
Linux Academy
System Administration
Using Puppet Certification
Introduction to Puppet

Introduction to Puppet

- What are common issues managing traditional infrastructure?
  - Provisioning of new nodes (servers)
  - Configuring nodes is time consuming
  - Managing configuration drift
  - Using custom scripts is not a solution
  - So many servers so little system administrators

What is Puppet?

- Software configuration management tool
- Define a configuration once and apply it to thousands of nodes
- Remediate configuration drift
- Get details about your hardware and software configurations
- Orchestrate change to thousands of nodes

What is Puppet?

- The Puppet's language is declarative
- Puppet code is written to express desired end state of the node
- Management of node resource are abstracted from the operating system
- Code is written inside of classes, classes are assigned to nodes
- Puppet was founded by Luke Kanies in 2005
- Written in Ruby and Clojure

More About Puppet

- The Puppet Forge (forge.puppet.com)
- PXE boot provisioning using Razor
- Testing Puppet code with RSpec
- Serverspec testing with Beaker
- Extend Puppet modules with Ruby

System Administration Using Puppet Certification

About This Course

About This Course

- Course Author: Travis N. Thomsen, Puppet Certified
- The course will focus on:
  - Puppet Enterprise 2016.2
  - Core concepts as they relate to Puppet Enterprise
  - Installing and configuring the Puppet master and agent nodes
  - Learn the Puppet DSL by utilizing LinuxAcademy.com lab servers
  - Learn how to create component Puppet modules
    - SSH
    - MOTD
    - Apache

About This Course

- The course will focus on:
  - Using roles and profiles for reusability
  - How to manage data with Hiera
  - Deploying Puppet modules with Code Manager
  - Node classification
  - Managing your Puppet infrastructure
  - The fundamentals required for the System Administration Using Puppet certification (2016.2)


Linux Academy
System Administration
Using Puppet Certification
Puppet Enterprise Stack and Core Concepts

Agent Components (On All Nodes)

- **Puppet agent** is the software used to communicate with the Puppet master to execute a Puppet run.
- **Puppet** is the core of our configuration management platform. It consists of a special programming language for describing desired system states, an agent that can enforce desired states, and several other tools and services.
- **Facter** is a system profiling tool. Puppet agent uses it to send important system info to Puppet Server, which can access that info when compiling that node's catalog.

Agent Components (On All Nodes)

- **Hiera** is a hierarchical data lookup tool. You can use it to configure your Puppet classes.
- **MCollective** (Marionette Collective) is a framework for building server orchestration or parallel job-execution systems.

Server Components

- **Puppet Server** is the JVM application that provides Puppet's core HTTPS services. Whenever Puppet agent checks in to request a configuration catalog for a node, it contacts Puppet Server.
- **PuppetDB** collects the data Puppet generates and offers a powerful query API for analyzing that data. It's the foundation of the PE console, and you can also use the API to build your own applications.
- **R10k** is a code management tool that allows you to manage your environmental configurations (such as production, testing, and development) in a source control repository.

Server Components

- **Razor Server** is a PXE boot provisioning application that deploys bare-metal systems.
- **PostgreSQL** is an open source relational database management system used by PuppetDB.
- **ActiveMQ** is the message broker used by MCollective.
- **Nginx** is the web server used by the Puppet Enterprise Console.
- **Console** Puppet Enterprise's web interface.

Puppet Enterprise Client Tools

- **Puppet orchestrator** allows you to control the rollout of changes in your infrastructure and provides the interface to the Puppet Application Orchestration service.
- **Puppet access** is a tool used to authenticate yourself to the PE RBAC token-based authentication service so that you can use other capabilities and APIs.
- **Code Manager** provides the interface for the Code Manager and file sync services.
- **PuppetDB CLI** is a tool for working with PuppetDB, such as building queries and handling exports.

Puppet Master vs. Puppet Server vs. Puppet Master Server

- **Puppet master** is a Ruby application that compiles configurations for any number of Puppet agent nodes, using Puppet code and various other data sources.
- **Puppet Server** is an application that runs on the Java Virtual Machine (JVM) and provides the same services as the classic Puppet master application.
- **Puppet Master Server (Puppet Master)** is the server that performs the catalog compile.

What is a Catalog?

When a node checks into the Puppet master server it retrieves a document called a catalog. The catalog describes the desired state of the node being 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.

Why is It Used?

- Separate privileges
- Reduce the agent's resource consumption
- Simulate changes
- Record and query configurations

Puppet Process

- Install Puppet Enterprise master and agents
- Create configurations (modules/classes) that describe a resources end state
- Assign configurations to nodes (declaring a class)
  - Configurations can be assigned in the console
  - Configurations can be assigned in the site.pp file
  - Configurations can be assigned in your Hiera data


Linux Academy
System Administration
Using Puppet Certification
Nodes

Puppet Nodes (Agents)

- Nodes are virtual, physical or non-ephemeral cloud systems
- Runs the Puppet agent

Required ports:

- 8140 for the Puppet agent
- 61613 for MCollective

Supported Operating Systems (Version 2016.5.2)

Operating system	Versions	Architecture
Red Hat Enterprise Linux	4, 5, 6, 7	• x86_64 • i386 for 5, 6 • IBM z System for 6, 7
CentOS	5, 6, 7	• x86_64 • i386 for 5, 6
Oracle Linux	5, 6, 7	• x86_64 • i386 for 5, 6
Scientific Linux	5, 6, 7	• x86_64 • i386 for 5, 6
SUSE Linux Enterprise Server	10 (SP 4 only), 11 (SP 1 and later), 12	• x86_64 • i386 for 10, 11 • IBM z System for 11, 12
Solaris	10 (update 9 or later), 11	• SPARC • i386
Ubuntu	10.04, 12.04, 14.04, 16.04	• x86_64 • i386

Supported Operating Systems (Version 2016.5.2)

Fedora	22, 23, 24	• x86_64 • i386
Debian	Wheezy (7), Jessie (8)	• x86_64 • i386
Microsoft Windows (Server OS)	2008, 2008R2, 2012, 2012R2, 2012R2 core	• x86_64 • i386
Microsoft Windows (Consumer OS)	Vista, 7, 8, 8.1, 10	• x86_64 • i386
OS X	10.9, 10.10, 10.11	x86_64
AIX	5.3, 6.1, 7.1	Power


Linux Academy
System Administration
Using Puppet Certification
Installing Puppet

Installation Puppet Enterprise

- Puppet Master install command: ./puppet-enterprise-installer
- Installation Flags
  - -c – Use a pe.conf file to configure the Puppet server.
  - -D – Displays debugging information.
  - -h – Display help.
  - -q – Run in quite mode; the installation process is not displayed.
  - -V – Display very verbose debugging information.
  - -y – Assumes yes/default and bypass any prompts for user input.

pe.conf

- The pe.conf file is a HOCON-formatted file that declares parameters and values needed to install and configure Puppet Enterprise.
- When the installation completes, you can find an updated pe.conf file in /etc/puppetlabs/enterprise/conf.d.

```
{
  "console_admin_password": "password",
  "puppet_enterprise::puppet_master_host": "<puppet-master-fqdn>",
  "pe_install::puppet_master_dnsaltnames": [
     "puppet”
  ]
}
```

Installation Directories

- Puppet Enterprise configuration files are installed in /etc/puppetlabs/puppet for *nix nodes and <COMMON_APPDATA>\\PuppetLabs for Windows nodes.
- Puppet Enterprise software binaries are installed in /opt/puppetlabs.
- Executable binaries are in /opt/puppetlabs/bin and /opt/puppetlabs/sbin.
- The installer automatically creates symlinks in /usr/local/bin.

Code and Data Directories

- R10k: /etc/puppetlabs/r10k
- Environments: /etc/puppetlabs/code/environments
- modules: Main directory for puppet modules (applies to master only)
- manifests: Contains the main starting point for catalog compilation (applies to master only)
- ssl: Contains each node's certificate infrastructure (all nodes) /etc/puppetlabs/puppet/ssl

Important Ports

- 3000: Used for the web-based installer of the Puppet master.
- 8140: The port on which the Puppet master and agents communicate.
- 61613: Used by MCollective for orchestration requests by Puppet agents.
- 443: The web port used to access the Puppet Enterprise Console.
- 5432: PostgreSQL runs on this port. It is used by PuppetDB in a split stack configuration.
- 8081: The PuppetDB traffic/request port.
- 8142: Used by Orchestration services to accept inbound traffic/responses from the Puppet agents.

Puppet Enterprise Services

On CentOS 7, the Puppet Enterprise services are installed in /usr/lib/systemd/system.

- pe-activemq: The ActiveMQ message server, which passes messages to the MCollective servers on agent nodes. Runs on servers with the Puppet master component.
- pe-console-services: Manages and serves the PE console.
- pe-puppetserver: The Puppet master server, which manages the Puppet master component.
- pe-nginx: Nginx, serves as a reverse-proxy to the PE console.
- mcollective: The MCollective daemon, which listens for messages and invokes actions. Runs on every agent node.
- puppet (on EL and Debian-based platforms): The Puppet agent daemon. Runs on every agent node.

Puppet Enterprise Services

- pe-puppetdb and pe-postgresql: Daemons that manage and serve the database components. Note that pe-postgresql is only created if we install and manage PostgreSQL for you.
- pe-orchestration-services: Runs the Puppet orchestration process.
- pxp-agent: Runs the Puppet agent PXP process.

