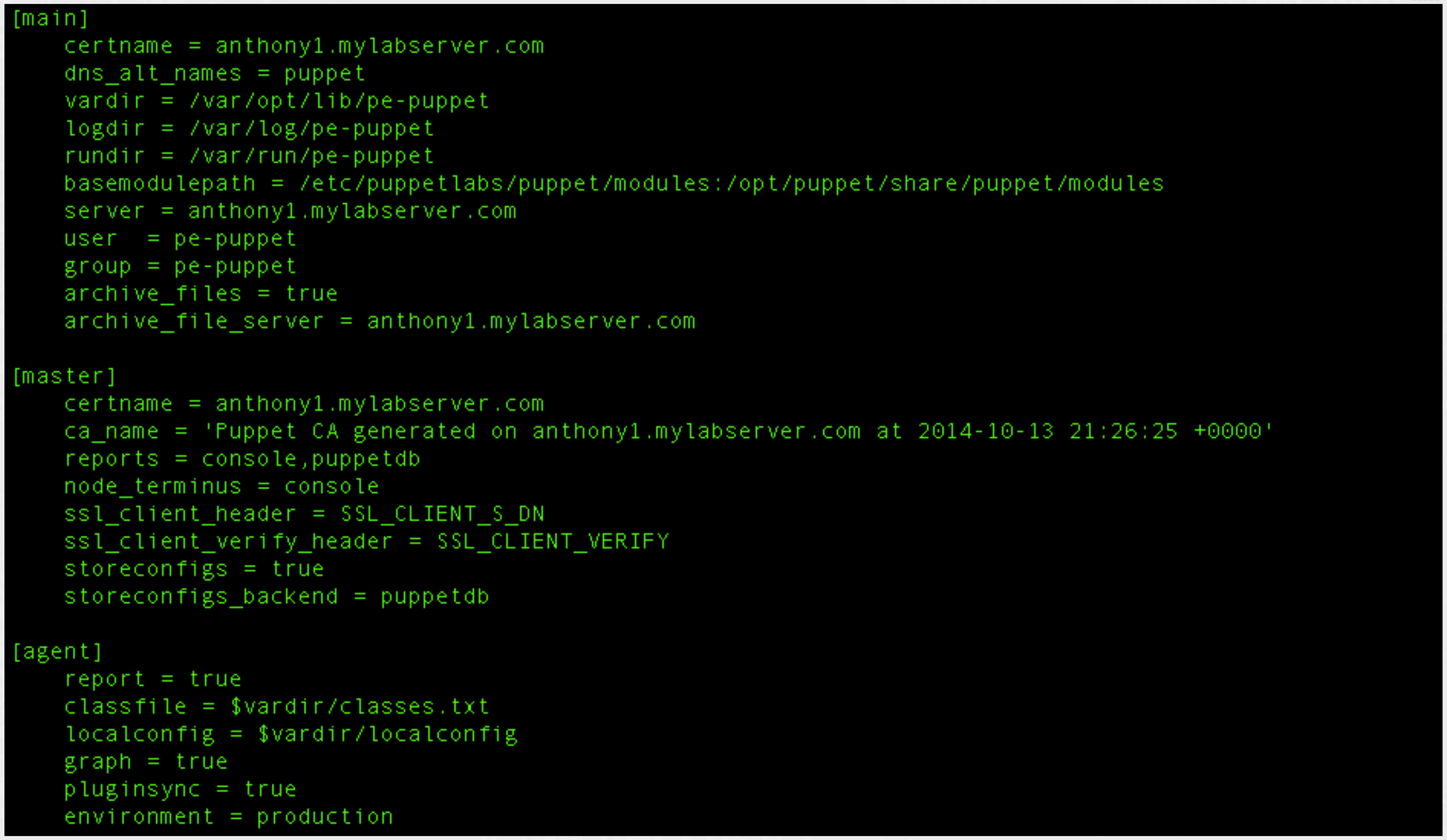
Linux Academy

Learning Puppet

Puppet.conf

Linux Academy

Learning Puppet



```
[main]
    certname = anthony1.mylabserver.com
    dns_alt_names = puppet
    vardir = /var/opt/lib/pe-puppet
    logdir = /var/log/pe-puppet
    rundir = /var/run/pe-puppet
    basemodulepath = /etc/puppetlabs/puppet/modules:/opt/puppet/share/puppet/modules
    server = anthony1.mylabserver.com
    user  = pe-puppet
    group = pe-puppet
    archive_files = true
    archive_file_server = anthony1.mylabserver.com

[master]
    certname = anthony1.mylabserver.com
    ca_name = 'Puppet CA generated on anthony1.mylabserver.com at 2014-10-13 21:26:25 +0000'
    reports = console,puppetdb
    node_terminus = console
    ssl_client_header = SSL_CLIENT_S_DN
    ssl_client_verify_header = SSL_CLIENT_VERIFY
    storeconfigs = true
    storeconfigs_backend = puppetdb

[agent]
    report = true
    classfile = $vardir/classes.txt
    localconfig = $vardir/localconfig
    graph = true
    pluginsync = true
    environment = production
```

Learning Puppet

Puppet.conf

- Puppet Enterprise: /etc/puppetlabs/puppet/puppet.conf
- Puppet OpenSource: /etc/puppet/puppet.conf

Config Sections:

- [main] – Global setting used by all services/servers and is over ridden by other sections
- [master] – Use by Puppet master and puppet cert command
- [agent] – Used by the Puppet agent service (even if the agent runs on the master)
- [user] – Used most commonly by the Puppet apply command

Note: Settings are loaded at service start time, to apply changes made to puppet.conf a restart to the pe-puppet service is required.

