Puppet

Puppet is an open-source IT automation tool. The Puppet Domain Specific Language (DSL) is a Ruby-based coding language that provides a precise and adaptable way to describe a desired state for each machine in your infrastructure. Once you've described a desired state, Puppet does the work to bring your systems in line and keep them there.

Puppet Enterprise (PE) is a complete configuration management platform, with an optimized set of components proven to work well together. It combines a version of open source Puppet (including a preconfigured production-grade Puppet master stack), with MCollective, PuppetDB, Hiera, and more than 40 other open source projects that Puppet Labs has integrated, certified, performance-tuned, and security-hardened to make a complete solution for automating mission-critical enterprise infrastructure.

Contents

[Configure Graphite class 2](#_Toc484515228)

[Assign puppet resources 4](#_Toc484515229)

[Puppet manifests 6](#_Toc484515230)

[Cowsay class 6](#_Toc484515231)

[Fortune class 7](#_Toc484515232)

[Main class 8](#_Toc484515233)

[Puppet module 10](#_Toc484515234)

[Creating a module 10](#_Toc484515235)

[Using existing module – NTP 13](#_Toc484515236)

[Using puppet to install and configure a server - MySQL 16](#_Toc484515237)

[Puppet variables and parameters 19](#_Toc484515238)

[Variables 19](#_Toc484515239)

[Class parameter 19](#_Toc484515240)

[Managing accounts in nodes with conditional statements 22](#_Toc484515241)

[If statement 22](#_Toc484515242)

[Unless keyword 23](#_Toc484515243)

[Case keyword 23](#_Toc484515244)

[Selector keyword 24](#_Toc484515245)

[Resource Ordering 25](#_Toc484515246)

[Chaining arrows and Autorequires 28](#_Toc484515247)

# Configure Graphite class

Puppet version

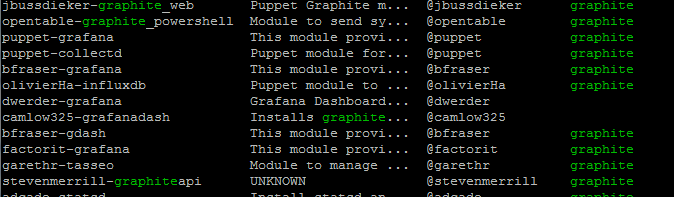


Using Puppet Enterprise to set up Graphite

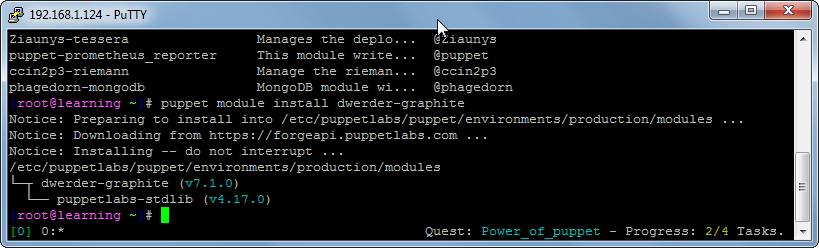
Graphite is built from several components, including the Graphite Django webapp frontend, a storage application called Carbon, and Whisper, a lightweight database system. Each of these components has its own set of dependencies and requires its own installation, and configuration.

Search for modules (Collection of config files) of graphite

**puppet module search graphite**

****

Install the graphite module **dwerder-graphite**

****

If the machine is offline, install local copy with the following command.

for m in /usr/src/forge/\*; do puppet module install $m --ignore-dependencies; done

Puppet modules path are found at:

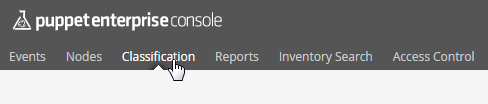
/etc/puppetlabs/puppet/environments/production/modules

A puppet module contains a main class that generally has the same name as the module name. Graphite class has the instruction puppet needs to set up graphite such as where and how to apply the class across your infrastructure (node - individual computers in a network). This is known as **classification**.

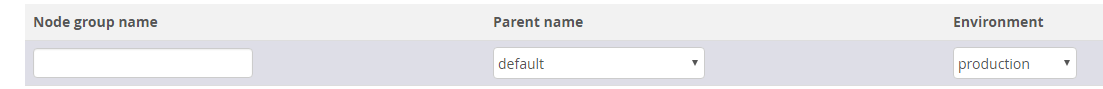
Get ipaddress of learning VM to access Puppet Enterprise console to set up nodes with facter. Facter contains facts about the system, e.g. ipaddress, OS type.



