NobleProg

Puppet Fundamentals

This course is intended for organisations already using Puppet or Systems Administrators looking to implement Puppet as a new solution.

The course is delivered over 3 days and involves a mixture of both presentations and exercises. On completion you will be able to deploy a basic system configuration for Puppet in a Master-Client environment.

Introduction: Who We Are

Your Trainer is: Andy Singleton

A Linux DevOps trainer and consultant specialising in Linux administration, Configuration Management, Docker and OpenStack.

Around 20 years in IT, with professional certifications including
- Red Hat RHCSA, RHCE, RHCDS, and RHCA
- PuppetLabs Puppet Certification

About NobleProg

NobleProg are your training and consultancy provider for areas including Management, IT, Statistics, Programming, and Artificial Intelligence

Puppet Fundamentals

Day 1	Day 2	Day 3
Introduction	Module Structure	General Architecture
About Puppet	PuppetForge	Multi-master Puppet
Agent and Master	Classifying nodes	Foreman
Community and Enterprise	Hiera	Cobbler
Reporting	Git	Best Practise
Resources	Module Development	Case Study
Resource Relationships	Module Testing	
Variables	Deploying Code	
Facter	Master vs Masterless	
Conditionals		
Templating		
Defined Types		

Introduction: Training Environment

User instance

1 CPU, 523 MB Ram, 8GB Storage
Red Hat / CentOS or Ubuntu

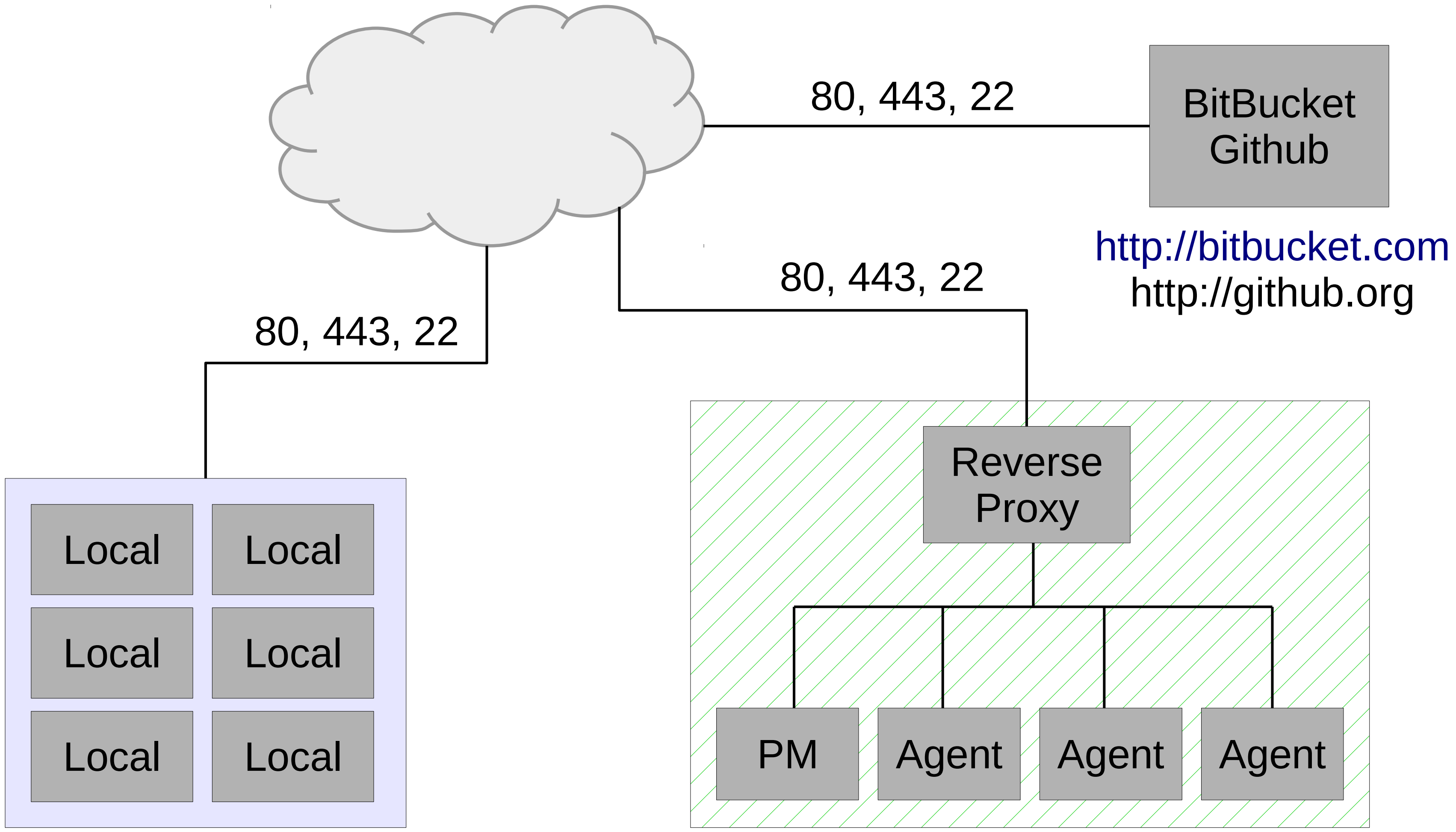
SSH access: Port 22

PuppetMaster instance

2 CPU, 4MB Ram, 20GB Storage
Red Hat / CentOS or Ubuntu

SSH access: Port 22
Web Access: Port 80 / 443

Introduction: Training Environment



https://portal.cloud2class.com
https://resource.cloud2class.com

Introduction: Workshops

The general aim of the workshop will be shown here

Workshops

- A set of practical tasks
- Aimed at achieving a specific outcome.
- An opportunity for discussion
- Other workshops will frequently depend on each other

If you think you cannot successfully complete a section, ask one of our instructors to help you out.

If at any point you accidentally damage your training instance, notify us and we will reset it for you

Workshop : Registration

Log on to the cloud2class platform. This will create user accounts for your instance(s), and allow access to other training resources.

Log on to the student training portal

- Browse to
- Enter yourhttps://portal.cloud2class.com first and last names
- Enter the lesson security details