Puppet.conf setting format

```
[main]
    vardir = /var/opt/lib/pe-puppet
    logdir = /var/log/pe-puppet
    rundir = /var/run/pe-puppet
    basemodulepath = /etc/puppetlabs/puppet/modules:/opt/puppet/share/puppet/modules
    server = anthony1.mylabserver.com
    user  = pe-puppet
    group = pe-puppet
```

Variable Interpolating in puppet.conf: vardir can be used in the settings file as $vardir, same for any setting.

Notice name of setting followed by equal sign and then the value

Learning Puppet

Fundamental settings, agent:

Basic Settings:

- ca_server – Server used for certificate authority requests
- Certname – Certificate name to use when communicating with the server
- Server – Hostname of the Puppet master
- Environment – Defaults to production

Run behavior:

- noop – Agent will not do any work only simulate changes and report to the master
- priority – Sets the nice command so catalog enforcement does not use up CPU
- report – Determines if reporting is enabled (defaults to true)
- usecacheonfailure – Fall back to last known working catalog if convergence fails

Learning Puppet

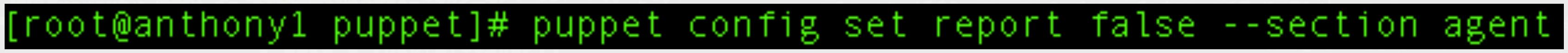
Fundamental settings, agent:

Service Behavior:

- runinterval – How often puppet agent daemon runs
- waitforcert - Keep trying to run puppet agent if the certificate is not initially available

Configuring Settings Via The Command Line

puppet config set <SETTING NAME> <VALUE> --section <CONFIG SECTION>



```
[root@anthony1 puppet]# puppet config set report false --section agent
```

Linux Academy

Learning Puppet

Resource Abstraction Layer

Learning Puppet

Resource Abstraction Layer

A system configuration is a collection of resources that make up the state of your system

- Users
- Cronjobs
- Packages installed
- Services
- Etc.

Primary issue: A command to start a service might be different depending SysVinit, Systemd, init, windows, etc.

Solution: Ex: Declare the state of a service rather than stating how to run the service

Resource Abstraction Layer

Resource Type: In Puppet every resource is an instance of a resource type

```
user { ‘username’:
	ensure => present,
	uid = ‘102’,
	gid => ‘wheel’,
	shell => ‘/bin/bash’
	home => ‘/home/username’,
	managehome => true,
}
```

Note: {commas,ensure,resource type, title}
Note: Declares what the resource is but does not say how enforce it

Resource Abstraction Layer

The resource abstraction layer is made up of two types, first describing/declaring the state of a resource then second using providers to enforce the desired state. By having an abstraction layer you are able to define resources in a way that makes it not dependent upon the OS.

Providers are executed by the RAL and are what provides instructions on how a resource type is enforced, i.e provide platform specific instructions for resource type enforcement such as adding users to Windows vs. adding users to Linux.

Example:
- Redhat/CentOS use yum and RPM
- Debian/Ubuntu use apt-get and DPKG

Linux Academy

Learning Puppet

Resources

- When building modules we are using the puppet DSL to declare the desired state of resources on a node
- Fundamentally all we are doing with Puppet is managing resources on a large and automated scale while caring “as little as possible” about the platform/distribution
- In Puppet we are using resource types to define instances of a resource on a node

Learning Puppet

- Commands to assist:
  - puppet resource: View resources already installed on a node (node level)
    - puppet resource [type]
    - puppet resource [type] [name]
    - puppet resource user
    - puppet resource user root
  - puppet describe: Provide information about resource types within puppet
    - Puppet describe –l (list all resource types available)
    - Puppet describe –s [type]
    - Puppet describe [type]

All resource types are also available via puppetlabs.com online docs

Linux Academy

Learning Puppet

Anatomy Of A Catalog Compilation

Learning Puppet

A catalog describes the desired state for each resource on a system

- Catalogs and any logic for determining resources in the DSL are compiles on the Puppet Master
- After catalog compilation the Agent (node) during the convergence downloads the compiled catalog
- After downloading the catalog the agent will enforce the desired state for each resource and described from in the catalog
- Catalogs can be stored in PuppetDB so catalog information can be queried

Learning Puppet

A catalog describes the desired state for each resource on a system

- Retrieve the node object
  - Node object provides *factual* information about a node
  - Set scope level variables
- Evaluate the Main Manifest
  - Match any matching node definitions
- Load and Evaluate Classes from Modules
  - Modules are loaded from the *modulepath* from puppet.conf
- Once a class is loaded puppet code is evaluated and any resources are added to the catalog

Linux Academy

Learning Puppet

DSL Overview

Common Resource Types: file

```
file { '/etc/ssh/sshd_config':
        ensure => file,
        owner => root,
        group  => root,
        mode  => '0644',
}
```

ensure:

- file – make sure it's a normal file
- directory – makes sure it is a directory (enables recursive)
- link – ensures file is a symlink (requires target attribute)
- absent – deletes file if it exists

Common Resource Types: file attributes

Demo:

- source
- content
- target

Common Resource Types: package

```
package { ‘tree’:
        ensure => present
}
```

Installing packages with an array (multiple packages at one time)

```
package { [‘tree’,’bind-utils’]:
        ensure => present,
}
```