Go to <https://ipaddress> to access the console and click on classification



Add new node



Add rules for the node



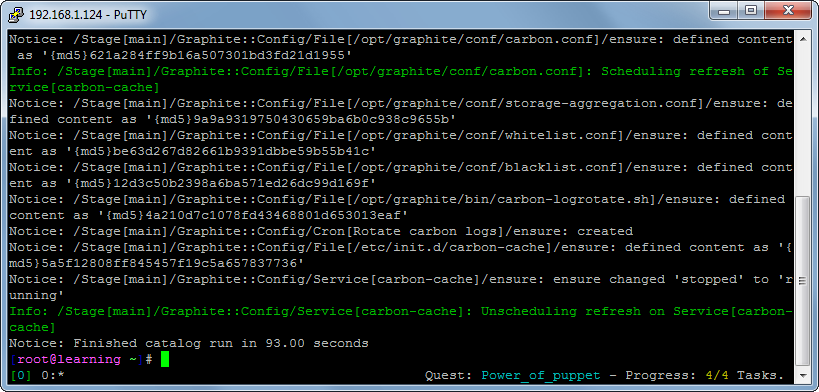
Add graphite class; disable default apache web server since graphite already has its own



Now that you have classified the learning.puppetlabs.vm node with the graphite class, Puppet knows how the system should be configured, but it won't make any changes until a Puppet run occurs. (Puppet agent does this automatically every 30mins)

To carry out puppet run manually

puppet agent --test



# Assign puppet resources

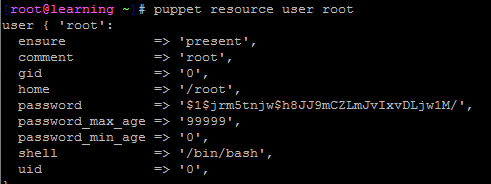
Resources contain **users, files, services, packages etc**

puppet resource = inspect

puppet describe = learn about

puppet apply = make changes to the resource

Check a user’s resource



Resource declaration format

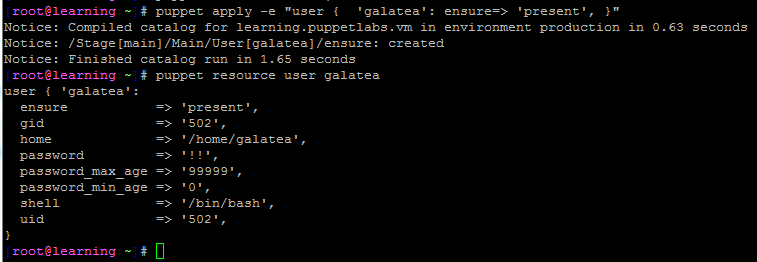
type {'title':

attribute **=>** 'value',

}

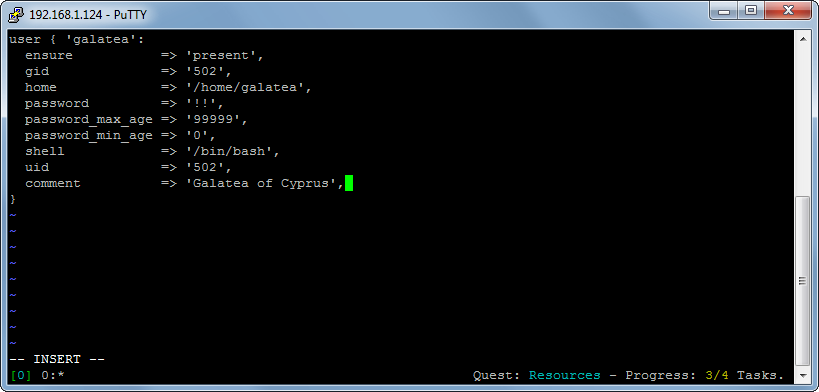
Creating a new user

**apply -e** is for one time use; i.e. if you want to reference a user again, you have to type the whole resource content

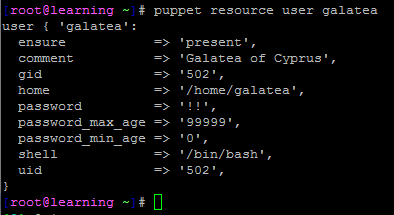


To edit the resource using vim, use the following command

puppet resource -e user galatea



Confirm changes



# Puppet manifests

Manifest is nothing more than some puppet code saved to a file with the .pp extension

In Puppet's DSL a class is a named block of Puppet code. The class is the next level of abstraction above a resource. A class declares a set of resources related to a single system component.

## Cowsay class

To use puppet to manage cowsay package

cd /etc/puppetlabs/puppet/environments/production/modules

To use cowsay command from command line, we can install cowsay using manifest

vim cowsayings/manifests/cowsay.pp

Enter the class definition on the pp file.

