# Definitions

* Component modules: Normal modules that manage one particular technology. (For example, puppetlabs/apache.)
* FQDN: Fully qualified domain name.
* Idempotence: The property of certain operations in mathematics and computer science, that can be applied multiple times with changing the result beyond the initial application. Catalog can be applied multiple times without causing issue.
* 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.

# Puppet installation

* Install the Puppet Master: ./puppet-enterprise-installer
* Install the Puppet Agent: curl -k https://:8140/packages/ current/install.bash | sudo bash

### Puppet Installation Flags

* -c : Use a pe.conf file to configure the Puppet server.
* -D : Displays debugging information.
* -h : Display help
* -q : Run in quiet mode; the installation process is not displayed. Requires answer file.
* -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.

Found in /etc/puppetlabs/enterprise/conf.d.

Sample pe.conf file:

{

"console\_admin\_password": "password",

"puppet\_enterprise::puppet\_master\_host": "",

"pe\_install::puppet\_master\_dnsaltnames": [

"puppet"

]

}

### Installation Directories

* Puppet Enterprise configuration files are installed in /etc/puppetlabs/puppet for \*nix nodes and \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 nodes certificate infrastructure (all nodes) /etc/puppetlabs/puppet/ssl

### Puppet Enterprise Logs

All Puppet Enterprise logs can be found in /var/log/puppetlabs.

* 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

### Puppet Ports

* 3000: Used for the web-based installer of the Puppet Master.
* 8140: The port that the Puppet Master and agent communicate on.
* 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.
* 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.conf

The puppet.conf file is located in /etc/puppetlabs/puppet.

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

* Config sections
  + main is the global section used by all commands and services. It can be overridden by the other sections.
  + master is used by the Puppet master service and the Puppet cert command.
  + agent is used by the Puppet agent service.
  + user is used by the Puppet apply command
* 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 = master.vagrant.vm

server = master.vagrant.vm

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 = master.vagrant.vm

always\_cache\_features = true

Sample puppet.conf for an agent node.

[main] server = master.vagrant.vm

certname = agent1.vagrant.vm

* Basic settings
  + always\_retry\_plugins: Affects how we cache attempts to load Puppet resource types and features.
  + basemodulepath: The search path for global modules. Should be specified as a list of directories separated by the system path separator character.
  + Default: $codedir/modules:/opt/puppetlabs/puppet/modules
  + ca\_server: The server to use for certificate authority requests.
  + certname: The name to use when handling certificates.
  + dns\_alt\_names: A list of hostnames the server is allowed to use when acting as the Puppet master. The hostname that an agent uses must be included this list or the agent will fail connecting to master. The hostname can also live in the certname setting.
  + environment: Defaults to production, is the environment to request but can be overridden by masters ENC (External Node Classifier).
  + environmentpath: A search path for directory environments, as a list of directories separated by the system path separator character.
  + manifest: The entry-point manifest for puppet master. This can be one file or a directory of manifests to be evaluated in alphabetical order. Puppet manages this path as a directory if one exists or if the path ends with a / or .
  + reports: The list of report handlers to use. When using multiple report handlers, their names should be comma-separated, with whitespace allowed. (For example, reports = http, log, store.)
  + http: Send reports via HTTP or HTTPS. This report processor submits reports as POST requests to the address in the reporturl setting. The body of each POST request is the YAML dump of a Puppet::Transaction::Report object, and the Content-Type is set as application/x-yaml.
  + log: Send all received logs to the local log destinations. Usually the log destination is syslog.
  + store: Store the YAML report on disk. Each host sends its report as a YAML dump and this just stores the file on disk, in the reportdir directory.
  + Default: store
    - rundir: The location where Puppet PID files are stored.
    - server: The Puppet master server to which the Puppet agent should connect.
    - ssldir: The location where SSL certs are stored.
    - vardir: The location where Puppet stores growing information.
* Run behavior settings
  + ignoreschedules: Schedules allow you to only execute a resource if it's during a specific time period; this setting can disable that feature that might be used when you are doing an initial setup on a node and everything needs to be executed or enforced the first time around
  + noop: Agent will not do any work only simulate changes and report to the master.
  + postrun\_command: command to run after Puppet command execute
  + prerun\_command: command to run before Puppet command executes
  + priority: The scheduling priority of the process. Valid values are 'high', 'normal', 'low', or 'idle', which are mapped to platform-specific values.
  + report: Whether to send reports after every transaction.
  + runinterval: how often the puppet agent daemon runs
  + tags: Limit the Puppet run to include only resources with certain tags (cool), specific data centers, etc
  + usecacheonfailure: Whether to use the cached configuration when the remote configuration will not compile.
  + waitforcert: Keep trying to run puppet agent if the certificate is not initially available (gives time for the master to sign)

# Resource Abstraction Layer

* Describing/declaring the state of a resource
* Providers enforce the desired state

### Resource Type:

Every resource is managed by a resource type

* a title
* a set of attributes.