Learning Puppet

Common Resource Types: service

```
service { ‘sshd’:
        ensure => running,
        enable => true,
}
```

Ensure: stopped/running
Enable: determines if a service should be enabled to start at boot time. Values: true/false

More attributes can be found with puppet describe service and in the puppetlabs docs

Case Statements

Note: $osfamily is a "fact" variable we are able to use as part of our code within any special setup

```
case $osfamily {
        'RedHat': {
                $ssh_name = 'sshd'
        }
        'Debian': {
                $ssh_name = 'ssh'
        }
        'default': {
                Warning('OS family does not match ')
        }

service { 'resource-name':
        name   =>  $ssh_name
        ensure => running,
        enable => true,
}
```

Tip: use resource name when the name is different so you can ref it later it code

If statements

```
if condition {
        code
} elsif condition {
        code
} else {
        code
}
```

Additional Conditional Statements

```
$apache = true

If $apache {
        file { '/etc/motd': ensure=> present, content => 'Apache web server', }
} else {
        file { '/etc/motd': ensure=>present, content => 'Unassigned server', }
}
```

Negative if: unless

```
Unless $memorytotal > 1024 {
        $maxclient = 300
}
```

Execute only if the given statement is false cannot include elsif or else statements

Common Resource Types: dependencies and relationships

Key concepts:

- Puppet does not enforce resources from top down, instead based on dependency relationships
- Code is executed from top down
- Always define the dependency relationships needed and never to define ones that you don't

- Resource metaparameters
  - require – required a referenced resource to be applied first (note: resource naming)
  - before – request to be applied before a referenced resource
  - subscribe – listen for Puppet changes to the referenced resource
  - notify – send a notification when Puppet changes the containing resource

Note: Metaparameters are attributes that work with any resource type

Learning Puppet

require

```
package { ‘ssh’:
        name => ‘openssh’,
        ensure => present,
}

service { ‘sshd’:
        ensure => running,
        enable => true,
        require => Package[‘ssh’],
}
```

Note: the uppercase P and what it is doing before

```
package { ‘ssh-package’:
        ensure => present,
        before => Service[‘sshd’]
}

service { ‘sshd’:
        ensure => running,
        enable => true,
}
```

Note: the uppercase S and what it is doing

subscribe

```
file { '/etc/ssh/sshd_config':
        ensure => present,
        source => 'puppet:///modules/ssh/ssh_config,
}

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

Note: Uppercase F, resource name and subscribe watches for changes so will only restart the sshd service if PUPPET changes the ssh_config file. Implies require

Learning Puppet

notify

```
file { '/etc/ssh/sshd_config':
        ensure => present,
        source => 'puppet:///modules/ssh/ssh_config,
        notify => Service['sshd'],
}

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

Note: Uppercase S, resource name and notify sends a message invoking service to restart. Implies before

Learning Puppet

Common Resource Types: metaparameters
Metaparameters are part of the puppet framework and works with any resource type

schedule – creates a window of time for the resource to be managed
alias – creates an alias for the resource name
audit – will check if an attribute for the resource has been modified
noop – tells the resource not to execute
loglevel – debug, info, notice, warning, err, alert, emerg, crit, verbose
tag – sets a specific tag for a given resource type (more advanced usage)

Note:Puppet has the ability to apply resource types with specific tags during a catalog run. I.E tagging a data center

Linux Academy

Learning Puppet

Conditional Expressions

Two ways of looking at conditional expressions

•Conditionals that alter the logic

- unless statements
- case statements
- if statements

•Conditionals that return a value

- selectors

Selectors should be used when setting a “plain value”

- Variable assignments
- Resource attributes
- Function arguments
- Resource titles
- A value in another selector
- Expressions

Learning Puppet

Selectors cannot be used in

- A case in another selector
- A case in a case statement

Learning Puppet

•Selectors return a value and do not evaluate a block of code

•This makes selectors ideal for setting variables or setting attributes in line

Example: setting a variable

```
$package = $osfamily ? {
	'redhat' => 'http',
	'debian' => 'apache2',
}

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

Learning Puppet

•Selectors return a value and do not evaluate a block of code

• Selectors should be used where a plain value is expected

•This makes selectors ideal for setting variables or setting attributes in line

Example: setting an attribute (*note you can use selectors too inside of a resource definition. You cannot use case or if statements in resource definitions)*