**class** cowsayings::cowsay {

**package** { 'cowsay':

ensure **=>** 'present',

}

}

Use parser to validate the syntax of the pp file. Nothing returned means no error.

puppet parser validate cowsayings/manifests/cowsay.pp



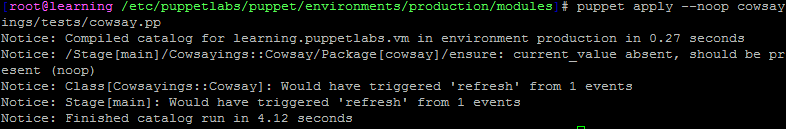
Declare the class using **include** keyword on test folder

vim cowsayings/tests/cowsay.pp

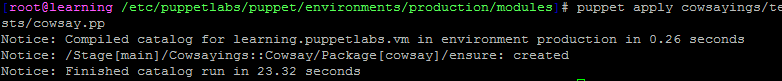
include cowsayings::cowsay

To carry out a dry run of the system, use **noop** keyword

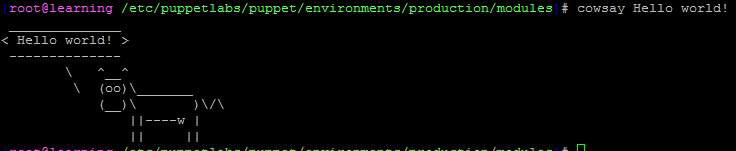
puppet apply --noop cowsayings/tests/cowsay.pp



Run it for real without - -noop flag



Confirm it’s working.



## Fortune class

Create Manifest

vim cowsayings/manifests/fortune.pp

Add class definition and validate the file

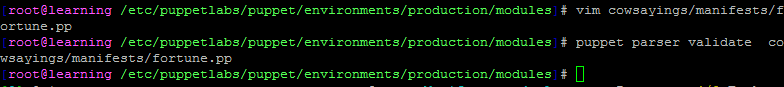
**class** cowsayings::fortune {

**package** { 'fortune-mod':

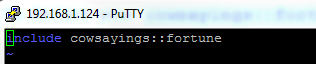
ensure **=>** 'present',

}

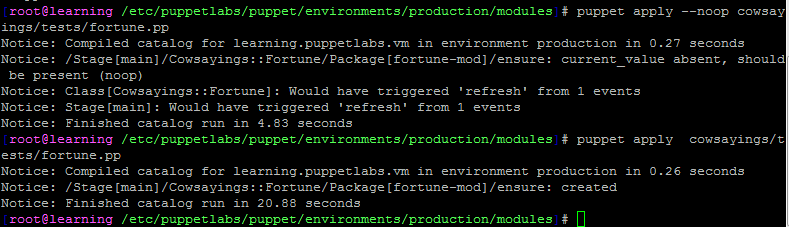
}



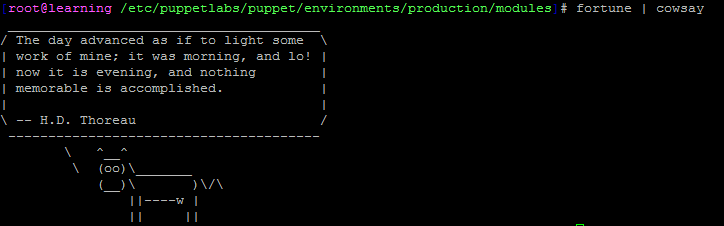
Declare the class file on test folder



Carry out dry-run with **--noop** flag, and then run without if everything is fine



Confirm it’s working. Fortune calls a random text and that is piped to cowsay



## Main class

Often a module will gather several classes that work together into a single class to let you declare everything at once.

The main class shares the name of the module itself, but instead of following the pattern of naming the manifest for the class it contains, Puppet recognizes the special file name **init.pp** for the manifest that will contain a module's main class.

Create init.pp file that contains the cowsayings class

vim cowsayings/manifests/init.pp

Declare the classes as within the tests directory

**class** cowsayings {

**include** cowsayings::cowsay

**include** cowsayings::fortune

}

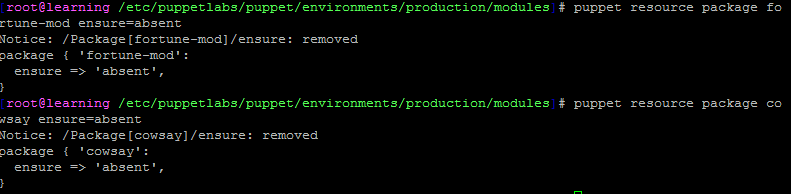
Validate the file



To carry out testing, delete the previously installed cowsay and fortune class using resource