Puppet Enterprise Logs

- Puppet master logs: /var/log/puppetlabs/puppetserver
- Puppet agent logs: /var/log/messages or /var/log/system.log
- ActiveMQ logs: /var/log/puppetlabs/activemq
- MCollective service logs: /var/log/puppetlabs/
- Console logs: /var/log/puppetlabs
- Installer logs: /var/log/puppetlabs/installer
- Database logs: /var/log/puppetlabs/puppetdb and /var/log/puppetlabs/postgresql
- Orchestration logs: /var/log/puppetlabs


Linux Academy
System Administration
Using Puppet Certification
Puppet.conf

Puppet.conf

- The puppet.conf file is located in /etc/puppetlabs/puppet
- Settings can be overridden at the command line

Config Sections

- **main:** The global section used by all commands and services.
- **master:** Used by the Puppet master service and the “puppet cert” command.
- **agent:** Used by the Puppet agent service.
- **user:** Used by the “puppet apply” command

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

Interpolating Variables

The values of settings are available as variables within puppet.conf, and you can insert them into the values of other settings. To reference a setting as a variable, prefix its name with a dollar sign.

Example:

- $codedir
- $confdir
- $vardir

Sample puppet.conf for a Puppet Maser

```
[main]
certname = puppet.domain.com
server = puppet.domain.com
user = pe-puppet
group = pe-puppet
environment_timeout = 0
app_management = true
module_groups = base+pe_only
environmentpath = /etc/puppetlabs/code/environments
codedir = /etc/puppetlabs/code

[agent]
graph = true

[master]
node_terminus = classifier
storeconfigs = true
storeconfigs_backend = puppetdb
reports = puppetdb
certname = puppet.domain.com
always_cache_features = true
```

Sample puppet.conf for an Agent Node

```
[main]
server = puppet.domain.com
certname = agent1.domain.com
```

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

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>


Linux Academy
System Administration
Using Puppet Certification
Resource Abstraction Layer

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

Resource Abstraction Layer

- Every resource is associated with a resource type, which determines the kind of configuration it manages.
- Declares what the resource looks like not how it is enforced.
- Every resource type has:
  - a title
  - a set of attributes

Resource Type Format:

```
<TYPE> { '<TITLE>':
  <ATTRIBUTE> => <VALUE>,
}
```

Resource Abstraction Layer

Example:

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

Resource Abstraction Layer

A resource declaration is an expression that describes the desired state for a resource and tells Puppet to add it to the catalog. When Puppet applies that catalog to a target system, it manages every resource it contains, ensuring that the actual state matches the desired state.

Providers implement the same resource type on different kinds of systems. They usually do this by calling out to external commands.

Example:

- RedHat/CentOS uses yum and RPM
- Debian/Ubuntu uses apt-get andDPKG
- Ruby uses gems

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.
- https://docs.puppet.com/puppet/latest/type.html

Commands

- 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]


Linux Academy
System Administration
Using Puppet Certification
Catalog Compilation

Catalog Compilation

- A catalog describes the desired state for each resource on the node
  - The catalog is compiled on the master
  - The compiled catalog is shipped to the node during the Puppet run
  - The desired state is enforced on the node by the catalog
  - The catalog is stored in PuppetDB
    - Default on PE install
- Puppet compiles the catalog using sources of configuration info
  - Agent-provided data (Facts)
  - External data (Hiera)
  - Puppet manifests (Puppet code)

Catalog Compilation

- Retrieve the node object
  - Node object provides factual information about a node
  - Set scope-level variables
- Set variables from the node object, facts, and the certificate
  - Variables provided by the node object will now be set as top-scope
  - Node's facts are also set as top-scope variables
  - Variables provided by the Puppet master will also be set
- Evaluate the main manifest (site.pp)
  - Match any matching node definitions
- Load and evaluate classes from modules
  - *The environmentpath* setting in puppet.conf tells Puppet where to find environments.
    - /etc/puppetlabs/code/environments/<ENVIRONMENT>/modules
    - Default environment is production

Catalog Compilation

- Evaluate classes from the node object
  - Variables provided by the node object will be set as top-scope
  - Node's facts are set as top-scope variables
  - Variables provided by the Puppet master will be set


Linux Academy
System Administration
Using Puppet Certification
Certificate Signing Request

Certificate Signing Request

Puppet Server includes a certificate authority (CA) service that accepts certificate signing requests (CSRs) from nodes, serves certificates and a certificate revocation list (CRL) to nodes, and optionally accepts commands to sign or revoke certificates.

- When you install a new PE agent, the agent will automatically submit a certificate signing request (CSR) to the Puppet master.
- Before Puppet agent nodes can retrieve their configuration catalogs, the certificate needs to be signed by the certificate authority (CA).

Certificate Signing Request

Command:

puppet cert
puppet cert list
puppet cert sign <NAME>
puppet cert revoke <NAME>

DNS altnames:

puppet cert sign (<HOSTNAME> or --all) --allow-dns-alt-names <NAME>

Regenerating Certificates

- On the Puppet master:
  - puppet cert clean<NAME>
- Deleting SSL certs on agent:
  - cp -r /etc/puppetlabs/puppet/ssl/ /etc/puppetlabs/puppet/ssl_bak/

Autosigning

- Should only be used when the environment fully trusts any computer able to connect to the Puppet master.
- The CA uses a config file containing a whitelist of certificate names and domain names.
  - $confdir/autosign.conf
  - *.domain.com
- To disable autosigning
  - autosign = false in the [master] in puppet.conf

System Administration Using Puppet Certification

Classes

Puppet Classes

- Classes are named blocks of Puppet code that are used in a modules.
- They are not applied until they are invoked by name.
- They can be added to a node's catalog by declaring it in a manifest or in the ENC.
- They use Resource Types to configure packages, files, services, etc.
- Classes can use parameters to request external data.
  - A default parameter should be supplied.
  - Each parameter can be preceded by an optional data type.
- Classes are singletons.

Puppet Classes

- Class names can consist of one or more namespace segments.
- Each namespace segment must begin with a lowercase letter and can include:
  - Lowercase letters
  - Digits
  - Underscores
- Namespace segments should match the following regular expression:
  - \A[a-z][a-z0-9_]*\Z
  - class_name123
- Multiple namespace segments can be joined together in a class name with the :: (double colon) namespace separator.
  - \A([a-z][a-z0-9_]*)?(::[a-z][a-z0-9_]*)*\Z
  - module_name::class_name

Puppet Classes

Class syntax:

```
class <CLASS_NAME> (
  <DATA_TYPE> <PARAM_NAME>
) {
  … puppet code ...
}
```

Example:

```
class ssh {
  file { “/etc/ssh/ssh_config”:
    ensure => file,
    source => “puppet:///modules/ssh/ssh_config”
  }
}
```

Class Variables

- Variable names begin with a $ (dollar sign) and are case-sensitive.
- Variable names can include:
  - Uppercase and lowercase letters
  - Numbers
  - Underscores (_)
- There are reserved variable names:
  - Data Types
  - Function names


Linux Academy
System Administration
Using Puppet Certification
Modules

Creating a Module

- Modules are self-contained bundles of code and data used to manage a single piece of technology.
- How to generate a module
  - puppet module generate <MAINTAINER>-<MODULE_NAME>
    - The tests directory deprecated
    - The examples directory has been added
- Module names contain
  - Lowercase letters
  - Numbers
  - Underscores
  - Should begin with a lowercase letter
  - Module names cannot contain the namespace separator (::)
  - Modules cannot be nested

Module Layout

- <MODULE NAME>
  - manifests
  - files
  - templates
  - lib
  - facts.d
  - examples
  - spec
  - functions
  - types

Important Directories

- manifests/ - Contains all of the manifests in the module.
- files/ - Contains static files, which managed nodes can download.
- lib/ - Contains plugins, like custom facts and custom resource types.
- facts.d/ - Contains external facts, which are an alternative to Ruby-based custom facts.
- templates/ - Contains templates, which the module's manifests can use.
- examples/ - Contains examples showing how to declare the module's classes and defined types.


Linux Academy
System Administration
Using Puppet Certification
Facter

Facter

- Facter is Puppet's cross-platform system profiling library. It discovers and reports per-node facts, which are available in your Puppet manifests as variables.
  - Core Facts: Built-in fact that ships with Facter
  - External Facts: Provide a way to use arbitrary executables or scripts as facts
  - Custom Facts: Extend Facter by writing Ruby code
- Facter Command
  - facter: Returns a list all facts.
  - facter <fact>: Returns a particular fact.
  - facter -p: Allows Facter to load Puppet-specific facts.


Linux Academy
System Administration
Using Puppet Certification
Autoloading

Autoloading

- Puppet will use a class's full name to find it in your module.
- Every class should be in its own file and use the .pp file extension.
- Names map to the file
  - First segment in a name identifies the module.
  - init.pp class will always be the module name.
  - The last segment identifies the file name.
  - Any segments between the first and last are subdirectories in the manifests directory.

Example:

apache - <MODULE DIRECTORY>/apache/manifests/init.pp
apache::mod - <MODULE DIRECTORY>/apache/manifests/mod.pp
apache::mod::passenger - <MODULE DIRECTORY>/apache/manifests/mod/passenger.pp


Linux Academy
System Administration
Using Puppet Certification
DSL Overview

Resource Types: Review

- Resource types are the basic building blocks of the Puppet DSL.
- Every resource type has:
  - a title
  - a set of attributes

Resource Type Syntax:

```
<TYPE> { '<TITLE>':
  <ATTRIBUTE> => <VALUE>,
}
```

- The attributes (sometimes called parameters) of a resource determine its desired state.