```
package { ‘resource-title’ :
        name   => $::osfamily ? {
                ‘redhat’ => ‘httpd’,
                ‘debian’ => ‘apache2’,
                default   => ‘httpd’,
          },
        ensure => installed,
}
```

Learning Puppet

Important:

•If a condition is not met inside of a selector and the default statement is not provided then compiling will fail

•Always ensure you use default match for all conditional statements

•Consider using case or if statements if you need to set a block of code or more than one variable

•*Selectors are only good for setting a variable or attribute*

•Case statements do not require a default match and, if a condition is not met, compiling WILL NOT fail but it is best practice to have a default to avoid other issues

Linux Academy

Learning Puppet

Variables and Parameterized Classes

Variables:

- Variables are constants and cannot be reassigned a value once assigned *within the given scope*
  - Variables can be reassigned in a different scope level
- Syntax:
  - Variables start with a $
  - Single quoted strings interpret variable names literally
    - $var = “hello”
    - $variable = ‘$var world’ will display $var world $variable = “$var world” is hello world
  - Double quoted evaluates the value of the variable
  - Append a variable by using the + symbol
    - $var = [‘item1’,’item2’]
    - $var += [‘item3’]
    - $var now equals [‘item1’,’item2’,’item3’]

Learning Puppet

Scope:

Top Scope – Variables defined within the site.pp node definition file

- Facts are top scope variables
- Best practice to access a fact by empty string before the scope operator
  - ::osfamily

Node Scope – Variables defined within the specific node definition

Class Scope – Variables defined within the class

Parameterized Classes

Goal:

- To reduce calculating parameters from within the class doing the work
- The parameter class will handle conditional statements related to distributions

Lab: Parameterize the base class

Linux Academy

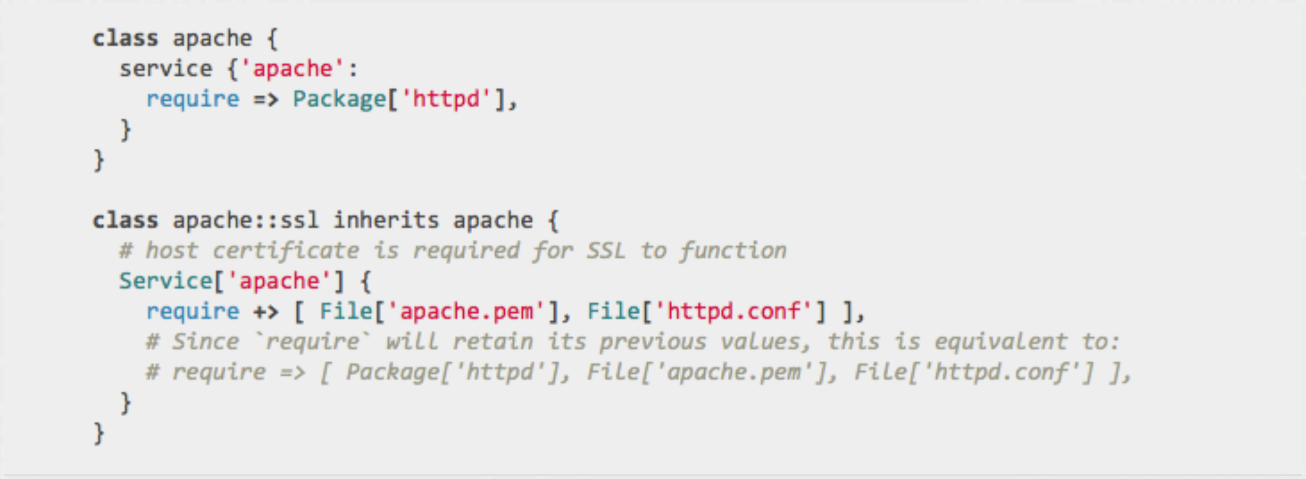
Learning Puppet

Inheritance

Inheritance

- Allows you to extend an existing class and use the local scope variables
- Does not support using parameterized classes
- Inheriting classes from other modules decreases modularity
- Inheriting classes increases compile time of the catalog
- Inheriting classes becomes an issue of readability with local scope variable
- Inheriting classes should ONLY be used to reduce repetition in code
  - Inherit a params class to use the parameters as default values
  - Override resource attributes

Inheritance



```
class apache {
  service {'apache':
    require => Package['httpd'],
  }
}

class apache::ssl inherits apache {
  # host certificate is required for SSL to function
  Service['apache'] {
    require +> [ File['apache.pem'], File['httpd.conf'] ],
    # Since `require` will retain its previous values, this is equivalent to:
    # require => [ Package['httpd'], File['apache.pem'], File['httpd.conf'] ],
  }
}
```

Image source: https://docs.puppetlabs.com/puppet/latest/reference/lang_classes.html#inheritance

Inheritance

Demo: Inheriting a params class to set a classes default values

Linux Academy

Learning Puppet

Inheritance

rvalue functions

- rvalue functions are functions that return a value to be used
- rvalue functions are built-in Puppet functions
- Custom functions can be created but will not be covered in this course
- Common built in rvalue functions:
    - defined
    - file
    - generate
    - regsubstr
    - Sha1
    - alert
    - More functions can be found https://docs.puppetlabs.com/references/latest/function.htm

rvalue functions

Defined – returns true if a class or resource both native and defined types

```
If defined(Service['httpd']) {
    notify { 'the service resource type has been defined': }
}
```

file – returns contents of a file from the server
generate – returns the result of a run shell command
regsubst – $var = regsubstr($string, 'regex to evaluate here')
sha1 – returns a SHA1 hash value from a string

rvalue functions

template – load an ERB template from the module, evaluate the template, and return the resulting contents as a string to the resource type

```
file { '/etc/motd':
        ensure => file,
        content => template('modulename/motd.erb')
}
```

Location of the motd.erb file would actually be modulename/templates/motd.erb
But the 'templates' is not required in the template functions

Linux Academy

Learning Puppet

Variable/Class/Module Naming Practices

Variable Recap:

Interpolation: When a variable is resolved within a double quoted string

```
$variable = "This is my string for ${var}"
```

+= to add an item to an array

Puppet always finds the "closest" scope to the code being executed

Scope is very important

Can access variables outside of scope by defining their scope

```
$apache::params::varname
```

Learning Puppet

Variable Names:

- Variable names can begin with AND include:
    - Uppercase and Lowercase letters
    - Numbers
    - Underscores
- DO NOT USE DASHES
- $string variable name is reserved and cannot be used

Good:

- $_server = “server”
- $Server = “Server”
- $3server = “server

Bad: $server-name

Learning Puppet

Class Names:

Each starting name space must begin with a ***<u>lowercase letter</u>***
Can include **lowercase letters, numbers and underscores**

Bad:

```
class Apache::Params {}
class apache::Params {}
class 1apache::params {}
```

Good:

```
class apache::params {}
class apache_mod::params{}
```

Reserved Class Names:

- main
- settings

✓These names cannot be the name of modules
✓These names can be included in a module name

Good:	main_apache
	apache_main
Bad:	main
	settings

Module names:

•Must begin with a lowercase letter

•Can include numbers, lowercase letters, and underscores

Linux Academy

Learning Puppet

Roles/Profiles Overview

Learning Puppet

Up until this point our focus has been learning the Puppet DSL and how to configure resources

Design considerations:

- Classify nodes by roles and roles call profiles
- Configure environments, rather than re-writing code (dev urls for dev environments etc)
- Design classes as reusable
- Modules should only manage their own resources
    - Apache module should manage only Apache resources.

Learning Puppet

Profiles: A stack of technology configuration and class declarations which also define class parameters if required and contain minimal logic

```
class profiles::apache ($parameters = $apache::params::param) {
  include apache
  include php
  include phpmyadmin
  #include any other declarations and parameter passing as required
}
```

Learning Puppet

Roles: A role is based off of a business function. I.E webserver/databaseserver/etc

```
class roles::webserver {
        include profiles
        #should contain no logic
        #cleans up our node definitions
}
```

Linux Academy

Learning Puppet

Hiera

Learning Puppet

Hiera is a key:value lookup tool that is built to provide node-specific data. The key component is not having to repeat yourself or change code/variables/configuration within a module. Instead we use hiera to set specific configuration information that applies to the module/class on a per node basis.

Advantages:

- Easier to configure data on a per-node basis. Example: Dev API URLs
- Easier to re-use Puppet forge (public) modules without having to edit classes
- Keep node configuration in one place and make it easily to override when necessary

Learning Puppet

- Hiera is installed and enabled by default on Puppet Enterprise 3.0+
- Hiera configuration file is located in /etc/puppetlabs/puppet/hiera.yaml
- OpenSource Hiera config file: /etc/puppet/hiera.yaml
- Available command line tool for testing key:value lookups
- Automatic parameter lookup

Learning Puppet

- Automatic parameter lookup

```
class hierademo ($parameter = “default”) {

}
```

Process of automatic parameter lookup:

1. Look for parameters passed using the class {} declaration
2. If no pass parameter it will look in hiera data source for the parameter hierademo::parameter
3. If not found in hiera data source it will use the default set “default”

Hiera.yaml configuration file

```
---
:backends:
  - yaml
:hierarchy:
  - defaults
  - "%{clientcert}"
  - "%{environment}”
  - ”%{::osfamily}”