puppet resource package fortune-mod ensure=absent

puppet resource package cowsay ensure=absent



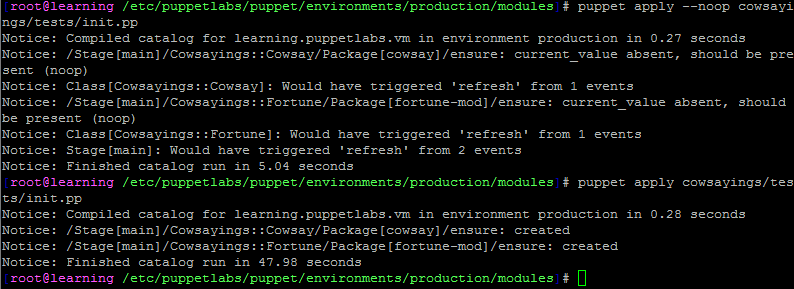
Create a test and declare the class

vim cowsayings/tests/init.pp

Add the following

include cowsayings

Carry out dry-run with **--noop** flag and run it after



A class is a collection of related resources and other classes which, once defined, can be declared as a single unit. Puppet classes are also singleton, which means that unlike classes in object oriented programming, a Puppet class can only be declared a single time on a given node.

A manifest is a file containing Puppet code, and appended with the .pp extension. In this quest, we used manifests in the ./manifests directory each to define a single class, and used a corresponding test manifest in the ./tests directory to declare each of those classes.

# Puppet module

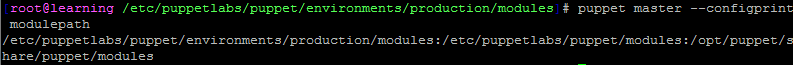
Modules allow you to organize your Puppet code into units that are testable, reusable, and portable, in short, **modular**. This means that instead of writing Puppet code from scratch for every configuration you need, you can mix and match solutions from a few well-written modules. And because these modules are separate and self-contained, they're much easier to test, maintain, and share than a collection of one-off solutions.

At their root, modules are little more than a structure of directories and files that follow Puppet's naming conventions. The module file structure gives Puppet a consistent way to locate whatever classes, files, templates, plugins, and binaries are required to fulfill the function of the module.

Puppet Labs hosts a free service called the [Forge](https://forge.puppetlabs.com/) where you can find a wide array of modules developed and maintained by others.

To find the **modulepath** where all modules are stored

puppet master --configprint modulepath



Puppet will look in the 3 directories above to find available modules

To list the modules

puppet module list

Use **tree** command, **-d** limit the output to directories, and **-L 2** limit the depth to two directories

tree -L 2 -d /etc/puppetlabs/puppet/environments/production/modules/

## Creating a module

The module will manage settings for the VIM editor.

Navigate to the directory puppet master will search for available modules

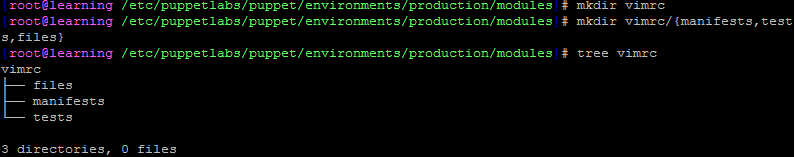
cd /etc/puppetlabs/puppet/environments/production/modules

Create a directory for the module

mkdir vimrc

Make 3 directories, for manifest, tests, file

mkdir vimrc/{manifests,tests,files}

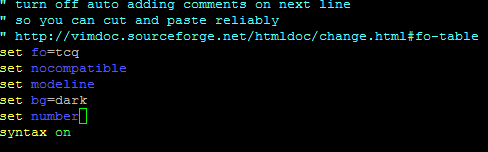


Copy existing VIM setting files on the Learning VM node group.

cp ~/.vimrc vimrc/files/vimrc

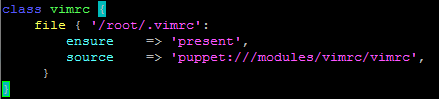
Edit the file to include **set number**

vim vimrc/files/vimrc



The source file is ready. Now write a manifest to tell puppet what to do with it.

* Create a main **init.pp** manifest for vimrc module
* Make vimrc class and tell Puppet server to take the **/vimrc/files/vimrc** file to required node/location
* The vimrc setting file path within puppet is in **/root/.vimrc.** Wewant to manage this and use a different one.
* **ensure => 'present',** ensures that the file is present in the system
* **source => …,** tells what the actual file should contain



Validate the file

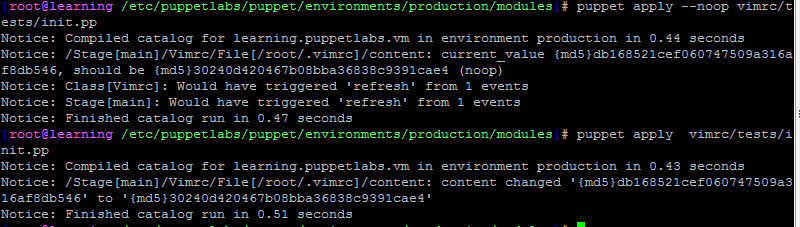


Declare and test the class

vim vimrc/tests/init.pp

**include** vimrc

Run with –noop flag and without



It checks old file and new file hash to check if it has been modified.