Security Details

*Word: *** monkey*

*Number: ***number*

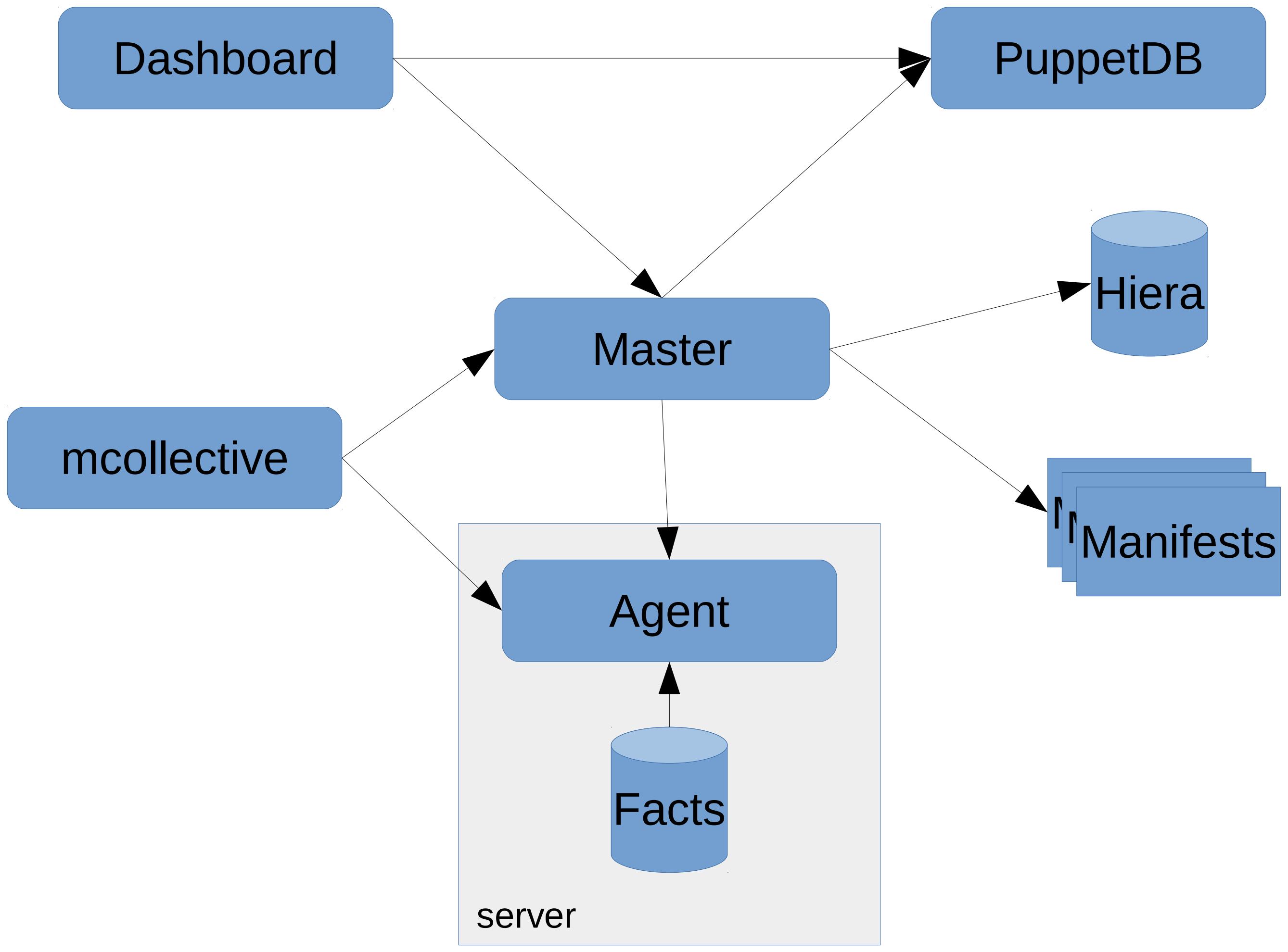
Introduction: Puppet

What it IS	What it is NOT
Configuration Management	A panacea
Powerful	Simple to configure
Widely used	A filestore
Scalable to thousands of nodes	The only option: Chef, Ansible
Used to define an end-state	Capable of operating alone
A complete replacement for system administration	A complete replacement for system administration