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

Attributes:

- 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,
}
```

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

Case statements

```
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,
}
```

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

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**: require 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.

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':
  name   => 'openssh',
  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; subscribe watches for changes, so will only restart the sshd service if Puppet makes a change the ssh_config file. Subscribe also implies require.

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.

System Administration Using Puppet Certification

The Puppet Style Guide

Puppet Language and Style

The style guide is to promote consistent formatting in the Puppet Language, especially across modules, giving users and developers of Puppet modules a common pattern, design, and style to follow.

- Readability matters.
- Scoping and simplicity are key.
- Your module is a piece of software.
- Version your modules.

Spacing, Indentation, and Whitespace

- Module manifests:
  - Must use two-space soft tabs,
  - Must not use literal tab characters,
  - Must not contain trailing whitespace,
  - Must include trailing commas after all resource attributes and parameter definitions,
  - Must end the last line with a new line,
  - Must use one space between the resource type and opening brace, one space between the opening brace and the title, and no spaces between the title and colon.

Example:

```
file { '/tmp/foo':  … }
```

Spacing, Indentation, and Whitespace

- Module manifests:
  - Should not exceed a 140-character line width, except where such a limit would be impractical
  - Should leave one empty line between resources, except when using dependency chains
  - May align hash rockets (=>) within blocks of attributes, one space after the longest resource key, arranging hashes for maximum readability first.

Arrays and Hashes

- Each element on its on line
- Each new element line indented one level
- First and last lines used only for the syntax of that data type

Example:

```
# array with multiple elements on multiple lines
service { 'some_service':
  require => [
    File['some_config_file'],
    File['some_sysconfig_file'],
  ],
}
```

Quoting

- All strings must be enclosed in single quotes, unless the string:
  - Contains variables
  - Contains single quotes
  - Contains escaped characters not supported by single-quoted strings
  - Is an enumerable set of options, such as present/absent, in which case the single quotes are optional
- All variables must be enclosed in braces when interpolated in a string.

Example:

file { “/tmp${file_name}”: … }

"${facts['operatingsystem']} is not supported by ${module_name}"

Quoting

- Double quotes should be used rather than escaping when a string contains single quotes, unless that would require an inconvenient amount of additional escaping.

Example:

warning("Class['class_name'] doesn’t work they way you expected it too.")

Escape Characters and Comments

- Puppet uses backslash as an escape character.
- Comments must be hash comments (# This is a comment), not /* */
- Documentation comments for Puppet Strings should be included for each of your classes, defined types, functions, and resource types and providers.

Example:

Escaping as \ would be “\\”

```
# Configures sshd
file { '/etc/ssh/ssh_config': ... }
```

Module Metadata

- Every module must have metadata defined in the metadata.json file.
- Hard dependencies must be declared in your module's metadata.json file.
- Soft dependencies should in the README.md.
- A full sample is your study guide.

Example:

```
{
  "name": "tthomsen-my_module_name",
  "version": "0.1.0",
  "author": "Travis N. Thomsen",
  "license": "Apache-2.0",
  "summary": "It's a modules that does things",
  "source": "https://github.com/mygithubaccount/tthomsen-my_module_name",
  "project_page": "https://github.com/mygithubaccount/tthomsen-my_module_name",
  "issues_url": "https://github.com/mygithubaccount/tthomsen-my_module_name/issues",
 "tags": ["things", "and", "stuff"],
}
```

Resources

- All resource names or titles must be quoted.
- Hash rockets (=>) in a resource's attribute/value list may be aligned.
- Ensure should be the first attribute specified.
- Resources should be grouped by logical relationship to each other, rather than by resource type.
- Semicolon-separated multiple resource bodies should be used only in conjunction with a local default body.

Example:

```
file { '/etc/ssh/ssh_config':
  ensure => file,
  mode   => "0600",
}
```

Classes and Defined Types

All classes and resource type definitions (defined types) must be separate files in the manifests directory of the module. Each separate file in the manifest directory of the module should contain nothing other than the class or resource type definition.

Example:

```
# /etc/puppetlabs/code/environments/production/modules/apache/manifests
# init.pp
class apache { }

# ssl.pp
class apache::ssl { }

# virtual_host.pp
define apache::virtual_host () { }
```

Classes and Defined Types

When a resource or include statement is placed outside of a class, node definition, or defined type, it is included in all catalogs. This can have undesired effects and is not always easy to detect.

Example:

```
#manifests/init.pp:
class { 'some_class':
  include some_other_class
}
```

Chaining Arrow Syntax

- When you have many interdependent or order-specific items, chaining syntax may be used.

Example:

```
# Points left to right
Package['package_name'] -> Service['service_name']

# On the line of the right-hand operand
Package['package_name']
-> Service['service_name']
```

Nested Classes or Defined Types

- Don't define Classes and defined resource types in other classes or defined types.
- Classes and defined types should be declared as close to node scope as possible.
- Don’t use nesting mkay!

Example of bad behavior:

```
class some_class {
  class a_nested_class { ... }
}

class some_class {
  define a_nested_define_type() { ... }
}
```

Parameter

- Declare required parameters before optional parameters.
- Optional parameters are parameters with defaults.
- Declare the data type of parameters, as this provides automatic type assertions.
- For Puppet 4.9.0 and greater, use Hiera data in the module and rely on automatic parameter lookup for class parameters.
- Puppet versions less than 4.9.0, use the “params.pp” pattern. In simple cases, you can also specify the default values directly in the class or defined type.

Example:

```
# parameter defaults provided via APL > puppet 4.9.0
class some_module (
  String $source,
  String $config,) {
  ... puppet code ...
}
```

Class Inheritance

- Class inheritance should not be used.
- Use data binding instead of params.pp pattern.
- Inheritance should only be used for params.pp, which is not recommended in Puppet 4.
- For maintaining older modules inheritance can be used but must not be used across module namespaces.

Example:

```
class ssh { ... }

class ssh::client inherits ssh { ... }

class ssh::server inherits ssh { ... }
```

Defined Resource Types

- Defined resource types are not singletons.
- Uniqueness
  - Can have multiple instances.
  - Resource names must be unique.

Variables

- Referencing facts
  - When referencing facts, prefer the $facts hash to plain top-scope variables.
    - It's clearer.
    - It’s easier to read.
    - Distinguishes facts from other top-scope variables.
  - **Example:** $facts[::operatingsystem]
- Namespacing variables
  - When referencing top-scope variables other than facts, explicitly specify absolute namespaces for clarity and improved readability. This includes top-scope variables set by the node classifier and in the main manifest.
  - This is not necessary for:
    - the $facts hash.
    - the $trusted hash.
    - the $server_facts hash.

Variables

- Variable format
  - Use numbers
  - Use lowercase letters
  - Use underscores
  - Don't use camel case
  - Don't use dashes

Good Examples:

- $this_is_vairable
- $so_is_this
- $also_good123

Bad Examples:

- $ThisIsNotGood
- $neither-is-this

Conditionals

- Keep resource declarations simple.
  - Don't mix conditionals with resource declarations.
  - Separate conditional code from the resource declarations.

Example:

```
$file_mode = $::operatingsystem ? {
  'debian' => '0007',
  'redhat' => '0776',
   default => '0700',
}

file { '/tmp/readme.txt':
  ensure  => file,
  content => "Hello World\n",
  mode    => $file_mode,
}
```

Conditionals

- Defaults for case statements and selectors.
  - Case statements must have default cases.
  - Case and selector values must be quoted.

Example:

```
case $Facts[::operatingsystem] {
  'centos': {    $version = '1.2.3'  }
  'debian': {    $version = '3.4.5'  }
  default: {    fail("Module ${module_name} is not supported on ${::operatingsystem}")  }
}
```


Linux Academy
System Administration
Using Puppet Certification
Data Types

Core Data Types

- The most common data types:
  - String
  - Integer, Float, and Numeric
  - Boolean
  - Array
  - Hash
  - Regexp
  - Undef
  - Default

Resource and Class References

- Resources and classes are implemented as data types.
- However, they behave differently from other values.

Abstract Data Types

- Abstract data types let you do more sophisticated or permissive type checking.
  - Scalar
  - Collection
  - Variant
  - Data
  - Pattern
  - Enum
  - Tuple
  - Struct
  - Optional
  - Catalogentry
  - Type
  - Any
  - Callable

The Type Data Type

All data types are of type Type.

Syntax:
Type[<ANY DATA TYPE>]

Example:
Type: matches any data type, such as Integer, String, Any, or Type.
Type[String]: matches the data type String, as well as any of its more specific subtypes like String[3] or Enum["running", "stopped"].
Type[Resource]: matches any Resource data type - that is, any resource reference.


Linux Academy
System Administration
Using Puppet Certification
Relationships and Dependencies

Relationship Metaparameters

By default, Puppet applies resources in the order they're declared in their manifest. However, if a group of resources must always be managed in a specific order, you should explicitly declare such relationships with relationship metaparameters, chaining arrows, and the require function.

- **before:** Applies a resource before the target resource.
- **require:** Applies a resource after the target resource.
- **notify:** Applies a resource before the target resource. The target resource refreshes if the notifying resource changes.
- **subscribe:** Applies a resource after the target resource. The subscribing resource refreshes if the target resource changes.

Chaining Arrows

You can create relationships between two resources or groups of resources using the -> and ~> operators.

- -> ordering arrow: Applies the resource on the left before the resource on the right.
- ~> notifying arrow: Applies the resource on the left first. If the left-hand resource changes, the right-hand resource will refresh.

Both chaining arrows have a reversed form (<- and <~).

Chaining Arrows: Operands

The chaining arrows accept the following kinds of operands on either side of the arrow:

- Resource references, including multi-resource references
- Arrays of resource references
- Resource declarations
- Resource collectors

Chaining Arrows: Operands

The chaining arrows accept the following kinds of operands on either side of the arrow:

- Resource references, including multi-resource references
- Arrays of resource references
- Resource declarations
- Resource collectors

Ordering

All relationships cause Puppet to manage one or more resources before one or more other resources.

By default, unrelated resources are managed in the order in which they're written in their manifest file. If you declare an explicit relationship between resources, it will override this default ordering.

Refreshing and Notification

- Some resource types can be refreshed when a dependency is changed.
- Built-in resource types that can refreshed:
  - service
  - mount
  - exec
  - Sometimes package
- Rules for notification and refreshing are:
  - Receiving refresh events
  - Sending refresh events
  - No-op

Refreshing and Notification

- Certain resource types can have automatic relationships with other resources, using autorequire, autonotify, autobefore, or autosubscribe.
- A complete list can be found in the resource type reference.
- Auto relationships between types and resources are established when applying a catalog.

Missing Dependencies

If one of the resources in a relationship is not declared the catalog will fail to compile.

- Could not find dependency <OTHER RESOURCE> for <RESOURCE>
- Could not find resource '<OTHER RESOURCE>' for relationship on '<RESOURCE>'.

Failed dependencies

If a resource with dependencies fails to be applied, all dependent resource will be skipped.

- notice: <RESOURCE>: Dependency <OTHER RESOURCE> has failures: true
- warning: <RESOURCE>: Skipping because of failed dependencies

Dependency Cycles

If two or more resources require each other, Puppet compiles the catalog, but it won't be applied because this causes a loop.

- err: Could not apply complete catalog: Found 1 dependency cycle: (<RESOURCE> =>

  <OTHER RESOURCE> => <RESOURCE>)

- Try the '--graph' option and opening the resulting '.dot' file in OmniGraffle or GraphViz


Linux Academy
System Administration
Using Puppet Certification
Conditional Statements

Conditional Statements

Conditional statements let your Puppet code behave differently in different situations. They are most helpful when combined with facts or with data retrieved from an external source.

Conditionals that alter logic:

- if statement
- unless statement
- case statement

Conditionals that return a value:

- selector

“If” Statements

“If” statements take a Boolean condition and an arbitrary block of Puppet code and will only execute the block if the condition is true. They can optionally include elsif and else clauses.

Syntax:

```
if condition {
  block of code
}
elsif condition {
  block of code
}
else {
  default option
}
```

“If” Statements

Example:

```
if $facts['os']['name'] == 'Windows' {
  include role::windows
}
elsif ($facts['os']['name'] == 'RedHat') and ($facts['os']['name'] == 'CentOS')  {
  include role::redhat
}
elsif $facts['os']['name'] =~ /^(Debian|Ubuntu)$/ {
  include role::debian
}
else {
 include role::generic::os
}
```

“If” Statements

- Behavior
  - The Puppet if statement behaves like if statements in any other language.
  - If none of the conditions match and there is no else block, Puppet will do nothing.
- Conditions
  - Variables
  - Expressions, including arbitrarily nested and/or expressions
  - Functions that return values

“If” Statements

- Regex capture variables
  - If you use a regular expression match operator as your condition, any captures from parentheses in the pattern will be available inside the associated code block as numbered variables ($1, $2, etc.), and the entire match will be available as $0:

Example:

```
if $trusted['certname'] =~ /^www(\d+)\./ {
  notice("This is web server number $1.")
}
```

“Unless” Statements

“Unless” is the reversed “if” statements. It takes a Boolean condition and an arbitrary block of Puppet code. It will only execute the block of code if the condition is false. There cannot be an elsif clause.

Syntax:

```
unless condition {
  block of code
}
```

Example:

```
unless $facts['memory']['system']['totalbytes'] > 1073741824 {
  $maxclient = 500
}
```

“Unless” Statements

- Behavior
  - The condition is evaluated first and, if it is false, the code block is executed.
  - If the condition is true, Puppet will do nothing.
  - The unless statement is also an expression that produces a value and can be used wherever a value is allowed.
- Conditions
  - Variables
  - Expressions, including arbitrarily nested and/or expressions
  - Functions that return values
- Regex capture variables
  - Although “unless” statements receive regex capture variables like “if” statements, they usually aren’t used.

“Case” Statements

Similar to the “if” statements, case statements choose one of several blocks of arbitrary Puppet code.

Syntax:

```
case condition {
  'control expression': { block of code }
  default: { block of code }
}
```

Example:

```
case $facts['os']['name'] {
  'Windows':                { include role::windows }
  'RedHat', 'CentOS':       { include role::redhat }
  /^(Debian|Ubuntu)$/: { include role::debian }
  default:                  { include role::generic::os }
}
```

“Case” Statements

- Behavior
  - Compares the control expression to each of the cases in the order they are defined.
  - The default case is always evaluated last.
  - The code block for the first matching case is executed.
  - A maximum of one code block will be executed.
  - If none of the cases match, Puppet will do nothing.
- Conditions
  - Variables
  - Expressions, including arbitrarily nested and/or expressions
  - Functions that return values

“Case” Statements

- Case matching
  - Most data types == equality operator
  - Regular expressions =~ matching operator
  - Data types =~ matching operator
  - Arrays are compared to the control value recursively.
  - Hashes compare each key/value pair.
  - default matches anything and unless nested inside an array or hash is always tested last, regardless of its position in the list.
- When used as a value
  - In addition to executing the code in a block, a case statement is also an expression that produces a value, and can be used wherever a value is allowed.
  - The value of a case expression is the value of the last expression in the executed block, or undef if no block was executed.

“Case” Statements

- Regex capture variables
  - If you use a regular expression match operator as your condition, any captures from parentheses in the pattern will be available inside the associated code block as numbered variables ($1, $2, etc.), and the entire match will be available as $0:

Example:

```
case $trusted['certname'] {
  /www(\d+)/: { notice("This is web server number $1."); }
  default:        { notice("Now for something completely different") }
}
```

Selectors

Selector expressions are similar to case statements but return a value. You should generally only use selectors in variable assignments.

Syntax:

```
case condition {
  'control expression': { block of code }
  default: { block of code }
}
```

Example:

```
$role = $facts['os']['name'] ? {
  'Windows'                 => 'role::windows',
  /^(Debian|Ubuntu)$/ => 'role::debian',
  default                   => 'role::redhat',
}
```

Selectors

- Behavior
  - The entire selector expression is treated as a single value.
  - The control expression is compared to each of the cases in the order they are defined.
  - The default case is evaluated last.
  - The value of the matching case is returned.
  - If no conditions match, the catalog will fail to compile.
- Conditions
  - Variables
  - Expressions, including arbitrarily nested and/or expressions
  - Functions that return values

Selectors

- Case matching
  - You cannot use lists of cases.
  - Most data types == equality operator
  - Regular expressions =~ matching operator
  - Data types =~ matching operator
  - Arrays are compared to the control value recursively.
  - Hashes compare each key/value pair.
  - default matches anything and unless nested inside an array or hash is always tested last, regardless of its position in the list.

Selectors

- Regex capture variables
  - If you use a regular expression match operator as your condition, any captures from parentheses in the pattern will be available inside the associated code block as numbered variables ($1, $2, etc.), and the entire match will be available as $0:

Example:

```
$role = $facts['os']['name'] ? {
  /^(Debian|Ubuntu)$/ => "You are running ${1}",
  default                    => "You are running an unknown operating system!",
}
```

System Administration Using Puppet Certification

Templates

Templates

- **template:** Loads an ERB template from a module, evaluates it, and returns the resulting value as a string.
- A template is referenced by template(<MODULE NAME>/<TEMPLATE FILE>)
  - template('modulename/motd.erb')
- The file is located in <MODULES DIRECTORY>/<MODULE NAME>/templates/motd.erb.

Example:

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

Embedded Ruby (ERB) Template Syntax

- ERB is a templating language based on Ruby.
- Puppet uses the template and inline_template functions to evaluate a template file.

Expression-printing:
<%= @value %>

If statement:
<% if condition %> ...text... <% end %>

Comments:
<%# This is a comment. %>

Looping:
<% @valuse.each do |value| -%>
some value <%= value %>
<% end -%>


Linux Academy
System Administration
Using Puppet Certification
Class Parameters and Defaults

Parameters

- Classes, defined types, and lambdas can all take parameters.
- A way for you to pass external data.

Syntax:

```
class <CLASS NAME> (
  <DATA TYPE> <PARAMETER NAME>,
  <DATA TYPE> <PARAMETER NAME> = <VALUE>,
  # ...
) {
  # ...
}
```

Parameters

Example:

```
class ntp (
  Boolean $service_manage = true,
  Boolean $autoupdate     = false,
  String  $package_ensure = 'present',
  # ...
)
{
  # ...
}
```

params.pp

- The main classes inherit from a <MODULE>::params class, which only sets variables.
- Using the “params.pp” pattern is now deprecated.
- Using a function or Hiera for your defaults data is now the recommended method.

Function Data Provider

- The function provider calls a function named <MODULE NAME>::data.
- This function is similar to the params.pp file.
- It takes no arguments and returns a hash.
- Puppet will try to find the requested data as a key in that hash.
- The <MODULE NAME>::data function can be one of:
  - A Puppet language function, located at <MODULE ROOT>/functions/data.pp.
  - A Ruby function (using the modern Puppet::Functions API), located at <MODULE ROOT>/lib/puppet/functions/<MODULE NAME>/data.rb.

Function Data Provider

Example:

```
# ntp/metadata.json
{
  ...  "data_provider": "function”
}

# ntp/manifests/init.pp
class ntp (
  # default values are in ntp/functions/data.pp
  $autoupdate,
  $service_name,
)
{
 ...
}
```

Function Data Provider