### Example

user { 'username':

ensure => present,

uid => '102',

gid => 'wheel',

shell => '/bin/bash',

home => '/home/username',

managehome => '',

}

Commands

* puppet describe will provide information about resource types within Puppet
* puppet describe -l lists all resource types available
* puppet describe -s <type> gives short information about resource type
* puppet describe <type> gives a long listing information about resource
* puppet resource will describe information about resources already installed on a running node puppet resource <type>
* puppet resource <type> <name>
* puppet agent will send a report to the Puppet master with all facts and information about the node, this is the node object
  + puppet agent will ensure that the agent's private key file is present
  + puppet agent run is started from the node being managed

# Facter

* facter: Returns a list all facts.
* facter : Returns a particular fact.
* facter -p: Allows Facter to load Puppet-specific facts

# Certificate Signing Request (CSR)

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.

### 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

### Deleting SSL Certs on Agent

cp -r /etc/puppetlabs/puppet/ssl/ /etc/puppetlabs/puppet/ssl\_bak/

## Autosigning

* Should only be used when the environment can fully trust 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

# Building Modules and Classes

### Class Structure and Names

* Class names can have:
* Lowercase letters
* Digits
* Underscores

\A[a-z][a-z0-9\_]\*\Z

* Namespace separator use double colons ( :: )

\A([a-z][a-z0-9\_]\*)?(::[a-z][a-z0-9\_]\*)\*\Z

* [Reserved Variable Names] [Reserved Variable Names]: (https://docs.puppet.com/puppet/4.5/lang\_ reserved.html#reserved-variable-names)

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"

}

}

## Module Structure and Names

* Module names can have:
  + Lowercase letters
  + Numbers
  + Underscores
* Should begin with a lowercase letter.
* Module names cannot contain the namespace separator ( :: )
* Modules cannot be nested

<MODULE NAME>

manifests

files

templates

lib

facts.d

examples

spec

functions

types

### Module 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.
* spec/ : Contains spec tests for any plugins in the lib directory.
* functions/ : Contains custom functions written in the Puppet language.
* types/ : Contains type aliases.

# Autoloading

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

# Custom and External Facts

Custom Facts

* Custom facts are snippets of Ruby code on the Puppet master.
* Usually shell commands are issued as part of the fact to return information.
* Executed on the Puppet nodes with the External Facts Plugin Module.
* Custom facts are located in lib/facter.

Example:

# hardware\_platform.rb

Facter.add('hardware\_platform') do

setcode do

Facter::Core::Execution.exc('/bin/uname --hardware-platform')

end

end

Facts distributed using pluginsync

* Enabled in the [main] section of puppet.conf by setting pluginsync=true

## External Facts

External facts provide a way to use arbitrary executables or scripts as facts, or set facts statically with structured data.

In a Module:

<MODULE\_PATH>/<MODULES>/facts.d/

### On Unix/Linux/OS X:

/opt/puppetlabs/facter/facts.d/

/etc/puppetlabs/facter/facts.d/

/etc/facter/facts.d/

### On Windows:

C:\ProgramData\PuppetLabs\facter\facts.d\

C:\Documents and Settings\All Users\Application Data\PuppetLabs\facter\ facts.d\

### STDOUT in the Format:

key1=value1

key2=value2

key3=value3

### Structured Data Facts:

yaml

json

txt

# DSL Overview

Resource Types

Resource types are the basic building blocks of the Puppet DSL

Every resource type has:

* a title
* a set of attributes

<TYPE> { ‘<TITLE>’ :

<ATTRIBUTE> => <VALUE>,

}

Example Resource Types: file

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

Attributes:

* source
* content
* target

Review all the resource types by visiting the [Resource Type Reference](https://puppet.com/docs/puppet/4.5/types/file.html)

# Style Guide

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.
* 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.

Example:

file { '/tmp/foo': … }

## 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.

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

file { "/tmp${file\_name}": … }

"${facts['operatingsystem']} is not supported by ${module\_name}" warning("Class[class\_name'] doesn't work they way you expected it too.")