Puppet: Components

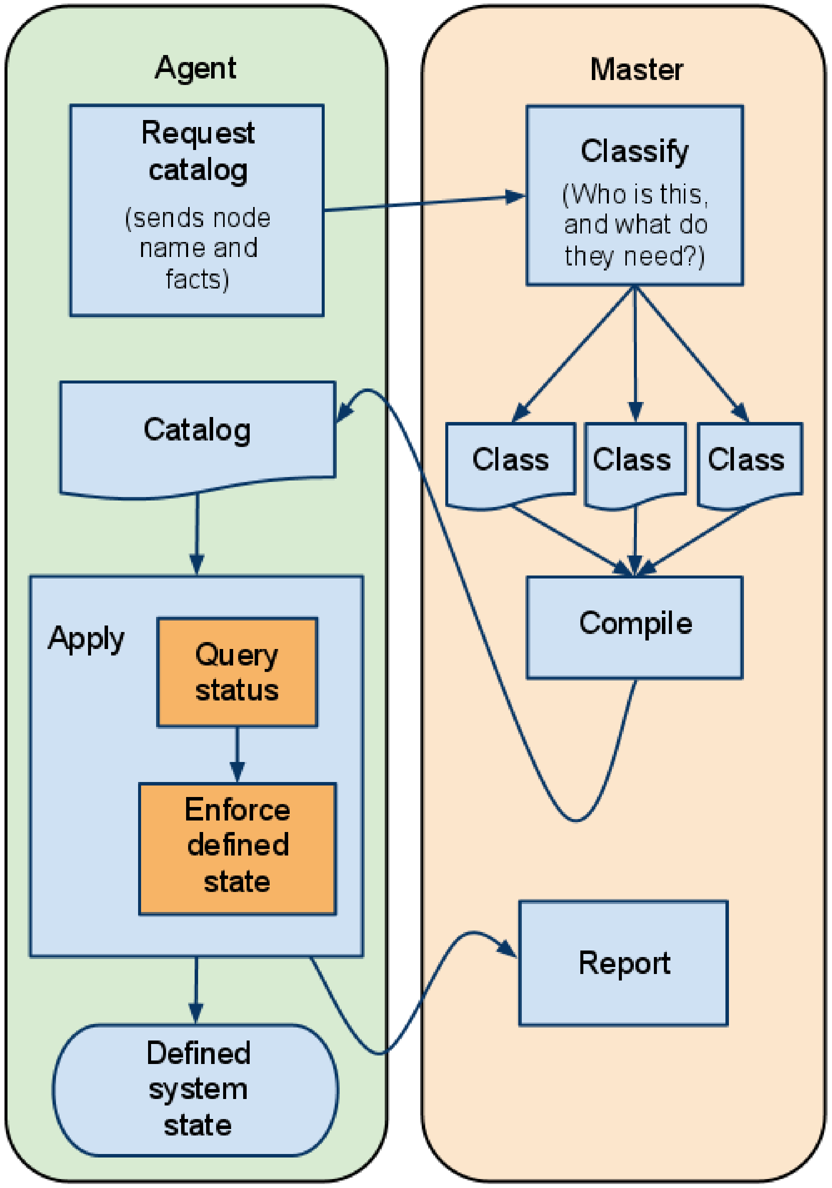
Component	Function
Agent	Enacts system changes
Master	Compiles manifests for agents. Distributes your server configuration
PuppetDB	Stores node state, run logs, and facts
Puppet Dashboard	Monitor and report on your nodes
Mcollective	Orchestration tool
Facter	System information collection
Hiera	Hierarchical configuration database