Vim is now using numbering system on our Learning VM which is also the agent/node.



# Using existing module – NTP

Check the package

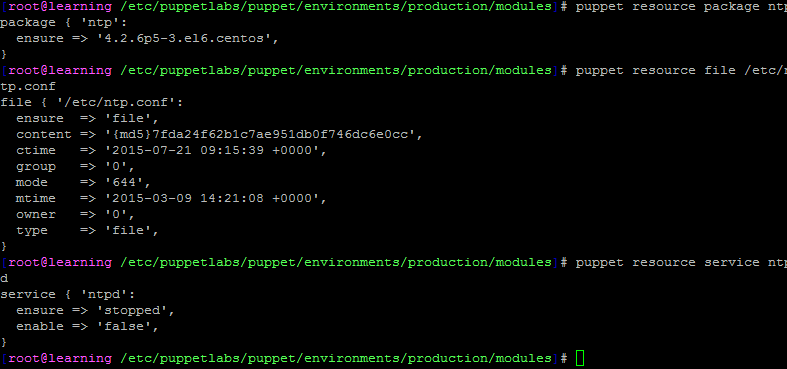
puppet resource package ntp

Check configuration file

puppet resource file /etc/ntp.conf

Check if it is running

puppet resource service ntpd



Install ntp by puppetlabs

puppet module install puppetlabs-ntp

Module is installed in /etc/puppetlabs/puppet/environments/production/modules

site.pp is the first manifest the Puppet agent checks when it connects to the master. It defines global settings and resource defaults that will apply to all nodes in your infrastructure. It is also where you will put your node definitions (sometimes called node statements). A node definition is a block of Puppet code that specifies a set of nodes and declares the classes that Puppet will enforce on those nodes. You can think of it as a code-defined version of the node group you set up in the Power of Puppet quest.

Open the site.pp

vim /etc/puppetlabs/puppet/environments/production/manifests/site.pp

Include the ntp protocol to be used at bottom of the site.pp file

**node** **default** {

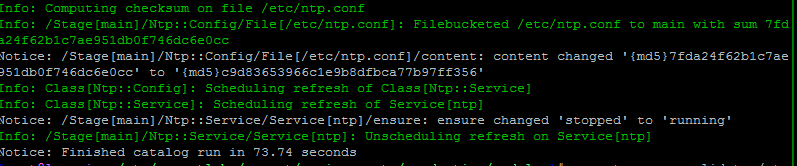
**include** ntp

}

Parse and run it manually

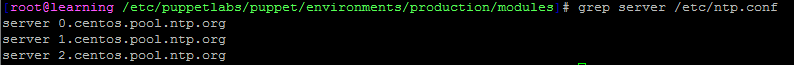


puppet agent -t



Check which default time server ntp is using

grep server /etc/ntp.conf



Replace the **include ntp** on **node default {}** of site.pp with

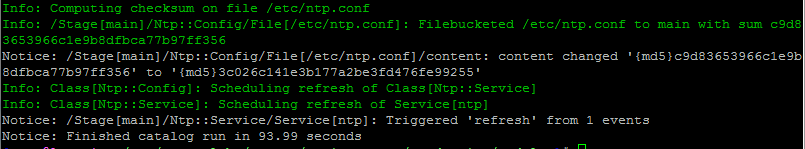
**class** { 'ntp':

servers **=>**

['nist-time-server.eoni.com','nist1-lv.ustiming.org','ntp-nist.ldsbc.edu']

}

Parse and run



# Using puppet to install and configure a server - MySQL

Install the mysql module

puppet module install puppetlabs-mysql

To classify the Learning VM with MySQL server class, edit the /etc/puppetlabs/puppet/environments/production/manifests/site.pp

Create password and set max connection allowed to 1024

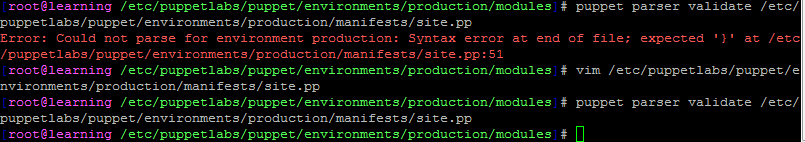
**class** { '::mysql::server':

root\_password **=>** 'strongpassword',

override\_options **=>** { 'mysqld' **=>** { 'max\_connections' **=>** '1024' } },