```
# ntp/functions/data.pp
function ntp::data() {
  $base_params = {
    'ntp::autoupdate'   => false,
    'ntp::service_name' => 'ntpd',
  }
 $os_params = case $facts['os']['family'] {
    'AIX': {
      { 'ntp::service_name' => 'xntpd' }
    }
    'Debian': {
      { 'ntp::service_name' => 'ntp' }
    }
    default: {
      {}
    }
  }
  # Merge the hashes and return a single hash.
  $base_params + $os_params
}
```


Linux Academy
System Administration
Using Puppet Certification
Metaparameters

Metaparameters

- Metaparameters are attributes that all resource type, custom types and defined types have.
- Available metaparameters:
  - alias
  - audit
  - consume
  - export
  - loglevel
  - noop
  - schedule
  - stage
  - tag


Linux Academy
System Administration
Using Puppet Certification
Puppet Functions

Functions

- There are two types of functions in statements and rvalues functions.
- Statements
  - They do not return arguments.
- Rvalues
  - They return values.
  - They can only be used in a statement requiring a value.
    - variable assignment
    - case statement

Functions

- Statement Functions
  - **alert:** Log a message on the server at level alert.
  - **create_resources:** Converts a hash into a set of resources and adds them to the catalog.
  - **err:** Log a message on the server at level err.
  - **fail:** Fail with a parse error.
  - **hiera_include:** Uses an array merge lookup to retrieve the classes array, so every node gets every class from the hierarchy.
  - **include:** Declares one or more classes, causing the resources in them to be evaluated and added to the catalog.
  - **warning:** Log a message on the server at level warning.

Functions

- Rvalue Functions
  - **defined:** Determines whether a given class or resource type is defined and returns a Boolean value.
  - **file:** Loads a file from a module and returns its contents as a string.
  - **generate:** Calls an external command on the Puppet master and returns the results of the command.
  - **hiera:** Performs a standard priority lookup of the hierarchy and returns the most specific value for a given key.

Functions

- Rvalue Functions
  - **hiera_array:** Finds all matches of a key throughout the hierarchy and returns them as a single flattened array of unique values.
  - **hiera_hash:** Finds all matches of a key throughout the hierarchy and returns them in a merged hash.
  - **regsubst:** Perform regex replacement on a string or array of strings.
  - **sha1:** Returns a SHA1 hash value from a provided string.
  - **template:** Loads an ERB template from a module, evaluates it, and returns the resulting value as a string.


Linux Academy
System Administration
Using Puppet Certification
Iteration and Loops

Iteration and Loops

- Iteration features are implemented as functions that accept blocks of code called lambdas.
- List of iteration functions
  - **each:** Repeat a block of code any number of times, using a collection of values to provide different parameters each time.
  - **slice:** Repeat a block of code any number of times, using groups of values from a collection as parameters.
  - **filter:** Use a block of code to transform some data structure by removing non-matching elements.
  - **map:** Use a block of code to transform every value in some data structure.
  - **reduce:** Use a block of code to create a new value or data structure by combining values from a provided data structure.
  - **with:** Evaluate a block of code once, isolating it in its own local scope. Doesn't iterate but has a family resemblance to the iteration functions.

Iteration and Loops

Example:

```
$values = ['a', 'b', 'c', 'd', 'e']

# function call with lambda:
$values.each |String $value| {
  notify { $value:
    message => "Value from a lambda code block: ${value}”
  }
}
```


Linux Academy
System Administration
Using Puppet Certification
Variables and Scope

Variables

- Variables store values so they can be accessed later.
- Variables are actually constants and can't be reassigned.
- Facts and built-in variables.
- Variable names are prefixed with a $ (dollar sign).
- They are assigned using the = (equal sign) assignment operator.
- Variable names can include:
  - Uppercase and lowercase letters
  - Numbers
  - Underscores

Example:

```
$variable_name1 = "value"
```

Variables

- Append a variable by using the + symbol
  - $variable = ['a', 'b']
  - $variable += ['c']
  - $variable now equals ['a', 'b', 'c']
- Assigning multiple variables
  - You can assign multiple variables at once from an array or hash.
  - Arrays
    - When using an array you need an equal number of variables and values.
    - Arrays can be nested.
  - Hashes
    - Variables are listed in an array on the left side of the assignment operator.
    - The hash is on the right of the assignment operator.
    - Hash keys must match their corresponding variable name.

Variables

Array Assignment Example:

```
[$a, $b, $c] = [1,2,3]       # $a = 1, $b = 2, $c = 3
[$a, [$b, $c]] = [1,[2,3]]  # $a = 1, $b = 2, $c = 3
[$a, $b] = [1, [2]]          # $a = 1, $b = [2]
[$a, [$b]] = [1, [2]]        # $a = 1, $b = 2
```

Hash Assignment Example:

```
[$a, $b] = {a => 10, b => 20}                          # $a = 10, $b = 20
[$a, $c] = {a => 5, b => 10, c => 15, d => 22}   # $a = 5, $c = 15
```

Variable Interpolation

- Variable interpolation is when a variable is resolved in a double-quoted strings.
- Inside the double-quoted strings the variable is referenced using a dollar sign with curly braces.
- ${var_name}
- Single quotes will treat the variable as a literal.

Example:

```
$variable = "${some_other_variable} is being interpolation in here."
```

Arrays and Hashes

- Arrays
  - Arrays are ordered lists of values.
  - There are functions that take arrays as parameters, including the iteration functions like each.
- Hashes
  - Hashes map keys to values.
  - The entries are maintained in the order they were added.
  - Hashes are merged using the + operator.

Array Example:

```
$array_variable = [ 'a', 'b', 'c' ]
```

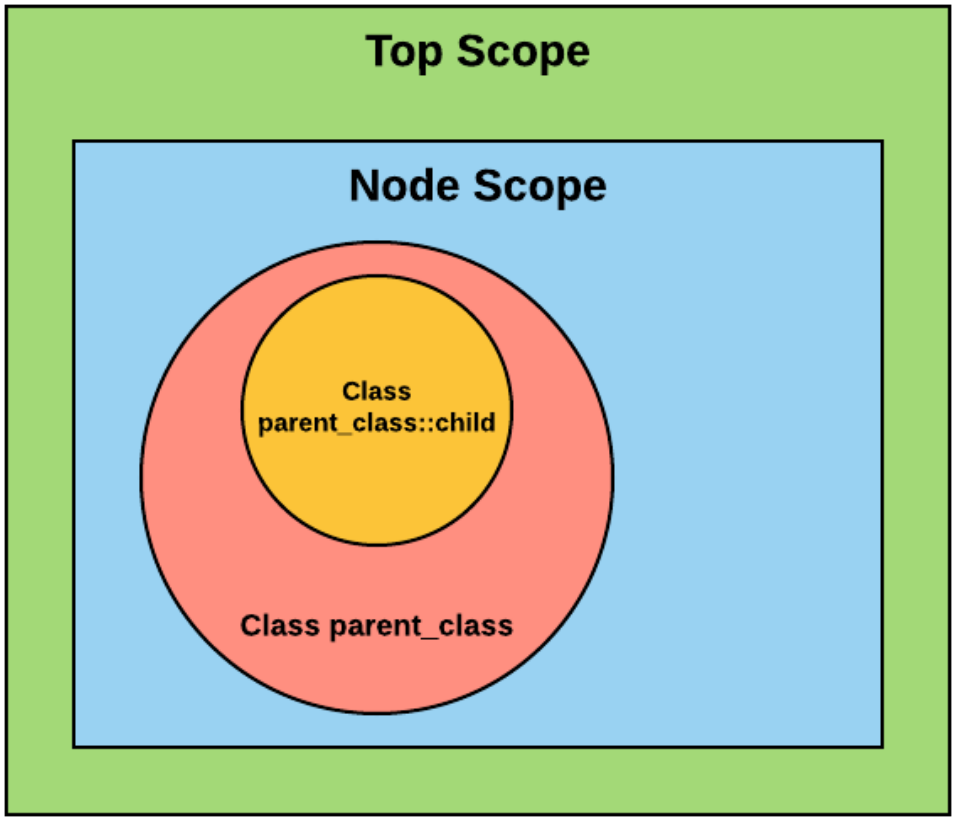
Hash Example:

```
$hash_variable = { key1 => "value1", key2 => "value2" }
```

Scope

- Scope is a specific area of code that is partially isolated from other areas of code.
- Top scope
  - Code that is outside any class definition, type definition, or node definition exists at top scope. Variables and defaults declared at top scope are available everywhere.
- Node scope
  - Code inside a node definition exists at node scope. Note that since only one node definition can match a given node, only one node scope can exist at a time.
- Local scopes
  - Code inside a class definition, defined type, or lambda exists in a local scope.
  - Variables and defaults declared in a local scope are only available in that scope and its children.

Scope


Top Scope
Node Scope
Class
parent_class::child
Class parent_class


Linux Academy
System Administration
Using Puppet Certification
Defined Resource Types

Defined Resource Types

- Defined resource types also called defined types or defines.
- Are blocks of code that can be evaluated multiple times with different parameters.
- They act like a new resource type.
- They are declared like a resource type.
- Definitions should be stored in the manifests/ directory.
- Defined type instance can include any metaparameter.

Defined Resource Types

- Defined type names can consist of one or more namespace segments.
- Each namespace segment must begin with a lowercase letter and can include:
  - Lowercase letters
  - Digits
  - Underscores
- Namespace segments should match the following regular expression:
  - \A[a-z][a-z0-9_]*\Z
  - define_name123
- Multiple namespace segments can be joined together in a define type name with the ::

  (double colon) namespace separator.
  - \A([a-z][a-z0-9_]*)?(::[a-z][a-z0-9_]*)*\Z
  - module_name::defined_type_name

Defined Resource Types

Syntax:

```
define name (
 <DATA TYPE> <PARAMETER> = <VALUE>,
) {
  ... puppet code ...
}
```

Declaring an instance:

```
<DEFINED TYPE> { '<TITLE>':
  <ATTRIBUTE> => <VALUE>,
}
```


Linux Academy
System Administration
Using Puppet Certification
Resource Collectors

Resource Collectors

- Resource collectors, also called the spaceship operator.
- It selects a group of resources by searching the attributes of every resource in the catalog.
- This search is independent of evaluation order.
- Collectors realize virtual resources.
- Can be used in chaining statements.
- Can override resource attributes.
- Can function as both a statement and a value.
- The resource type, capitalized.

Resource Collectors

- Operators
  - ==
  - !=
  - and
  - or

Syntax:

```
<RESOURCE TYPE> <| <SEARCH EXPRESSION> |> {
  …
}
```

Example:

```
User <| groups == 'admin' |> {
  …
}
```


Linux Academy
System Administration
Using Puppet Certification
Roles and Profiles Overview

Overview

- The roles and profiles are used to build reliable, reusable, configurable, and refactorable system configurations.
- They are two extra layers of indirection between your node classifier and your component modules.
- Component modules: Normal modules that manage one particular technology. (For example, puppetlabs/apache.)
- Profiles: Wrapper classes that use multiple component modules to configure a layered technology stack.
- Roles: Wrapper classes that use multiple profiles to build a complete system configuration.

Profiles

- A profile is just a normal class stored in the profile module.
- Make sure you can safely include any profile multiple times - don't use resource-like declarations on them.
- Profiles can include other profiles.
- Profiles own all the class parameters for their component classes.
- Components class shouldn't use a value from Hiera data.
- Ways a profile can get the data it needs to configure component classes:
  - Hardcode it in the profile.
  - Look it up from Hiera.

Profiles

Example:

```
class profiles::apache(
  String $apache_vhost_name,
  String $apache_vhost_docroot,
  Bolean $apache_default_vhost = false,
  String $apache_vhost_port       = 80,
) {
  class { 'apache':
    default_vhost => $apache_default_vhost,
  }

  apache::vhost { $apache_vhost_name:
    port       => $apache_vhost_port,
    docroot => $apache_vhost_docroot,
  }
}
```

Roles

- The only thing roles should do is declare profile classes.
- Use include <PROFILE NAME>.
- Don’t declare any component classes or normal resources in a role.
- Roles can use conditional logic to decide which profiles to use. Roles should not have any class parameters of their own.
- Roles should not set class parameters for any profiles.
- The name of a role should be based on your business’s conversational name for the type of node it manages.

Roles

Roles Names Example:

role::web
role::jenkins::master
role::jenkins::slave

Example:

```
class role::web {
  include profile::base
  include profile::apache
  include profile::php
}
```

Roles

- Assigning a role to a node
  - The PE console node classifier.
  - The main manifest.
  - Hiera or Puppet lookup.

System Administration Using Puppet Certification

Hiera Overview

Overview

- Hiera is a key/value data store for looking up data.
- Let you set node-specific data without repeating yourself.
- Why use Hiera?
  - Single source of truth for your data.
    - Configure default data with hierarchal overrides.
  - Use Puppet modules from the Forge.
    - No need to edit the module, just put the data in Hiera.
  - Publish your own modules for collaboration.
    - Keeps your data out of your module before sharing it.
    - No more clashing variable names.

Setting Up Hiera

- hiera.yaml is located in /etc/puppetlabs/puppet/
- .:backends: tells Hiera what kind of data sources it should process. In this case, we'll be using YAML files.
- :yaml: configures the YAML data backend.
- :datadir: tells Hiera the location of the data sources.
- :hierarchy: configures the data sources Hiera be using.
  - Separate their hierarchies into directories.
  - More specific data at the top.
  - Least specific at the bottom.

Setting Up Hiera

- You can use facts in your Hiera lookups.

hiera.yaml

```
---:backends:
  - yaml
:yaml:
  :datadir: "/etc/puppetlabs/code/environments/%{environment}/hieradata”
:hierarchy:
  - "nodes/%{::trusted.certname}”
  - common
```

Automatic Parameter Lookup

- Process of automatic parameter lookup:
- Look for parameters passed using the class {} declaration.
  - If no parameter is passed, it will look in the Hiera data source for the parameter <CLASS NAMESPACE>::parameter
  - If it's not found in the Hiera data source it will use the default set "common"

Example:

```
define apache::vhost(
  $port,
)