Puppet: Components



Agent and Master : Index

Agent and Master : Interaction



Agent and Master : Running the Agent

Puppet agent:

Listens on port 8139
Logs to syslog
Certs are in /var/lib/puppet/ssl

Package requirements:

Puppet
Facter
Hiera

Starting the Puppet Agent.

```
sudo puppet agent –enable (Ubuntu)
sudo puppet agent
sudo service puppet start
```

Debugging / Viewing the output

```
sudo puppet agent -v -t
```

Agent and Master : Locating the puppet-master

How do we find a Puppetmaster?

- DNS - “puppet”
- DNS – srv (experimental)
- Puppet.conf

```
[agent]
    server = np-puppet.openstacklocal
```

- Command line

```
sudo service puppet agent --server
```

Agent and Master : Running the Master

Puppetmaster:

Listens on port 8140
Logs to syslog
Certs are in /etc/puppet/ssl

Package requirements:

Puppet
Facter
Hiera
deep_merge (gem)

Starting the Puppetmaster

```
sudo puppet master
sudo service puppetmaster start
```

Debugging / Viewing the output

```
sudo puppet master -v -d --no-daemonize
```

Agent and Master : Signing Certificates

How to ensure we configure the right machines

- Autosign – Do Not Use!

```
[main]
    autosign = true
```

- Certificate management : Puppet Cert

Node :

```
puppet agent -v -t --waitforcert 60
```

Puppet-master :

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

Agent and Master : Configuration

```
# Settings in [main] are used if a more specific section doesn't set a value.
[main]
    certname = puppetmaster01.example.com
    logdir = /var/log/pe-puppet
    rundir = /var/run/pe-puppet
    Basemodulepath = /etc/puppet/modules
    server = puppet.example.com
    user  = pe-puppet
    group = pe-puppet
    archive_files = true
    archive_file_server = puppet.example.com

# This section is used by the Puppet master and Puppet cert applications.
[master]
    certname = puppetmaster01.example.com
    dns_alt_names = puppetmaster01,puppetmaster01.example.com,puppet,puppet.example.com
    ca_name = 'Puppet CA generated on puppetmaster01.example.com at 2013-08-09 19:11:11 +0000'
    reports = http,puppetdb
    reporturl = https://localhost:443/reports/upload
    node_terminus = exec
    external_nodes = /etc/puppetlabs/puppet-dashboard/external_node
    ssl_client_header = SSL_CLIENT_S_DN
    ssl_client_verify_header = SSL_CLIENT_VERIFY
    storeconfigs_backend = puppetdb
    storeconfigs = true
    autosign = true

# This section is used by the Puppet agent application.
[agent]
    report = true
    classfile = $vardir/classes.txt
    localconfig = $vardir/localconfig
    graph = true
    pluginsync = true
    environment = production
```

Agent and Master : Webservers

WEBrick

- Default webserver for Puppet-master
- Does not scale
- Ideal for research or proof-of-concept