}

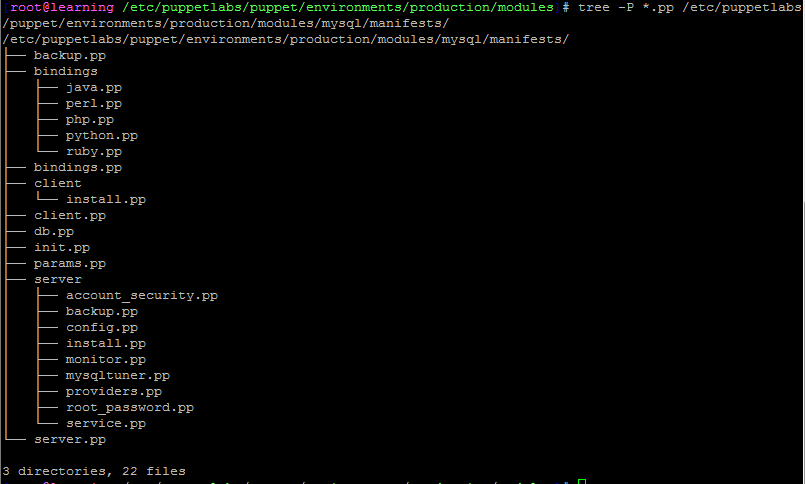
Parse and validate then run it with **puppet agent -t**



Check the result on MySQL **my.conf** file



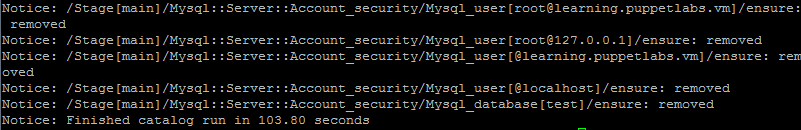
Check MySQL manifests



Trigger **the server::account\_security.pp** class to remove default accounts

Edit the **site.pp** file and include that class in **node default {..}**

mysql::server::account\_security



Create a database with puppet by including the following on **site.pp**

mysql\_database { 'lvm':

ensure **=>** 'present',

charset **=>** 'utf8',

}

Add user

mysql\_user { 'lvm\_user@localhost':

ensure **=>** 'present',

}

Grant privilege

mysql\_grant { 'lvm\_user@localhost/lvm.\*':

ensure **=>** 'present',

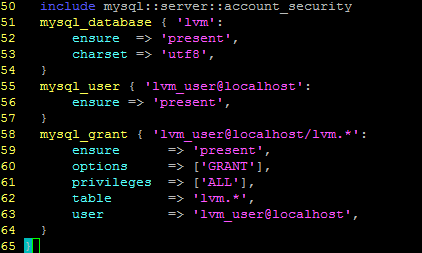
options **=>** ['GRANT'],

privileges **=>** ['ALL'],

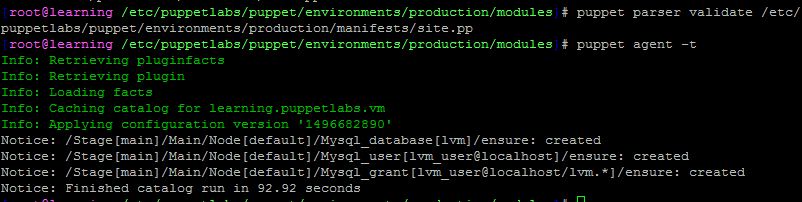
table **=>** 'lvm.\*',

**user** **=>** 'lvm\_user@localhost',

}



Parse and run



# Puppet variables and parameters

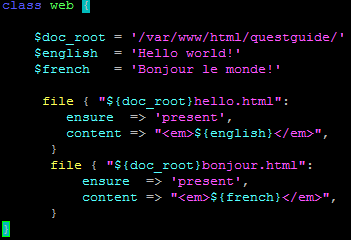
## Variables

On puppet module path, create a directory with manifests and tests folder



Make init.pp file with the following

vim web/manifests/init.pp

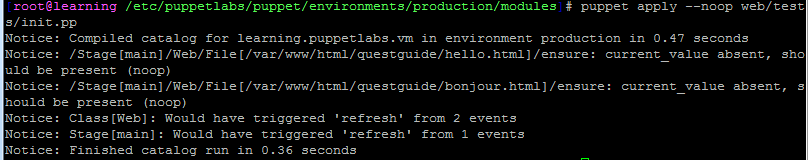


Parse the manifest file; then create class declaration on tests folder and include that class



****

Carry out dry run

****

## Class parameter

A class can be defined with variable within manifests

**class** classname ( $parameter **=** 'default' ) {

...

