**Puppet part II**

**Using Puppet Modules to Create a LAMP Stack**

Within Puppet, modules are the building blocks of your servers’ configurations. Modules install and configure packages, create directories, and generate any other server changes that the user includes in the module. A Puppet module aims to perform all parts of a certain task, such as downloading the Apache package, configuring all files, changing the MPM data, and setting up virtual hosts. Modules are, in turn, broken down into classes that are .pp files meant to simplify the module into various tasks and improve the module’s readability for any future users.

In this guide, Apache and PHP modules will be created from scratch, and a MySQL module will be adapted from the Puppet Lab’s MySQL module found on the [Puppet Forge](https://forge.puppetlabs.com/). These steps will create a full LAMP stack on your server and provide an overview of the various ways modules can be utilized.

**Creating the Apache Module**

1. From the Puppet Master, navigate to Puppet’s module directory and create the apache module:
2. cd /etc/puppetlabs/code/environments/production/modules
3. Navigate to the manifests directory:
4. cd manifests

From here, the module will be separated into classes, based upon the goals of that particular section of code. In this instance, there will be an init.pp class for downloading the Apache package, a params.pp file to define any variables and parameters, config.pp to managed any configuration files for the Apache service itself, and a vhosts.pp file to define the virtual hosts. This module will also make use of [Hiera](http://docs.puppetlabs.com/hiera/latest/) data to store variables for each node.

**Create the Initial Apache Class and Parameters**

1. From within the manifests directory, an init.pp class needs to be created. This class should share its name with the module name:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/init.pp

|  |  |
| --- | --- |
|  | class apache {  } |

This file will be used to install the Apache package. Ubuntu 14.04 and CentOS 7 have different package names for Apache, however. Because of this, a variable will be used:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/init.pp

|  |  |
| --- | --- |
|  | class apache {  package { 'apache':  name => $apachename,  ensure => present,  }  } |

The package resource allows for the management of a package. This is used to add, remove, or ensure a package is present. In most cases, the name of the resource ('apache', above) should be the name of the package being managed. Because of the difference in naming conventions, however, this resource is simply called apache, while the actual *name* of the package is called upon with the name reference. name, in this instance, calls for the yet-undefined variable $apachename. The ensure reference ensures that the package is present.

1. Now that there are variables that need to be defined, the params.pp class will come into play. While these variables could be defined within the init.pp code, because more variables will need to be used outside of the resource type itself, using a params.pp class allows for variables to be defined in if statements and used across multiple classes.

Create and open params.pp:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/params.pp

|  |  |
| --- | --- |
|  | class apache::params {  } |

Outside of the original init.pp class, each class name needs to branch off of apache. As such, this class is called apache::params. The name after the double colon should share a name with the file.

1. The parameters should now be defined. To do this, an if statement will be used, pulling from information provided by [Facter](https://puppetlabs.com/facter), which is already installed on the Puppet master. In this case, Facter will be used to pull down the operating system family (osfamily), to discern if it is Red Hat or Debian-based.

The skeleton of the if statement should resemble the following:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/params.pp

|  |  |
| --- | --- |
|  | class apache::params {  if $::osfamily == 'RedHat' {  } elseif $::osfamily == 'Debian' {  } else {  print "This is not a supported distro."  }  } |

And once we’ve added the variables that have already been referenced:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/params.pp

|  |  |
| --- | --- |
|  | class apache::params {  if $::osfamily == 'RedHat' {  $apachename = 'httpd'  } elseif $::osfamily == 'Debian' {  $apachename = 'apache2'  } else {  print "This is not a supported distro."  }  } |

**Note**

For the duration of this guide, when something needs to be added to the parameter list the variables needed for Red Hat and Debian will be provided, but the expanding code will not be shown. A complete copy of params.pp can be viewed [here](https://www.linode.com/docs/assets/params.pp).

1. With the parameters finally defined, we need to call the params.pp file and the parameters into init.pp. To do this, the parameters need to be added after the class name, but before the opening curly bracket ({):

/etc/puppetlabs/code/environments/production/modules/apache/manifests/init.pp

|  |  |
| --- | --- |
|  | class apache (  $apachename = $::apache::params::apachename,  ) inherits ::apache::params { |

The value string $::apache::params::value tells Puppet to pull the values from the apache modules, params class, followed by the parameter name. The fragment inherits ::apache::params allows for init.pp to inherit these values.

**Manage Configuration Files**

Apache has two different configuration files, depending on whether you are working on a Red Hat- or Debian-based system.

1. Copy the httpd.conf and apache2.conf files to the files directory located at /etc/puppetlabs/code/environments/production/modules/apache/files/.
2. Both files need to be edited to turn KeepAlive settings to Off. This setting will need to be added to httpd.conf. Otherwise, a comment should be added to the top of each file:

/etc/puppetlabs/code/environments/production/modules/apache/files/httpd.conf

|  |  |
| --- | --- |
|  | # This file is managed by Puppet |

1. These files now need to be added to the init.pp file, so Puppet will know where they are located on both the master server and agent nodes. To do this, the file resource is used:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/init.pp

|  |  |
| --- | --- |
|  | file { 'configuration-file':  path => $conffile,  ensure => file,  source => $confsource,  } |

Because the configuration file is found in two different locations, the resource is given the generic name configuration-file with the file path defined as a parameter with the path attribute. ensure ensures that it is a file. source is another parameter, which will call to where the master files created above are located on the Puppet master.