Passenger

- Allows running of Rails or Rack applications within a webserver
- Scalable
- Fast

Workshop : Configuring a Puppetmaster

Puppet has already been installed on the training machines.
Configure your machines to use themselves as the puppetmaster,
And to use Passenger as the webserver

Configure the puppet-master

- Install the puppet-server package
- Run the puppet-master in verbose mode

Configure the client

- Configure the puppet agent to use your hostname
- Run the agent in verbose mode

Install Passenger (optional)

- Download the training repo:
  git clone https://github.com/andysingleton/training-repo.git
- Configure puppet master to use Passenger as the webserver

Community vs Enterprise : Index

Community vs Enterprise : Features

Community

- Lower Cost
- Variety of Node Classifiers (Foreman, Razor, Spacewalk)

Community

- Cloud Provisioning
- Unified Installer
- Event Inspector
- Mcollective / “Live Management”

Community vs Enterprise : Dashboard

Community

- No interaction with Mcollective
- Only partially maintained

Enterprise

- Behaves as a unit
- Better component integration
- Fully functioning dashboard
- More complicated under the hood

Community vs Enterprise : Support

Community

- Forums and User groups
- Commercial Support (**NOT** from PuppetLabs)

Enterprise

- Forums and User groups
- Commercial Support (from PuppetLabs)

Reporting : Index

Reporting : Configuring Reports

/etc/puppet/puppet.conf

```
[master]
    reports = store , puppetdb , http
    reportdir = /var/lib/puppet/reports
    reporturl = http://np-puppet:3000/reports/upload
```

- store : yaml files are kept in “reportdir” *(/var/lib/puppet/reports/)*
- puppetdb : */etc/puppet/puppetdb.conf* determines target
- http : Pre-process them as html and forward to “*reporturl*”
- log : Use local log destinations (*syslog*)
- rrdgraph : generate RRD graphs to the “*rrddir*”
- tagmail : email logs based on tags in the messages

Workshop : Reporting Tools

Run the puppet agent manually against the shared puppetmaster.
You should see the result on the various reporting tools

Run puppet against another master

- Run the Puppet agent manually against np-puppet.openstacklocal
  puppet agent -v -t --server=np-puppet.openstacklocal --ssldir=/var/tmp/puppet

View reporting tools

- PuppetDB
  https://resource.cloud2class.com/989/puppetdb
- Puppet Dashboard (community)
  https://resource.cloud2class.com/989/dashboard
- PuppetBoard
  https://resource.cloud2class.com/989/puppetboard

Language : Index

Language : Overview

Resource Declarations contain :

- A Resource Type, many Attributes, Functions, Conditionals, ...

Classes contain :

- Class Definitions, Class Declarations, Resource Declarations, Defined Types, Variable assignments, Functions, Conditionals, ...

Manifests contain :

- Class Definitions, Class Declarations, Resource Declarations, Defined Types, Variable assignments, Functions, Conditionals, ...

Modules contain :

- Manifests, Templates, Files, ...

Language : Resources

- Describes an aspect of the system
- Resources are Declared – This includes them in the catalog
- The block of code is called a Resource Declaration
- Typically in Classes or Defined Types (which are then declared)

A Class containing a Resource:

```
class configure_ssh {
    file { “/etc/ssh/sshd_config”:
      ensure       => 'present'
      owner        => 'root',
      group        => 'root',
      mode         => '0600',
      source       => 'puppet:///modules/configure_ssh/sshd_config',
    }
  }
```

Language : Resources

Common Resource Types

- File:
  Covers any file, directory, link, device-file or symlink

- Package and Service:
  Used to deploy software and manage the state of services

- User:
  Create and manage the presence/absence of users

- Exec
  Used to perform manual tasks: DO NOT USE IT!

  https://docs.puppetlabs.com/references/latest/type.html

Language : Resource Relationships

Before

Require

Notify

Subscribe

Chaining Arrows (-> and ~>)

Language : Resource Relationships

before

- Apply this resource first

require

- Apply another resource first

```
class nginx {
  package { 'nginx':
    ...
    before => Service['nginx'],
  }

  service { 'nginx':
    ...
    require => Package['nginx'],
  }
}

Package['nginx'] -> Service['nginx']
```