:yaml:
# datadir is empty here, so hiera uses its defaults:
# - /var/lib/hiera on *nix
# - %CommonAppData%\PuppetLabs\hiera\var on Windows
# When specifying a datadir, make sure the directory exists.
  :datadir:
```

:hierarchy:

- How hiera looks up data based off hierarchy depends on the lookup function being used
- Assuming a priority lookup (hiera()) function
  - Hiera will attempt to match the data file from top first then to bottom
  - Given the following hierarchy
  - :hierarchy:
  - - %{::fqdn}
  - - %{::osfamily}

  - Hiera will attempt to match data in the fqdn data source before looking for osfamily.
  - Note: hiera can use facts from a node to determine lookup data source

Learning Puppet

:hierarchy:

If a data source is set to %{::osfamily} then a .yaml file needs to be located in the data source directory with the same name as the osfamily

Example: /etc/puppetlabs/puppet/hieradata/redhat.yaml

Note: above is a custom data source location

Available hiera lookup functions:

- hiera: Priority lookup, it will grab the first matching value from the first matching data source in the hierarchy and return just that string, in array form if there is nested data.
- hiera_array: Will return all possible matches from all data sources. It will return the returned data from data sources higher in the hierarchy first. If a hash is found then the lookup will fail.
- hiera_hash: Grabs a matching value from every level in the hierarchy and retrieves all of the hash values for a given key and then merges the hashes into a single hash.

Linux Academy

Learning Puppet

Site.pp And Node Definition Matching

Node Definition Lookup

If a node is defined within the site.pp manifest then a class MUST be declared

Name lookup process:

1. Puppet will look for exact definition matches within site.pp and the certificate name
2. Puppet will then evaluate any regular expressions if there is no exact match
3. If the nodes name looks like an FQDN then puppet will do the following:
    1. Chop off the final group and start and step 1 (exact match), then regular expression match
    2. If there is no match puppet will remove group two and start again at step 1
    3. If no match Puppet will resort to the default definition if it is defined

Learning Puppet

Node Definition Lookup Example: webserver01.mylabserver.com

Note: assuming no exact matches or regular expression matches

1. Attempt to match webserver01.mylabserver.com, if no match then (either exact or regular expression)
2. webserver01.mylabserver, if no match then *(either exact or regular expression)*
3. webserver01, if no match *(either exact or regular expression)*
4. Match Default

Learning Puppet

Note: if a node matches multiple node definitions due to regular expressions, puppet will use ONE of them with no guarantee as to which one it will use.

Linux Academy

Learning Puppet

External Node Classifiers (ENCs)

External Node Classifiers (ENCs)

- An ENC is an arbitrary script or application that can tell Puppet which classes a node should have. It either replaces or works with the node definitions in the site manifest file (site.pp)

- ENC can be enabled in Puppet.conf by setting the two following settings
  - node_terminus=exec (from plain to exec or console)
  - external_nodes –This is the path to the executable of the ENC

- *https://docs.puppetlabs.com/guides/external_nodes.html*

External Node Classifiers (ENCs) & Site.pp Merging

A Puppet catalog is made up of:

- ENCs work with the site.pp by merging the node objects
  - All classes specified in the node object as defined in site.pp OR node_terminus executable
  - Any classes or resources which are in the site manifest but outside any node definitions
- Any classes or resources in the most specific node definition in site.pp that matches the current node (if site.pp contains any node definitions)

Linux Academy

Learning Puppet

Troubleshooting

Troubleshooting Installation Issues

- DNS
- Ports open (3000, 8140, 443)
- Configuration issues
- Answers file
- Cannot run master 3.x with agents other than 3.x
- dns_alt_names – all potential host names for the puppet master need to be configured here during install
- Installing puppet master and console on separate servers
  - Ensure puppet master is installed before the console

Troubleshooting Installation Issues

Recover from a failed install

- Fix the configuration issue that caused the failure
- Run puppet-enterprise-uninstaller
- Run the installer again

Rake API For Common Tasks

Reports:prune (rake task) – to clean out old old reports

Following task is usually automatically installed in a cron during installation

- Example: rake \ -f /opt/puppet/share/puppet-dashboard/Rakefile / RAILS_ENV=production \ reports:prune upto=1 unit=mon

Optimizing Dashboard Database

- Puppet-dashboard rake RAILS_ENV=production db:raw:optimize
  - Should be done monthly
  - Needs to have a manual cronjob added

Console And Database Troubleshooting

Restart pe-puppet-dashboard-workers service

- Clears Pending tasks/dead workers

Autovacuum=on (helps maintain and clean out old reports/logs from postgreSQL)

Puppet Agent Run Errors

Erorr: Could not request certificate: The certificate retrieved from the master does not match the agent's private key

- Remove the .pem key for the node from the nodes ssl location /etc/puppetlabs/puppet/ssl/certs/
- Clean the certificate off the master using puppet cert clean <certname>
- Run puppet agent again, then sign the certificate

Warning: Unable to fetch my node definition, but the agent run will continue

- Puppet master is unreachable
    - Could be the server itself is not reachable (dns, networking, etc)
    - Could also be the service itself is stopped and needs to be started

Linux Academy

Learning Puppet

Deactivating A Puppet Enterprise Agent Node

Linux Academy

Learning Puppet

1. Stop the agent service on the node
2. Deactivate the node on the master by using puppet node deactivate <cert name>
3. Revoke the certificate by using puppet cert clean <certname>
4. Restart the Puppet master service (pe-puppetserver restart) after cleaning cert
5. Delete the node on the console
6. Stop the mcollective service on the node
7. Remove mcollective certificates from the node rm –f /etc/pupeptlabs/mcollective/ssl/client

Linux Academy

Learning Puppet

Reports

Learning Puppet

Information found on reports:

- **Total**: Total number of resources being managed
- **Skipped**: How many resources were skipped (either due to tags or schedule metaparameter)
- **Scheduled**: How many resources met the scheduling restriction, if one is present
- **Out of Sync**: How many resources were out of sync (not in the desired configuration state)
- **Applied**: How many resources were attempted to be put into the desired configuration state
- **Failed**: How many resources were not successfully fixed (put into the desired configuration state)
- **Restarted:** How many resources were restarted
- **Failed restarts**: how many resources could not be restarted
- **Total time for configuration run (puppet agent execution)**
- **How long it took to retrieve the configuration** (compiled catalog) from the puppet master

Learning Puppet

- Regular puppet errors for both the agent and master can be found in the systems syslog
- For reporting to be enabled on the node level the node puppet.conf needs to have the “report” setting set to true
- The report setting on the Puppet master specifies how reports are processed
- Puppet.conf example: reports = tagmail, http

- Builtin report processors
  - http – send reports to https/http
  - tagmail – send specific reports to specific email addresses
  - rrdgraph – Graph all data in RRD library
  - log – Send logs to local syslog
  - store – store reports in yaml form in the location specified in the reportdir setting
- Plugin report processors
  - IRC
  - Twitter
  - Jabber
  - Hipchat
  - Growl
  - Campfire
  - PagerDuty
  - Etc.

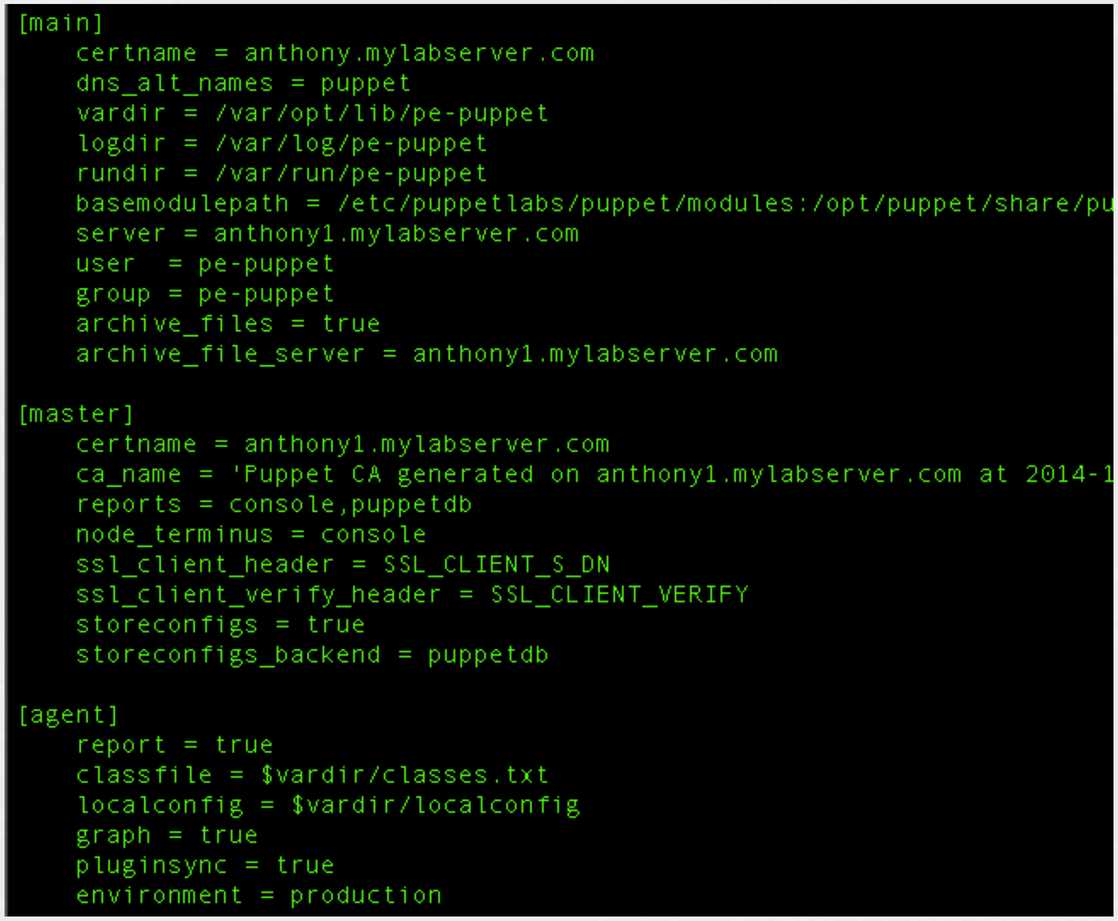
Learning Puppet

- Regular puppet errors for both the agent and master can be found in the systems syslog
- For reporting to be enabled on the node level the node puppet.conf needs to have the “reports” setting set to true
- The report setting on the Puppet master specifies how reports are processed
- Puppet.conf example: reports = tagmail, http

- Builtin report processors
  - http – send reports to https/http
  - tagmail – send specific reports to specific email addresses
  - rrdgraph – Graph all data in RRD library
  - log – Send logs to local syslog
  - store – store reports in yaml form in the location specified in the reportdir setting
- Plugin report processors
  - IRC
  - Twitter
  - Jabber
  - Hipchat
  - Grow
  - Campfire
  - PagerDuty
  - Etc.

Linux Academy

Learning Puppet