## Escape Characters and Comments

* Puppet uses backslash as an escape character.
* Escaping as \\ would be "\\\\"
* 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

# 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.

## 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 () { }

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.
* Seriously, dude, don't nest classes or defined types!

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 4.9.
* 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.

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.

Defaults for case statements and selectors.

* Case statements must have default cases.
* Case and selector values must be quoted.

Example:

$file\_mode = $facts['os']['family'] ? {

'Debian' => '0007',

'RedHat' => '0776',

default => '0700',

}

file { '/tmp/readme.txt':

ensure => file,

content => "Hello World\n",

mode => $file\_mode,

}

case $Facts[::operatingsystem] {

'centos': { $version = '1.2.3' }

'debian': { $version = '3.4.5' }

default: { fail("Module ${module\_name} is not supported on ${::operatingsystem}") }

}

More [reference](https://puppet.com/docs/puppet/6.3/style_guide.html)

# 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>]

#### Details:

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.

# 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

## 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 action 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

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 RESOURCES> for <RESOURCES>
* Could not find resource '<OTHER RESOURCES>' for relationship on '<RESOURCES>'.

## Failed Dependencies

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

* notice: <RESOURCES>: Dependency <OTHER RESOURCES> has failures: true
* warning: <RESOURCES>: 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: (<RESOURCES> => <OTHER RESOURCES> => <RESOURCES> )
* Try the --graph option and opening the resulting .dot file in OmniGraffle or GraphViz

# 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

}

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::generic::os

}

### Behaviour

* 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 and or expressions
* Functions that return values

### 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 a elsif clauses.

Syntax:

unless condition { block of code }

Example:

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

### 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 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::generic::os }

}

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 and or expressions
* Functions that return values

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.

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',

}

### 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

### 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.

### 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!”,

}

# Variables and Scope

* 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
* 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

#### Variable Assignment Example:

$variable\_name1 = "value"

#### 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 variables 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 the order they were added in.
* 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.

# Metaparameters

Metaparameters are attributes that all resource type, custom types and defined types have.

Available Metaparameters

* alias
* audit
* before
* consume
* export
* loglevel
* noop
* notify
* require
* schedule
* stage
* subscribe
* tag

#### Example:

file { '/etc/ssh/sshd\_config':

owner => root,

group => root,

alias => 'sshdconfig',

}

service { 'sshd':

subscribe => File['sshdconfig'],

}

# 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.

Example:

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

# function call with lambda:

$values.each |String $value| {

notice { "Value from a lambda code block: ${value}": }

}

# Class Parameters and Defaults

Classes, defined types, and lambdas can all take parameters. •

Which is a way for you to pass external data.

Syntax:

Class <CLASS NAME> (

<DATA TYPE> <PARAMETER NAME> ,

<DATA TYPE> <PARAMETER NAME> = <VALUE> ,

# ...

) {

# ...

}

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 to 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 return a hash.
* Set data\_provider to function in metadata.json.
* 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.

# Puppet 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

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.

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.
* 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 regexp 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.

[Puppet Function list]: (https://docs.puppet.com/puppet/latest/function.html)

# Templates

Loads an ERB template from a module, evaluates it, and returns the resulting value as a string.

A template is referenced by template(<MODULE>/<TEMPLATE FILE>)

template('modulename/motd.erb')

The file is located in <MODULE 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: some value

<% @valuse.each -%>

<% do |values| %> some value <%= value %>

<% end -%>

# 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 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

Syntax:

define name (

<DATA TYPE> <PARAMETER>= <VALUE>,

) {

... puppet code ...

}

Declaring an Instance:

<DEFINED TYPE> { '<TITLE>’:

<ATTRIBUTE> => <VALUE>,

}

#### Example:

define apache::vhost (

Integer $port,

String[1] $docroot,

String $servername = $title,

String[1] $vhost\_name = '\*',

) {

# ...

}

apache::vhost {'mywebsite':

port => 80,

docroot => '/var/www/mywebsite',

}

# 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.

Operators

* ==
* !=
* and
* or

Syntax:

<RESOURCE TYPE> <| <SEARCH EXPRESSION> |>

Example:

User <| groups == ‘admin’ |>

# 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.

## 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.

Syntax:

class <CLASS NAME> {

# Declare: @@ <RESOURCE BEING EXPORTD> { <TITLE> :

<ATTRIBUTE> => <VALUE>,

}

# Collect:

<REFERENCE RESOURCE BEING EXECUTED> <<| |>>

}

Example:

class ssh {

# Declare:

@@sshkey { $::hostname:

type => dsa,

key => $::sshdsakey,

}

# Collect: Sshkey <<| |>>

}

Declaring an Exported Resource

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

Syntax:

@@<RESOURCE TYPE> { <TITLE> :

<ATTRIBUTE> => <VALUE>,

}

# Exported Resources

[puppet-sshkeys]

# Roles and Profiles

## 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.
* There are three ways a profile can get the data it needs to configure component classes:
  + Hardcode it in the profile.
  + Look it up from Hiera.

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

#### Roles Names Example:

role::web

role::jenkins::master

role::jenkins::slave

Example:

class role::web {

include profile::base

include profile::apache

include profile::php

}

# Hiera Overview

Hiera is a key/value datastore 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

* The hiera.yaml file 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 spesific data at the top.
  + Least spesific at the bottom.
* You can use facts in your Hiera lookups.

## Sample 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 pass parameter it will look in hiera data source for the parameter <CLASS NAMESPACE>::parameter
* If not found in hiera data source it will use the default set "default"

## Hiera Lookup Functions

### 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:

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: 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.

# Managing and Deploying Puppet Code

* 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.
* You can r10k to manage your Puppet code instead of Code Manager.
  + You should really 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.

# Set Up and Configuring Code Manager

* Create your own control repo.

wget <https://github.com/puppetlabs/control-repo/archive/production.zip>

yum install unzip -y

unzip production.zip cd production

* Create a control repo in GitHub.
  + Log in to your Github account.
  + Click Repositories.
  + Click the New button.
  + Enter puppet-control for the Repository name.
  + Click Create Repository.
* Initialize your the control repo.
  + Check in code.
  + Add remote repo.
  + Push code.

git init

git remote add origin <URL\_TO\_REPOSITORY>

git commit -am "first commit"

git push origin master

* Create an rsa key for code manager.

mkdir -p /etc/puppetlabs/puppetserver/ssh

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

* Enter the path to where the rsa key will go.

/etc/puppetlabs/puppetserver/ssh/id\_rsa

* Press enter for an empty passphrase
* Make sure ssh is owned by pe-puppet

chmod -R pe-puppet:pe-puppet /etc/puppetlabs/puppetserver/ssh

* Update PE Master node group.
* Add the following parameters to the puppet\_enterprise::profile::master class:
  + code\_manager\_auto\_configure to true
  + update r10k\_remote with the URL to your git repo

update r10k\_private\_key with the path to your rsa key /etc/puppetlabs/puppetserver/ssh/id\_rsa

* Execute a puppet agent -t on the Puppet master server.
* View code manager configuration.

r10k deploy display –fetch

* Create a deploy user.
* Reset the password for your deploy user.
* Add the deploy user to the Code Deployers User role.
* Create a token for your deploy user.

puppet-access login --service-url https://<HOSTNAME OF PUPPET ENTERPRISE CONSOLE>:4433/rbac-api --lifetime 180d

* Deploying your code to the master.

puppet-code deploy --all --wait Git URL Example: git@:puppet/control.git RSA Key Example: "/etc/puppetlabs/puppetserver/ssh/id-control\_repo.rsa"

Git URL Example:

[git@<GIT SERVER>:puppet/control.git](mailto:git@:puppet/control.git)

RSA Key Example: "/etc/puppetlabs/puppetserver/ssh/id-control\_repo.rsa"

# Node Classification

## Node Definition Lookup

Node Definition Lookup Example: webserver01.mylabserver.com

Attempt to match webserver01.mylabserver.com

Attempt to webserver01.mylabserver

Attempt to webserver01

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.

External Node Classifiers

ENCs can co-exist with standard node definitions in site.pp, and the classes declared in each source are effectively merged.

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=console 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

# 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 if 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.

You 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.
* The job with the --noop
* puppet job show command to review details about the run.

# MCollective

## Overview

* Puppet Enterprise includes the MCollective.
* Which is 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

## 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.

## MCollective Components:

* pe-activemq: Service (which runs on the Puppet master server) routes all MCollective-related messages.
* pe-mcollective: Service (which runs on every agent node) listens for authorized commands and invokes actions in response.
* mco command (available to the peadmin user account on the Puppet master server) can issue authorized commands to any number of nodes.