Language : Resource Relationships

notify

- Tell another resource when we have been changed

subscribe

- Update our resource when another resource has changed

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

  service { 'nginx':
    ...
    subscribe => File['/etc/nginx/nginx.conf'],
  }
}

File['/etc/nginx/nginx.conf'] ~> Service['nginx']
```

Language : Class Definitions

```
class 'webusers' (
    ensure =  $ensure,
) {
    user { “sarah”:
        ensure         => $ensure,
        uid            => 1001,
        gid            => 1001,
        firstname      => 'Sarah',
        lastname       => 'Conner',
        home           => $name,
        shell          => '/bin/bash',
        ssh-key        => 'rsa-......',
        managehome => true,
    }
}
```

Language : Loading Classes

Autoloading

- When a class or resource is declared, puppet uses its full name to find it in the available modules.
- If it can't find the class, you can't use it

Naming rules

- Base classes should be defined in init.pp
- Other classes should have the format *module::class*, and be defined in their own .pp file
- Further namespaced classes will be searched for in sub-directories and their own .pp file

modules/apache/manifests/init.pp

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

modules/apache/manifests/vhost.pp

```
class apache::vhost {
  ...
}
```

modules/apache/manifests/vhost/config.pp

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

Language : Definition vs Declaration

Defining a Class : Making it available

```
class my_own_class {
  Resource declarations go here
}

Class my_other_class (
  $my_parameter=”my default value”,
) {
  Resource declarations go here
}
```

Declaring a Class : Making it happen

```
  class { myclass:
    my_parameter => “/etc/ssh/”
}

include myclass
hiera_include ('myclass')
contain myclass
require myclass
```

Workshop : Creating a simple class

Put together a simple class containing a resource definition to deploy an executable when the class is declared and to start a service.

1 : Checkout the training repo

- *git clone https://github.com/andysingleton/training-repo.git*

2 : Create a Module

- Create a directory under */etc/puppet/modules*
  (The convention is alphanumerics, and underscores)
- Create the following directories under your module directory

  ./manifests
  ./templates
  ./files

Workshop : Creating a simple class

Put together a simple class containing a resource definition to deploy an executable when the class is declared and to start a service.

3 : Create a Manifest

- Create *init.pp* under “manifests” using your editor of choice

4 : Create a Class that...

- Deploys *turret.py* from files to */usr/sbin/turret.py*
- Deploys *turret-sysv* from files to */etc/init/turret*
- Starts the turret service after its files have been deployed

5 : “Classify” your node

- Add the name of your class to /etc/puppet/manifests/site.pp
- Run puppet on your local node

Language II : Index

Language II : Variables

Important variable concepts :

- Assignment : $a = 3
- Resolution
- Interpolation : '$variable' vs “$variable”
- Scope

Language II : Variables

Boolean :

- *$myvar = true*

String :

- *$myvar = 'somevalue'*

List :

- *$myvar = ['first' , 'second']*

Hash :

- *$myvar = { 'name' => 'consuela' }*

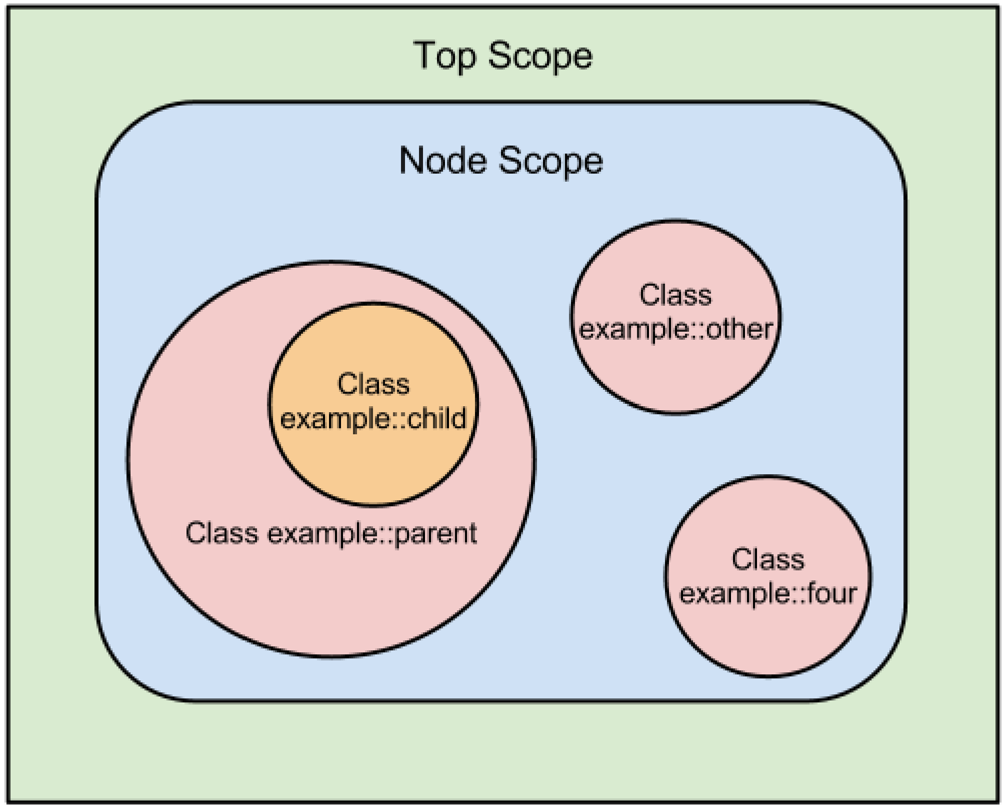
Nested :

- *$myvar = { ['name' => 'consuela' , 'shell' => '/bin/bash'] }*