}

Then it can be **declared** within tests folder like resources

**class** {'classname':

parameter **=>** 'value',

}

Change the web class created earlier to add parameters and use them in file

**class** web ( $page\_name, $message ) {

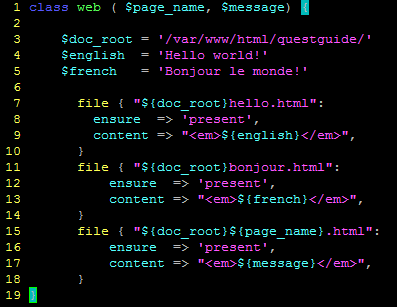
**file** { "${doc\_root}${page\_name}.html":

ensure **=>** 'present',

content **=>** "<em>${message}</em>",

}

}



Declare the class with required variables in its **web/tests/init.pp** file

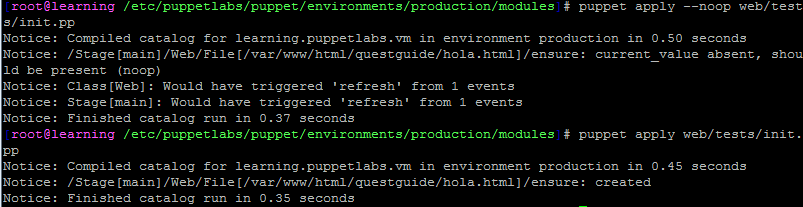
**class** {'web':

page\_name **=>** 'hola',

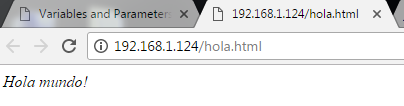
message **=>** 'Hola mundo!',

}

Carry out dry run before actual run



The file can be seen; page is **hola.html** and message is **‘Hola mundo!’**.



# Managing accounts in nodes with conditional statements

## If statement

Use puppet facts to make your modules portable.

Full list of facts available

facter -p | less

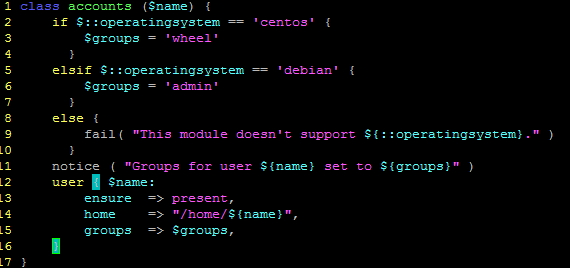
*(Press q to exit)*

Create account directory

mkdir accounts

mkdir accounts/{manifests,tests}

On accounts/manifests/init.pp include the following



On accounts/tests/init.pp declare the class with a user



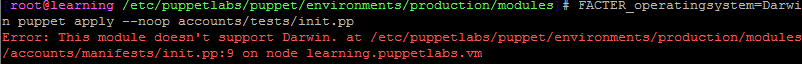
Check how the manifest will run on debian using facter

FACTER\_operatingsystem=Debian puppet apply --noop accounts/tests/init.pp

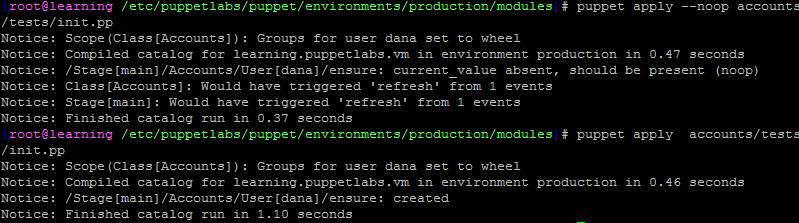
Dana’s group is correctly set to admin



FACTER\_operatingsystem=Darwin puppet apply --noop accounts/tests/init.pp



Run it within the Learning VM which is CentOS (Master and agent is same in this case)



Check user dana and group is correctly set to wheel



## Unless keyword

Opposite of “if statement”. Only execute if the conditional statement is false. If true, do nothing.

## Case keyword

**case** $::operatingsystem {

'CentOS': { $apache\_pkg **=** 'httpd' }

'Redhat': { $apache\_pkg **=** 'httpd' }

'Debian': { $apache\_pkg **=** 'apache2' }

'Ubuntu': { $apache\_pkg **=** 'apache2' }

**default**: { **fail**("Unrecognized operating system for webserver.") }

}

**package** { $apache\_pkg :

ensure **=>** **present**,

}

## Selector keyword

$rootgroup **=** $::osfamily **?** {

'Solaris' **=>** 'wheel',

'Darwin' **=>** 'wheel',

'FreeBSD' **=>** 'wheel',

'default' **=>** 'root',

}

# Resource Ordering

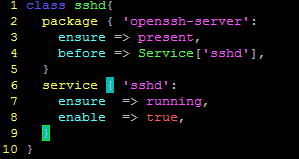
Create sshd directory with manifests and tests sub directory.



Fill in sshd class with the openssh-server package resource and sshd service resource.