## 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

### The mco command

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

### Using mco help

mco help

mco help <subcommand>

mco <subcommand> --help

#### Synstax:

mco <SUBCOMMAND> <ACTION>

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

Examples:

mco ping

mco rpc rpcutil ping

mco rpc service restart service=puppet

## Host Filters

-W, --with FILTER Combined classes and facts filter

-S, --select FILTER Compound filter combining facts and classes

-F, --wf, --with-fact fact=val Match hosts with a certain fact

-C, --wc, --with-class CLASS Match hosts with a certain config management class

-A, --wa, --with-agent AGENT Match hosts with a certain agent

-I, --wi, --with-identity IDENT Match hosts with a certain configured identity

# Troubleshooting

## 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 false, }
* Make sure the agent can reach the DNS name that is configured in puppet.conf.
* Check that the pe-puppetserver service is running.
* 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 the filebucket:

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} e33a27e4b9a 87bb17a2bdff115c4b080 failed: Could not back up/etc/puppetlabs/ mcollective/se rver.cfg: getaddrinfo: Name or service not known

Example:

# Define filebucket 'main':

filebucket { 'main':

server => ‘<PUPPET MASTER DNS NAME>’ ,

path => false,

}

## General Troubleshooting

Use --profile or add profile to true in the agent's puppet.conf file.

Use --logdest and --debug to log additional details to syslog.

## Database Troubleshooting

* Troubleshoot classification
* 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/.pem --key /etc/puppetlabs/puppet/ssl/private\_keys/.pem

Determine What Data the NC Will Generate for a Given Node Name:

curl https://$(hostname -f):4433/classifier-api/v1/classified/ nodes/ > 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

### Determine What Data the NC Will Generate for a Given Node Name:

curl https://$(hostname -f):4433/classifier-api/v1/classified/ nodes/<SOME NODENAME> > 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

PostgreSQL is taking up too much space

PostgreSQL should have autovacuum=on set by default.

PostgreSQL buffer memory causes PE install to fail

### Check /var/log/pe-postgresql/pgstartup.log

FATAL: could not create shared memory segment: No space left on deviceDETAIL: Failed system call was shmget(key=5432001, size=34427584512,03600).

Tweaking 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 SHMAX CALCULATION>

sysctl -w kernel.shmall= <YOUR SHMALL CALCULATION>

## Optimizing the Databases

* 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 server, using 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.

### Vacuuming PostgreSQL

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

### Backing Up PostgreSQL

sudo -u pe-postgres /opt/puppetlabs/server/apps/postgresql/bin/pg\_ dumpall -c -f <BACKUP FILE>.sql

# 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 aelempted 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.

Built in report processors:

* http: send reports to https/http.
* log: Send logs to local syslog
* store: store reports in yaml form in the location specified in the reportdir setting

Report processors you download:

* tagmail: send specific reports to specific email addresses.

Puppet Enterprise Roles Based Access Control

[RBAC Permissions](https://puppet.com/docs/pe/2016.2/rbac_permissions.html)

# Removing Nodes

You will need to do the following step to remove a node from Puppet Enterprise:

* Deactivates the node in PuppetDB.
* Deletes 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:

service puppet stop

On the Puppet Master:

puppet node purge <CERTNAME>

puppet agent -t

service pe-puppetserver restart

If the deactivated node still shows up, stop MCollective.

On the Agent Node:

service mcollective stop /etc/puppetlabs/mcollective/ssl/clients.

# Checking Values of Settings

puppet master --configprint <CONFIG NAME>

puppet config print <CONFIG NAME>

puppet config print <CONFIG NAME> --section <SECTION NAME>

# Puppet Resource Command

puppet resource <RESOURCE NAME>

# Puppet Professional Certification

[About the Puppet certification](https://puppet.com/support-services/certification/professional-overview-2018)

[Exam](https://ondemand.questionmark.com/home/405096/user)

[Puppet Professional Practice Exam](https://puppet.com/support-services/certification/puppet-professional-practice-exam)

# Code Repositories for this Course

Roles Repository : https://github.com/linuxacademy/content-ppt-204-role

Profiles Repository : https://github.com/linuxacademy/content-ppt-204-profile

Control Repository : https://github.com/linuxacademy/content-ppt-204-control

NTP Repository : https://github.com/linuxacademy/content-ppt-204-ntp

SSH Repository : https://github.com/linuxacademy/content-ppt-204-ssh

Nginx Repository : https://github.com/linuxacademy/content-ppt-204-nginx