```
class manage_ssh (
    $permitrootlogin = 'no',
    $ensure = 'running',
    $enable = true,
) {
  case $osfamily {
    'Debian': { $servicename = 'ssh' }
    'RedHat': { $servicename='sshd' }
  }
  service { $servicename:
    ensure     => $ensure,
    enable     => $enable,
    hasrestart => true,
    hasstatus  => true,
    require    => Package['openssh-server']
  }
```

Language II : Scoping

“Variables in a given scope are only available within that scope and its children, and any local scope can override the variables it receives from its parents.”



Language II : Scoping

Top scope

- Variables defined outside of a class, type, or node definition at top scope. e.g. in site.pp

Node scope

- Variables defined inside a node definition are available everywhere for that specific node

Local scope

- Variables defined inside a class or defined type are in local scope.
- Available only within the scope and its children.

/etc/puppet/manifests/site.pp

```
$purple = “top scope”
$green = [“top scope list”]

node 'mynode.example.com' {
  $yellow = 'node local scope'
}

class core {
  $blue = “local scope”
  $green += ['appended data']
}

class a_n_other {
  $blue = “another local scope”
  $purple = “overridden local scope”
}
```

Language II : Scoping

Global Qualified Name

- A method to access any variable anywhere:

  $apache::params::confdir

Language II : Facter

Core Facts

- Node infromation always available from facter

Custom Facts

- Written in Ruby
- Distributed manually or from a module (*[module]/lib/facter/[fact].rb*)
- Loaded in puppet.conf using '*pluginsync = true*', and '*facter -p*'

External Facts

- Arbitrary executables, or static files (e.g. yaml)
- Distributed from a module (*[module]/facts.d/[your binary]*)
- Loaded in puppet.conf using '*pluginsync = true*', and '*facter -p*'
- Manually drop into */etc/facter/facts.d/*

Language II : Facter

- Facter collects system information in the form of “Facts”.

  facter -p

- Facts can be resolved in the same manner as any variable

  if $fact_name == 'ok' {

  *ensure => $facts['fact_name']   *requires trusted_node_data = true*

- Can also be referenced as $::fact_name for clarity

Language II : Conditionals

if / elsif / else

- If expression evaluates to true, execute code block

unless

- If expression evaluates to false, execute code block

case

- Execute code block relating to matched value

selector

- Return a value instead of executing a code block

```
class deploy_ssh {
  if $osfamily == 'Debian' {
    $ssh_service = 'ssh'
  }
  elsif $osfamily == 'RedHat' {
    $ssh_service = 'sshd'
  }
  else {
    $ssh_service = 'ssh'
  }

  case $osfamily {
    'Debian': { $ssh_service = 'ssh' }
    /RedHat/: { $ssh_service ='sshd' }
    default:  { $ssh_service = 'ssh', }
  }