If you're writing a module to manage SSH, for instance, you will need to ensure that the openssh-server package is installed before you try to manage the sshd service

Create **init.pp** file on **sshd/manifests/**



Declare the class using **init.pp** file under **sshd/tests/**

**include** sshd

When Puppet compiles a catalog, it generates a graph that represents the network of resource relationships in that catalog.

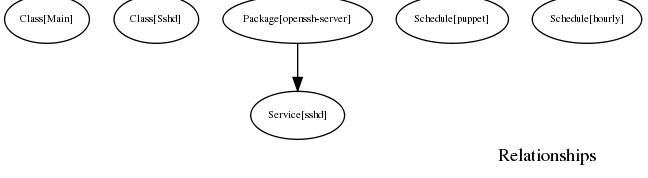
Dry run

puppet apply sshd/tests/init.pp --noop --graph

Real run

puppet config print graphdir

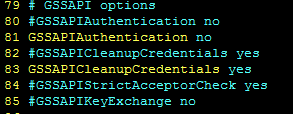
dot -Tpng /var/opt/lib/pe-puppet/state/graphs/relationships.dot -o /var/www/html/questguide/relationships.png



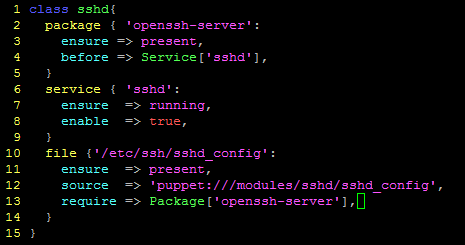
To configure the sshd copy its config file

cp /etc/ssh/sshd\_config sshd/files/sshd\_config

Make some change to the file **sshd/files/sshd\_config** such as change GSSAPIAuthentication line from **yes** to **no**



Go to manifest init file to add file resource

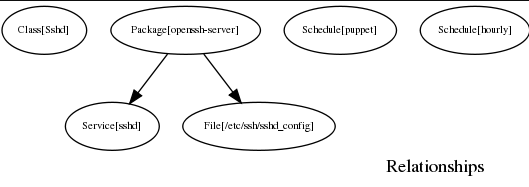


Dry run

puppet apply sshd/tests/init.pp --noop --graph

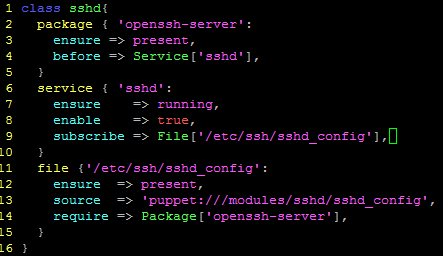
Regenerate graph

dot -Tpng /var/opt/lib/pe-puppet/state/graphs/relationships.dot -o /var/www/html/questguide/relationships.png

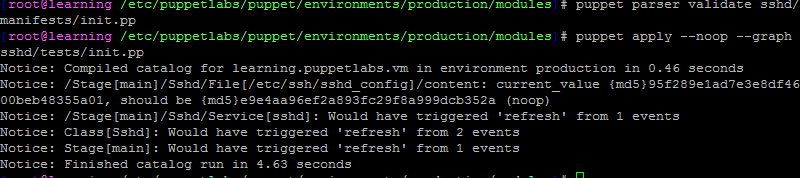


Puppet uses another pair of metaparameters to manage this special relationship between a service and its configuration file: notify and subscribe. The notify and subscribe metaparameters establish the same dependency relationships as before and require, respectively, and also trigger a refresh whenever Puppet makes a change to the dependency.

Make sshd service subscribe to the config file so it uses it before running the service.



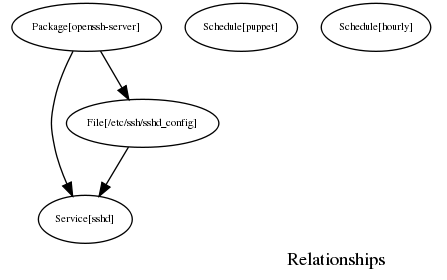
Parse and dry run



Make graph again

dot -Tpng /var/opt/lib/pe-puppet/state/graphs/relationships.dot -o /var/www/html/questguide/relationships.png





## Chaining arrows ****and Autorequires****

**Chaining arrows** provide another means for creating relationships between resources or groups of resources. The appropriate occasions for using chaining arrows involve concepts beyond the scope of this quest, but for the sake of completeness, we'll give a brief overview.

The -> (ordering arrow) operator causes the resource to the left to be applied before the resource to the right.

The ~> (notification arrow) operator causes the resource on the left to be applied before the resource on the right, and sends a refresh event to the resource on the right if the left resource changes.

**Autorequires** are relationships between resources that Puppet can figure out for itself. For instance, Puppet knows that a file resource should always come after a parent directory that contains it, and that a user resource should always be managed after the primary group it belongs to has been created.