```
[main]
    certname = anthony.mylabserver.com
    dns_alt_names = puppet
    vardir = /var/opt/lib/pe-puppet
    logdir = /var/log/pe-puppet
    rundir = /var/run/pe-puppet
    basemodulepath = /etc/puppetlabs/puppet/modules:/opt/puppet/share/pu
    server = anthony1.mylabserver.com
    user  = pe-puppet
    group = pe-puppet
    archive_files = true
    archive_file_server = anthony1.mylabserver.com

[master]
    certname = anthony1.mylabserver.com
    ca_name = 'Puppet CA generated on anthony1.mylabserver.com at 2014-1
    reports = console,puppetdb
    node_terminus = console
    ssl_client_header = SSL_CLIENT_S_DN
    ssl_client_verify_header = SSL_CLIENT_VERIFY
    storeconfigs = true
    storeconfigs_backend = puppetdb

[agent]
    report = true
    classfile = $vardir/classes.txt
    localconfig = $vardir/localconfig
    graph = true
    pluginsync = true
    environment = production
```

Linux Academy

Learning Puppet

- Regular puppet errors for both the agent and master can be found in the systems syslog
- For reporting to be enabled on the node level the node puppet.conf needs to have the "reports" setting set to true
- The report setting on the Puppet master specifies how reports are processed
- Puppet.conf example: reports = tagmail, http

- Builtin report processors
    - http – send reports to https/http
    - tagmail – send specific reports to specific email addresses
    - rrdgraph – Graph all data in RRD library
    - log – Send logs to local syslog
    - store – store reports in yaml form in the location specified in the reportdir setting
- Plugin report processors
    - IRC
    - Twitter
    - Jabber
    - Hipchat
    - Grow
    - Campfire
    - PagerDuty
    - Etc.

Linux Academy

Learning Puppet

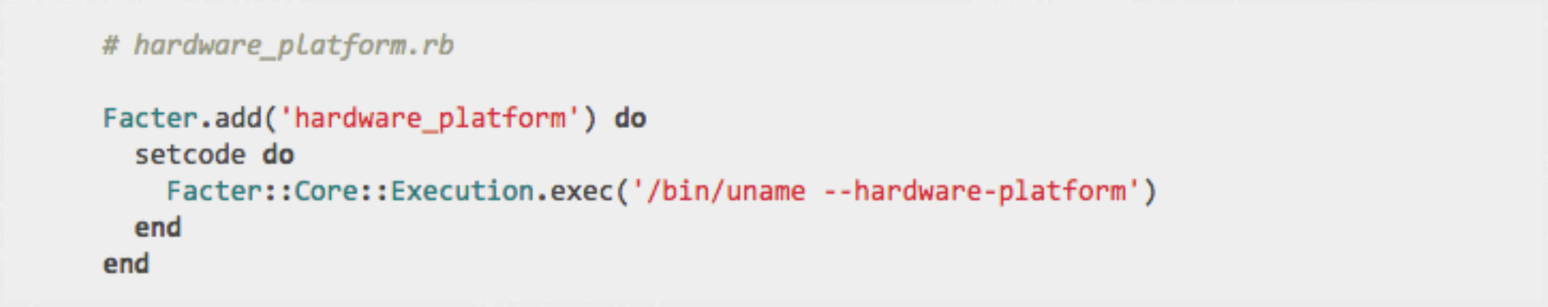
Custom Facts & External Facts

Custom Facts

- Written on the Puppet Master
- Are ruby blocks of code that execute to create a “custom fact”
- Usually shell commands are issued as part of the fact to return information
- Executed on the Puppet nodes with the External Facts Plugin Module
    - $LOAD_PATH or the ruby library load path
    - Environment variable FACTERLIB
    - Facts distributed using pluginsync
        - Enabled in the [main] section of puppet.conf by setting pluginsync=true

Custom Facts

Command facter –p will load all the facts that have been distributed via pluginsync