  $ssh_service = $osfamily ? {
    'Debian'  => 'ssh',
    'RedHat' => 'sshd',
    Default  => 'ssh',
  }
```

Workshop : Extending Facter

We are going to create an external fact that checks the current state of our Turret service, then add some conditional logic based on it.

1 : Create an external executable fact

- Either use the training-repo “*check_turret.sh*” script or create your own; We want to know if the service is running or not
- Check “*facter*” responds correctly

2 : Populate /var/www/html/index.html with a status page

- Extend your class to change the content of index.html
- If the turret service is up, provide a web-page stating that everything is fine
- If the service is down, provide a page saying the service is down

Language III : Index

Language III : Templates

Templates are an easy way to distribute similar files
to different nodes.

What is a template :

- Plain text plus embedded Ruby code
- A “.erb” file in *[module]/templates*

How to use it :

- With the “template” function from within a File resource
- Some knowledge of Ruby is good

Benefits :

- Can contain complex logic
- Keeps the filestore clean
- As simple or as complex as you want them to be

Language III : Templates

Ruby Expression

- Replace the content of the tags with the output of the code
- This can include variables from your manifests

  PermitRootLogin <%= @permitroot -%>

Ruby Code

- Execute any code. It will not be replaced by any value in the final content.

  PermitRootLogin <% if @mycondition == false -%>

Comments

- Suppressed in the final output

  <%# We include this loop to workaround bug #659 %>

<%-

- Supress leading whitespace

-%>

- Supress following linebreak

Language III : Templates

Scoped Variables :

- All variables in your current scope canbe used wth an @
  PermitRootLogin <%= @permitroot -%>

Out of scope variables :

- Any variable can be referenced with the scope.lookupvar method
  <%= scope.lookupvar('apache::user') %>

Language III : Templates

modules/ntp/manifests/config.pp (snippet)

```
 file { $config:
   ensure  => file,
   owner   => 0,
   group   => 0,
   mode    => '0644',
   content => template($config_template),
 }
```

modules/templates/authorized_keys.erb

```
<% [@servers].flatten.each do |server| -%>
server <%= server %><% if @preferred_servers.include?(server) -%> prefer<% end %>
<% end -%>

<% if scope.lookupvar('::is_virtual') == "false" or @udlc -%>
# Undisciplined Local Clock. This is a fake driver intended for backup
# and when no outside source of synchronized time is available.
server  127.127.1.0
fudge   127.127.1.0 stratum 10
restrict 127.127.1.0
<% end -%>
```

Language III : Templates

modules/usermgmt/manifests/user.pp (snippet)

```
file { "${home}/.ssh/authorized_keys":
  ensure  => $ensure,
  owner   => $name,
  group   => $name,
  mode    => '0600',
  content => template("${module_name}/authorized_keys.erb"),
  require => File["${home}/.ssh"],
}
```

modules/templates/authorized_keys.erb

```
<% @ssh_pub_keys.sort.each do |key| -%>
<%= key %>
<% end -%>
```

Language III : Templates

modules/usermgmt/manifests/user.pp (snippet)

```
file { "${home}/.ssh/authorized_keys":
  ensure  => $ensure,
  owner   => $name,
  group   => $name,
  mode    => '0600',
  content => template("${module_name}/authorized_keys.erb"),
  require => File["${home}/.ssh"],
}
```

modules/templates/authorized_keys.erb

```
<% @ssh_pub_keys.sort.each do |key| -%>
<%= key %>
<% end -%>
```

Language : Defined Types

Defined Types are repeatable pieces of configuration
They are composed of other resources, and behave exactly as any other resource.

Creating a Defined Type

- The define keyword
- A name
- A list of parameters
- A block of code

Language : Defined Types

```
define planfile ($user = $title, $content) {
    file {"/home/${user}/.plan":
      ensure  => file,
      content => $content,
      mode    => 0644,
      owner   => $user,
      require => User[$user],
    }
  }

  user {'nick':
    ensure     => present,
    managehome => true,
    uid        => 517,
  }
  planfile {'nick':
    content => "Some new content",
  }
```

Language : Defined Types

Defined Types are repeatable pieces of configuration
They are composed of other resources, and behave exactly as any other resource.

Creating a Defined Type

- The resources inside the type must depend on parameters