#/etc/puppetlabs/code/environments<ENVIRONMENT>/hieradata/common.yml
---
apache::vhost::port: 80
```

hiera Function

- **hiera:** Performs a standard priority lookup of the hierarchy and returns the most specific value for a given key. The returned value can be any type of data.
- Arguments:
  - A string key that Hiera searches for in the hierarchy. Required.
  - An optional default value to return if Hiera doesn't find anything matching the key.
  - The optional name of an arbitrary hierarchy level to insert at the top of the hierarchy.

hiera_array Function

- **hiera_array:** Finds all matches of a key throughout the hierarchy and returns them as a single flattened array of unique values. If any of the matched values are arrays, they're flattened and included in the results. This is called an array merge lookup.
- Arguments:
  - A string key that Hiera searches for in the hierarchy. Required.
  - An optional default value to return if Hiera doesn't find anything matching the key.
  - The optional name of an arbitrary hierarchy level to insert at the top of the hierarchy.

hiera_hash Function

- **hiera_hash:** Finds all matches of a key throughout the hierarchy and returns them in a merged hash. If any of the matched hashes share keys, the final hash uses the value from the highest priority match. This is called a hash merge lookup.
- Arguments:
  - A string key that Hiera searches for in the hierarchy. Required.
  - An optional default value to return if Hiera doesn't find anything matching the key.
  - The optional name of an arbitrary hierarchy level to insert at the top of the hierarchy.

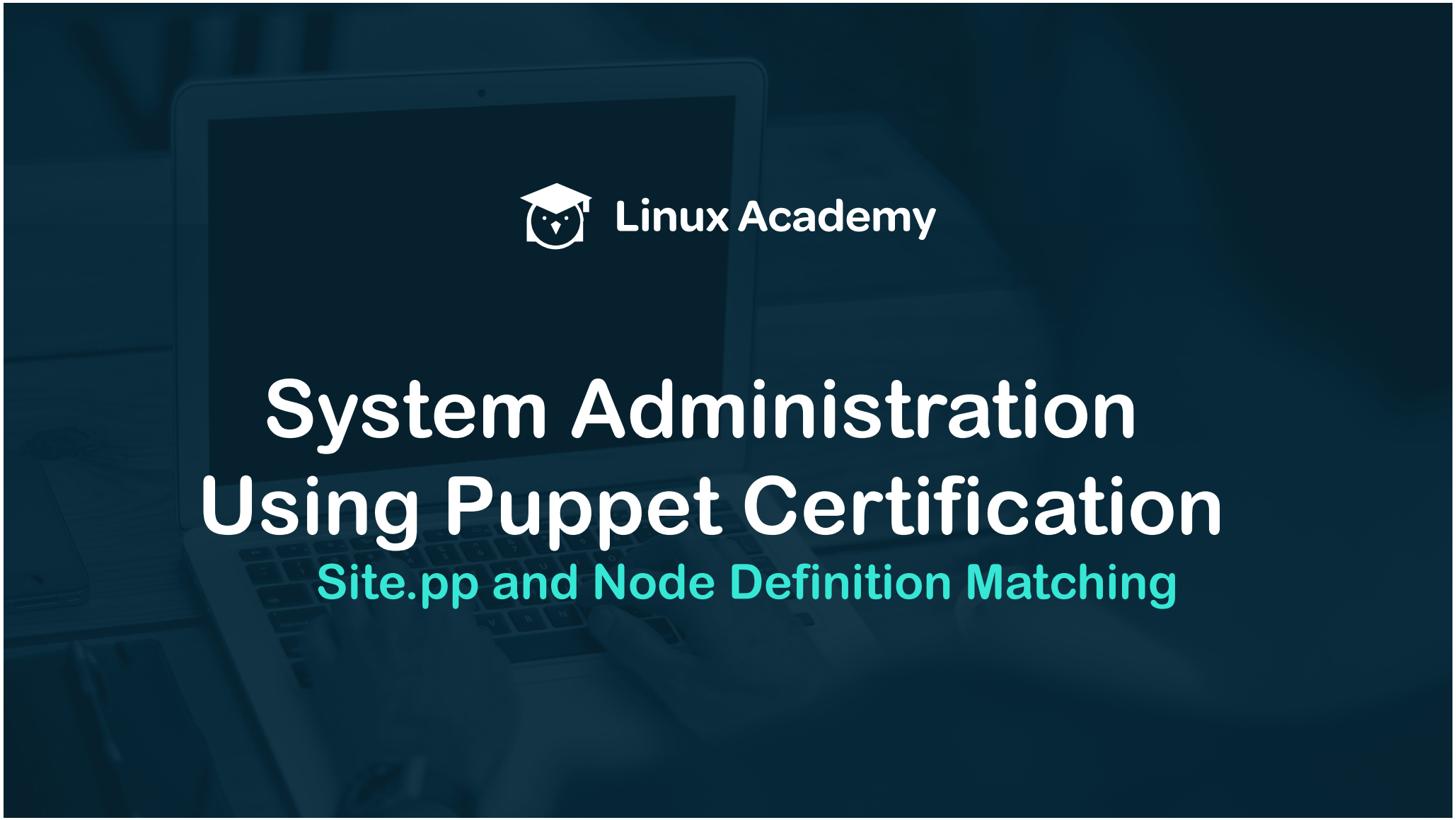

Linux Academy
System Administration
Using Puppet Certification
Managing and Deploying Puppet Code Overview

Overview

- Code Manager and r10k are used to manage and deploying your Puppet code.
  - Install Puppet modules.
  - Create and maintain environments.
  - Deploy new code to your masters.
  - Keep your module code in Git.
- Code Manager automates the management and deployment of your new Puppet code.
  - Push your code updates to your Git repository.
  - Puppet creates environments based off of the branch.
  - Installs modules.
  - Deploys and syncs the new code to your masters.
  - All without interrupting agent runs.

Overview

- You can use r10k to manage your Puppet code instead of Code Manager.
  - You should really use Code Manager.
  - Code Manager works with r10k.
- Both tool are built into Puppet Enterprise.
- Create a control repository for maintaining your environments and code.
- Set up Puppetfiles if you want to install modules in your environments.
- Configure Code Manager (recommended) or r10k. Existing environments will not preserved.
- /etc/puppetlabs/code/environments/production will be overwritten.


Linux Academy
System Administration
Using Puppet Certification
Site.pp and Node Definition Matching

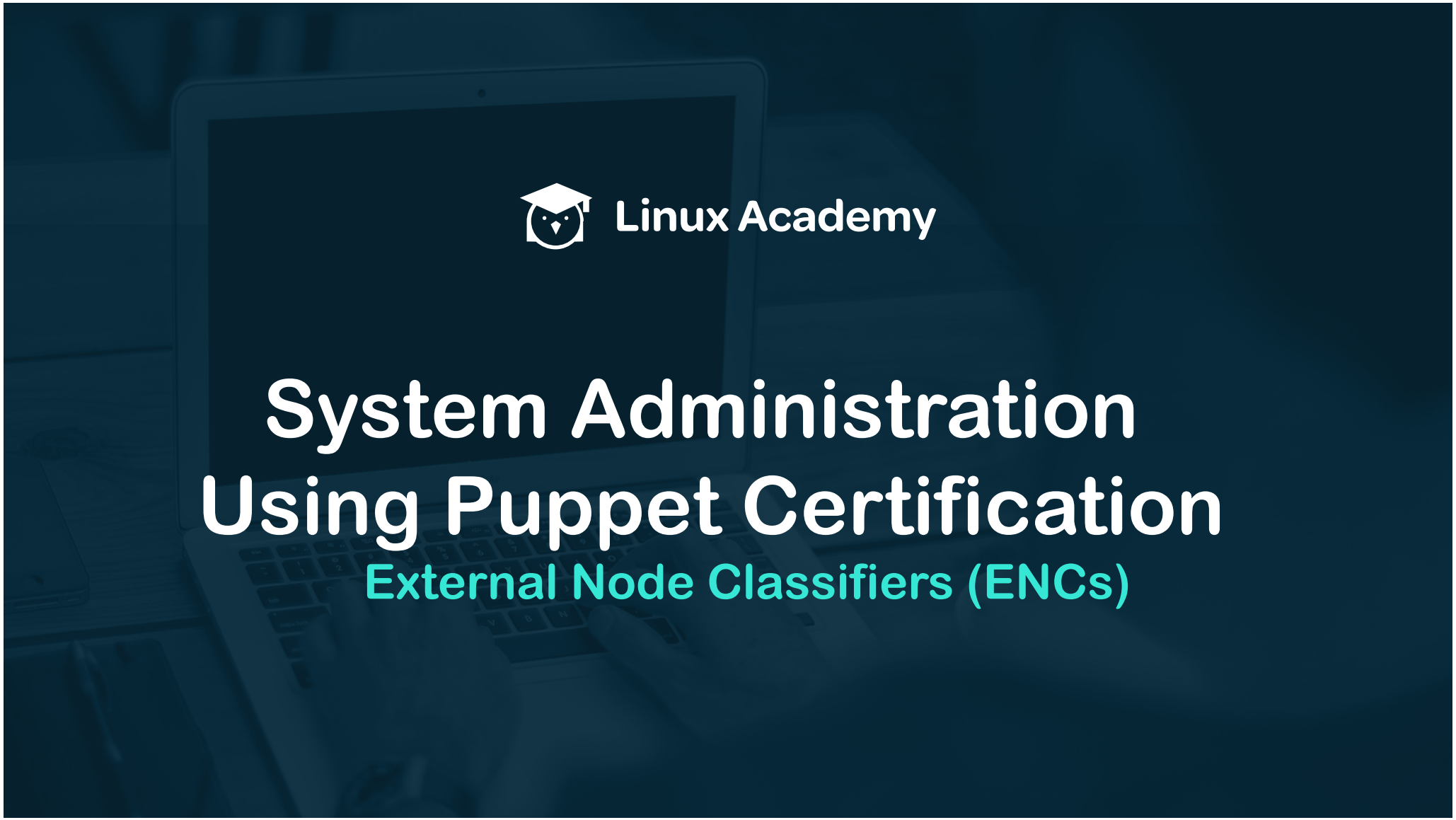
Node Definition Lookup

- If a node is defined within the site.pp manifest then a class MUST be declared.
- Name lookup process:
  - Puppet will look for exact definition matches within site.pp and the certificate name.
  - Puppet will then evaluate any regular expressions if there is no exact match.
  - If the nodes name looks like a FQDN, then Puppet will do the following:
    - Chop off the final group and start at step 1 (exact match), then regular expression match.
    - If there is no match, Puppet will remove group two and start again at step 1.
    - If no match, Puppet will resort to the default definition if it is defined.

Node Definition Lookup

- Node Definition Lookup Example: webserver01.mylabserver.com
- Note: assuming no exact matches or regular expression matches
- Attempt to match webserver01.mylabserver.com, if no match (either exact or regular expression)
- Attempt webserver01.mylabserver, if no match (either exact or regular expression)
- Attempt webserver01, if no (either exact or regular expression)
- Match Default
- No Match (if no default)

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.



System Administration Using Puppet Certification

External Node Classifiers (ENCs)

External Node Classifiers (ENCs)

- An external node classifier is an arbitrary script or application which can tell Puppet which classes a node should have.
- It can replace or work in concert with the node definitions in the main site manifest (site.pp).
- An external node classifier is an executable that can be called by puppet master; it doesn't have to be written in Ruby. Its only argument is the name of the node to be classified, and it returns a YAML document describing the node.
- ENCs can co-exist with standard node definitions in site.pp, and the classes declared in each source are effectively merged.

External Node Classifiers (ENCs)

- **node_terminus:** Tells Puppet what the ENC it will be using.
  - Default node_terminus=classifier
- **external_nodes:** This is the path to the executable of the ENC
- Replace node_terminus=classifier with node_terminus=exec.

Example:

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

Using Hiera as an ENCs

- **hiera_include:** Assigns classes to a node using an array merge lookup that retrieves the value for a user-specified key from Hiera's data.
- You can use Hiera as an ENC by:
  - Use your default node in sites.pp
  - Add hiera_include('classes'))
  - Define classes in your Hiera data.

Example:

```
# Assuming apache.yaml:
classes:
  - role::apache

# Assuming common.yaml:
classes:
   - role::base
```

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
System Administration
Using Puppet Certification
MCollective Overview

Overview

- Enterprise includes the MCollective.
- Used to invoke actions in parallel across multiple nodes.
- You can write custom plugins to add new actions.
- MCollective is built around the idea of predefined actions.
- It is essentially a highly-parallel remote procedure call (RPC) system.
- Actions are distributed in plugins.

Overview

- MCollective Plugins:
  - **package:** Install and uninstall software packages.
  - **puppet:** Run Puppet agent, get its status, and enable/disable it.
  - **puppetral:** View resources with Puppet's resource abstraction layer.
  - **rpcutil:** General helpful actions that expose stats and internals to SimpleRPC clients.
  - **service:** Start and stop system services.

Overview

- MCollective Components:
  - **pe-activemq:** Service routes all MCollective-related messages.
  - **pe-mcollective:** Service listens for authorized commands and invokes actions in response.
  - **mco:** Command can issue authorized commands to any number of nodes.


Linux Academy
System Administration
Using Puppet Certification
Using MCollective

Using MCollective

- To run MCollective commands you must:
  - Be logged in to the Puppet master server.
  - Use the peadmin user account.
  - By default, the peadmin account cannot log in with a password.
- Using sudo
  - sudo -i -u peadmin
- Adding SSH keys
  - You can have other users to run commands.
  - Add the user's public SSH keys to peadmin’s authorized keys file.
  - /var/lib/peadmin/.ssh/authorized_keys

Using MCollective

- The mco command
  - All MCollective actions are invoked with mco.
  - The mco command relies on a config file.
    - /var/lib/peadmin/.mcollective
    - It is only readable by the peadmin use.

Using mco help:

mco help
mco help <SUBCOMMAND>
mco <SUBCOMMAND> --help

Using MCollective

Synstax

mco <SUBCOMMAND> <ACTION>
mco rpc <AGENT PLUGIN> <ACTION> <INPUT>=<VALUE>

Examples:

mco ping
mco rpc rpcutil ping
mco rpc service restart service=puppet


Linux Academy
System Administration
Using Puppet Certification
Troubleshooting

Troubleshooting the Installer

- Common installer problems:
  - Check your DNS
  - Puppet communicates on ports 8140, 61613, and 443.
  - If you are installing the console and the Puppet master on separate servers and tried to install the console first, the installer may fail.
- Recovering from a failed install.
  - If you encounter errors during installation, you can fix them and run the installer again.

Troubleshooting Connections

- Troubleshooting connections between components
  - Is the agent able to reach the Puppet master?
  - Try telnet <puppet master's hostname> 8140
  - Make sure the agent can reach the DNS name that is configured in puppet.conf.
  - Check that the pe-puppetserver service is running.

Troubleshooting Connections

- Make sure the agent has a signed certificate.
- Check the logs for:
  - warning: peer certificate won't be verified in this SSL session
- Revoke the certificate and regenerate it.
  - On the master:
    - puppet cert clean <NODE NAME>
  - On the agent:
    - rm -r $(puppet agent --configprint ssldir)
    - puppet agent -t (or --test)

Troubleshooting Connections

If you get the following error during a Puppet run:

err: /Stage[main]/Pe_mcollective/File[/etc/puppetlabs/mcollective/server.cfg]/content:change from {md5}778087871f76ce08be02a672b1c48bdc to{md5}e33a27e4b9a87bb17a2bdff115c4b080 failed: Could not back up/etc/puppetlabs/mcollective/server.cfg: getaddrinfo: Name or service not known

- Make sure the filebucket has been configured properly.
- Check site.pp:

Example:

```
# Define filebucket 'main':
filebucket { 'main':
  server => '<PUPPET MASTER'S DNS NAME>',
  path   => false,
}
```

General Troubleshooting

- For additional debug-level messages you can:
  - Set profile to true in the agent's puppet.conf file.
  - Execute a Puppet run with --profile.
  - Execute a Puppet run with --logdest and --debug to log additional details to syslog.

Database Troubleshooting

- Troubleshoot classification
  - In previous version of PE an external node script to reach the PE console node classifier.
  - This external node script was removed.
  - You can curl the console to troubleshoot the node classifier.

Determine what node groups the NC has and what data they contain:

```
curl https://$(hostname -f):4433/classifier-api/v1/groups > classifier_groups.json
    --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem
    --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem
    --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem
```

Database Troubleshooting

Determine what data the NC will generate for a given node name:

```
curl https://$(hostname -f):4433/classifier-api/v1/classified/nodes/<SOME NODE NAME> >
node_classification.json
    --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem
    --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem
    --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem
```

Database Troubleshooting

- PostgreSQL is taking up too much space
  - PostgreSQL should have autovacuum=on set by default.
  - If you're having memory issues from the database growing too large and unwieldy, make sure this setting did not get turned off.
- PostgreSQL buffer memory causes PE install to fail
  - When PE is installed on a machines with large amounts of RAM, PostgreSQL will use more shared buffer memory than is available.
  - This will cause it not to start.
  - Check /var/log/pe-postgresql/pgstartup.log and look for:
    - FATAL: could not create shared memory segment: No space left on device
    - DETAIL: Failed system call was shmget(key=5432001, size=34427584512,03600).

Database Troubleshooting

- Tweak the machine's shmmax and shmall kernel settings before installing PE.
  - shmmax should equal 50% of the total RAM.
  - shmall should be calculated by dividing the new shmmax setting by the PAGE_SIZE.
  - Get the PAGE_SIZE by running getconf PAGE_SIZE.

To set the new kernel settings by run:
sysctl -w kernel.shmmax=<your shmmax calculation>
sysctl -w kernel.shmall=<your shmall calculation>

Optimizing the Databases

The PostgreSQL vacuum command can help optimize your databases.

su - pe-postgres -s /bin/bash -c "vacuumdb -z --verbose <DATABASE NAME>”

You should backup your database nightly.

sudo -u pe-postgres /opt/puppetlabs/server/apps/postgresql/bin/pg_dumpall -c -f <BACKUP_FILE>.sql

- Changing PuppetDB’s parameters:
  - PuppetDB parameters are set in the jetty.ini.
  - jetty.ini is managed by PE.
  - You need to update the setting in the console or they will be overwritten.
- Changing the PuppetDB user/password
  - Stop the pe-puppetdb service.
  - On the database serverUsing psql execute:
    - ALTER USER console PASSWORD '<new password>';
  - Edit /etc/puppetlabs/puppetdb/conf.d/database.ini and update the password.
  - Start the pe-puppetdb service.


Linux Academy
System Administration
Using Puppet Certification
Reporting

Reporting

- Puppet creates a report on actions as well as the infrastructure when it applies the catalog during a Puppet run.
- You can create and use report processors to get information or alerts from the reports.
- How it works:
  - Puppet agent sends its report to the Puppet master for processing.
  - Report processor plugins handle received reports.
  - Each processor typically does one of two things with the report:
    - It sends some of the report data to another service, such as PuppetDB, that can collate it.
    - It triggers alerts on another service if the data matches a specified condition, such as a failed run.
    - That external service can then provide a way to view the processed report.

Reporting

- Configuration:
  - Reporting is enable by default.
  - It's configuration can be change in the puppet.conf file on the Puppet master server.
  - Multiple report processors can be configure and need to be separated by commas.
- Report data:
  - Metadata about the node, its environment and Puppet version, and the catalog used in the run.
  - The status of every resource.
  - Actions that Puppet took during the run, also called events.
  - Log messages that were generated during the run.
  - Metrics about the run, such as its duration and how many resources were in a given state.

Reporting

- 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.

Reporting

- Information found on reports:
  - **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.

Reporting

- Built in report processors:
  - **http:** Send reports to https/http.
  - **log:** Send logs to local syslog.
  - **store:** Stores reports in yaml format in the location specified in the reportdir setting. Default.
- Report processors from the Forge:
  - **tagmail:** send specific reports to specific email addresses.

Reporting

- Plugin report processors
  - IRC
  - Twitter
  - Jabber
  - Hipchat
  - Growl
  - Campfire
  - PagerDuty
  - Etc.

System Administration Using Puppet Certification

Removing Nodes

Removing Nodes from PE

- You will need to do the following steps to remove a node from Puppet Enterprise:
  - Deactivate the node in PuppetDB.
  - Delete the Puppet master's information cache for the node.
- Frees up the license that the node was using.
- Allows you to re-use the hostname for a new node.
- On the agent node:
  - Stop the agent service.

Removing Nodes from PE

- You will need to do the following step to remove a node from Puppet Enterprise:
  - On the Puppet master:
    - Purge the node.
      - puppet node purge <CERTNAME>
    - Run puppet agent -t.
    - Restart Puppet
      - service pe-puppetserver restart
- If the deactivated node still shows up, stop MCollective.
  - On the agent node
    - service mcollective stop
    - On the agent node delete /etc/puppetlabs/mcollective/ssl/clients.


Linux Academy
System Administration
Using Puppet Certification
Puppet Orchestrator Overview

Overview

- The Puppet orchestrator is a set of interactive command line tools that give you the ability to control the rollout of configuration changes when and how you want them.
- Tools:
  - puppet job
    - Allows you to manage and enforce the order of Puppet agent runs across an environment.
    - Enforces the order of agent runs by instantiating an application model and assigning nodes to application components.
  - puppet app
    - Lets you view the application models and application instances written and stored on the Puppet master.
    - Lets you see what is available to include in an orchestration run.

Overview

- You can control when Puppet runs and where node catalogs are applied.
- You no longer need to wait on arbitrary run times to update your nodes.
- Orchestrator workflow
  - Write Puppet code to be uses with Puppet Application Orchestration.
  - puppet parser validate command to validate.
  - puppet app show command to validate that your application or application instances looks correct.
  - puppet job plan command to show applications or application instances and the node run order that would be included in a job.
  - puppet job run command to enforce change on your infrastructure and configure your application.
    - Run the job with the --noop
  - puppet job show command to review details about the run.


Linux Academy
System Administration
Using Puppet Certification
Exported Resources

Exported Resources

- Exported resources require catalog storage and searching to be enabled on your Puppet master.
- Formerly known as “storeconfigs”.
- Both the catalog storage and the searching (among other features) are provided by PuppetDB.
- Exported resource declaration specifies a desired state for a resource.
- It does not manage the resource on the target system.
- Publishes the resource for use by other nodes.
- Any node can then collect the exported resource and manage its own copy of it.

Exported Resources

- Purpose
  - Exported resources allow the Puppet compiler to share information among nodes by combining information from multiple nodes' catalogs.
  - This helps you manage things that rely on nodes knowing the states or activity of other nodes.

Exported Resources

Syntax:

```
class <CLASS NAME> {
  # Declare:
  @@<RESOURCE BEING EXPORTED> { <TITLE>:
    <ATTRIBUTE> => <VALUE>,
  }
  # Collect:
  <REFERENCE RESOURCE BEING EXPORTED> <<| |>>
}
```

Exported Resources

Example:

```
class ssh {
  # Declare:
  @@sshkey { $::hostname:
    type => dsa,
    key  => $::sshdsakey,
  }
  # Collect:
  Sshkey <<| |>>
}
```

Exported Resources

- Declaring an exported resource
  - To declare an exported resource, prepend @@ (a double “at” sign) to the resource type of a standard resource declaration:

Syntax:

```
@@<RESOUCE TYPE> { <TITLE>:
  <ATTRIBUTE> => <VALUE>,
}
```


Linux Academy
System Administration
Using Puppet Certification
Puppet Professional Certification

Puppet Professional Certification

- About the Puppet certification:
  - https://puppet.com/content/puppet-certification
- Pearson VUE:
  - http://pearsonvue.com/puppet/
- The exam costs $200.00 per attempt.
- You will need to bring two forms of identification.
- You will have 90 minutes to complete the exam.

Puppet Professional Certification

- If you fail the first attempt, you will have a 14 day wait.
- If you fail the second attempt, you will have a 90 day wait.
- If you fail the third attempt, you will need to contact Puppet to request a forth attempt.
- Puppet Professional 2016 Practice Exam:
  - https://puppet.com/support-services/certification/2016-professional-practice-exam

Puppet Professional Certification

- Linux Academy Practice exam “mimics” a real exam but does not contain the same questions or exact same format due 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.