```ruby
# hardware_platform.rb

Facter.add('hardware_platform') do
  setcode do
    Facter::Core::Execution.exec('/bin/uname --hardware-platform')
  end
end
```

Source: https://docs.puppetlabs.com/facter/2.3/custom_facts.html#the-concept

External Facts

External Facts enable the ability to use executable scripts or programs written with languages like c, perl, bash, etc. Essentially the scripts are executed as part of the node during the Puppet agent run which populates the fact data. This is different from a custom fact because custom facts are built with ruby on the Puppet master and generally execute just a shell command.

- External facts can be referenced by ruby custom facts but not the other way around
- Best way for external facts to be used is through pluginsync

Pluginsync

- External and custom facts should be built and located in the lib/facter directory of a Puppet module
- The facts are loaded on the node as part of the puppet file server
- The pluginsycn setting in the [main] section needs to then be set to true

Linux Academy

Learning Puppet

Puppet Module

Puppet Module

Puppet module is a command used to help manage building and downloading modules from the Puppet forge. Puppet module allows you to make your modules available for third party download or for you to download modules already built and available on http://forge.puppetlabs.com

Linux Academy

Learning Puppet

Puppet Module Parameters and Arguments

- list – List installed modules on the system
- uinstall – Uninstall a Puppet module
- install
    - --version – specifies which version of the module you would like to install
    - --modulepath – specific which module path to install too
    - --environment – select the environment to install too
    - --force – forcibly install a module
    - --debug – see additional information about what is happening during the install process
    - --ignore – skip install any dependencies required by this module

- upgrade – Upgrade the Puppet module
- search – Search the Puppet forge for a module
- build – build a module release package
- generate – generate a boilerplate for a new module (skeleton directory with directories like manifest)
- changes – show what files were changed for an installed puppet module

Preparing A Module For The Forge

- Make sure the proper directory structure exists
- Populate metadata.json (required) auto created when puppet module generate is used
    - Metadata.json includes dependencies and basic module information like name
    - Note “modulefile” is the old requirement and has been updated to metadata.json
- Remove all symlinks
- Puppet module build <module dir> will build a tar for upload to Puppet Forge

Metadata.json dependencies

```
"dependencies": [
{ “name”: “ownership/module”, “version_requirement”: “>=0.1.0”}
{“name”: “ownership/module2”, “version_requirement”: “>=0.1.0 < 0.5.0”}
]
```

Examples:
1.2.x would match any number in x spot i.e 1.2.3 this is known as “semantic” version
Semantic versions i.e 1.3.x cannot be used with >= < operators for ranges and minimals

Linux Academy

Learning Puppet

Puppet Professional Exam

Puppet Professional Exam

- Register at http://puppetlabs.com/services/certification/puppet-professional
- Currently $200 each attempt
- If first attempt fail, must wait 14 days
- If second attempt fail, must wait 90 days
- If third attempt fail, must contact puppet to request a fourth attempt “within” 6 months of first attempt
- http://puppetlabs.com/services/certification/faq

Puppet Professional Exam

- Linux Academy Practice exam “mimics” a real exam but does not contain the same questions or exact same format do to NDA agreements with Puppet.
- If you learn the content, memorize the study guides, and practice you will know the concepts required to pass the exam.
- The more you use Puppet and are familiar with writing code, using the console, and troubleshooting the easier the exam is.
- Using Linux Academy content it is still possible to pass the exam without hands-on job experience but the amount of study and practice required is substantially more.

Linux Academy

Learning Puppet

How Puppet Enforces Execution In A Catalog

Resource Ordering Behavior

What happens if there are multiple file resource types that send a “notify” metaparameter notification to a service?

```
file { ‘file1’:
        notify => Service[‘sshd’],
}
file { ‘file2’:
        notify => Service[‘sshd’],
}
```

Parser order

Puppet will evaluate the syntax within a class from the top down. However, resource declarations are not dependent upon this parser order and there is no guarantee as to the order a resource is executed/enforced unless there are specific relationship attributes used.

```
class myclass {
        package { $package: ensure => present, }
        $package = “httpd”
}
```

Will this error due to parse order? Or will the variable be evaluated and executed within the resource type? Or does the variable need to be assigned before it is used?

Dependency Cycle

If one or more resources require each other creating a relationship dependency loop then puppet compiling will fail.