1. Open the params.pp file. The $conffile and $confsource variables need to be defined within the if statement:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/params.pp

|  |  |
| --- | --- |
|  | if $::osfamily == 'RedHat' {  ...  $conffile = '/etc/httpd/conf/httpd.conf'  $confsource = 'puppet:///modules/apache/httpd.conf'  } elsif $::osfamily == 'Debian' {  ...  $conffile = '/etc/apache2/apache2.conf'  $confsource = 'puppet:///modules/apache/apache2.conf'  } else {  ... |

These parameters will also need to be added to the init.pp file, following the example of the additional parameters. A complete copy of the init.pp file can be seen [here](https://www.linode.com/docs/assets/puppet_apacheinit.pp).

1. When the configuration file is changed, Apache needs to restart. To automate this, the service resource can be used in combination with the notify attribute, which will call the resource to run whenever the configuration file is changed:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/init.pp

|  |  |
| --- | --- |
|  | file { 'configuration-file':  path => $conffile,  ensure => file,  source => $confsource,  notify => Service['apache-service'],  }  service { 'apache-service':  name => $apachename,  hasrestart => true,  } |

The service resource uses the already-created parameter that defined the Apache name on Red Hat and Debian systems. The hasrestart attribute will trigger a restart of the defined service.

**Create the Virtual Hosts Files**

The Virtual Hosts files will be managed differently, depending on whether the server is based on Red Hat or Debian distributions. Because of this, the code for virtual hosts will be encased in an if statement, similar to the one used in the params.pp class but containing actual Puppet resources. This will provide an example of an alternate use of if statement within Puppet’s code.

1. From within the apache/manifests/ directory, create and open a vhosts.pp file.
2. Create the skeleton of the if statement:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/vhosts.pp

|  |  |
| --- | --- |
|  | class apache::vhosts {  if $::osfamily == 'RedHat' {  } elsif $::osfamily == 'Debian' {  } else {  }  } |

1. The location of the virtual hosts file on our CentOS 7 server is /etc/httpd/conf.d/vhost.conf. This file will need to be created as a template on the Puppet master. The same needs to be done for the Ubuntu virtual hosts file, which is located at /etc/apache2/sites-available/example.com.conf, replacing example.com with the server’s FQDN. The template for this file also needs to be created on the Puppet master. Navigate to the templates file within the apache module, and then create two files for your virtual hosts:

For Red Hat systems:

/etc/puppetlabs/code/environments/production/modules/apache/templates/vhosts-rh.conf.erb

|  |  |
| --- | --- |
|  | <VirtualHost \*:80>  ServerAdmin <%= @adminemail %>  ServerName <%= @servername %>  ServerAlias www.<%= @servername %>  DocumentRoot /var/www/<%= @servername -%>/public\_html/  ErrorLog /var/www/<%- @servername -%>/logs/error.log  CustomLog /var/www/<%= @servername -%>/logs/access.log combined  </Virtual Host> |

For Debian systems:

/etc/puppetlabs/code/environments/production/modules/apache/templates/vhosts-deb.conf.erb

|  |  |
| --- | --- |
|  | <VirtualHost \*:80>  ServerAdmin <%= @adminemail %>  ServerName <%= @servername %>  ServerAlias www.<%= @servername %>  DocumentRoot /var/www/html/<%= @servername -%>/public\_html/  ErrorLog /var/www/html/<%- @servername -%>/logs/error.log  CustomLog /var/www/html/<%= @servername -%>/logs/access.log combined  <Directory /var/www/html/<%= @servername -%>/public\_html>  Require all granted  </Directory>  </Virtual Host> |

Only two variables are used in these files: adminemail and servername. These will be defined on a node-by-node basis, within the site.pp file.

1. Return to the vhosts.pp file. The templates created can now be referenced in the code:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/vhosts.pp

|  |  |
| --- | --- |
|  | class apache::vhosts {  if $::osfamily == 'RedHat' {  file { '/etc/httpd/conf.d/vhost.conf':  ensure => file,  content => template('apache/vhosts-rh.conf.erb'),  }  } elsif $::osfamily == 'Debian' {  file { "/etc/apache2/sites-available/$servername.conf":  ensure => file,  content => template('apache/vhosts-deb.conf.erb'),  }  } else {  print "This is not a supported distro."  }  } |

Both distribution families call to the file resource and take on the title of the virtual host’s location on the respective distribution. For Debian, this once more means referencing the $servername value. The content attribute calls to the respective templates.

**Note**

Values containing variables, such as the name of the Debian file resource above, need to be wrapped in double quotes ("). Any variables in single quotes (') are parsed exactly as written and will not pull in a variable.

1. Both virtual hosts files reference two directories that are not on the distributions by default. These can be created through the use of the file resource, each located within the if statement. The complete vhosts.conf file should resemble:

/etc/puppetlabs/code/environments/production/modules/apache/manifests/vhosts.pp

|  |  |
| --- | --- |
|  | class apache::vhosts {  if $::osfamily == 'RedHat' {  file { '/etc/httpd/conf.d/vhost.conf':  ensure => file,  content => template('apache/vhosts-rh.conf.erb'),  }  file { "/var/www/$servername":  ensure => directory,  }  file { "/var/www/$servername/public\_html":  ensure => directory,  }  file { "/var/www/$servername/log":  ensure => directory,  }  } elsif $::osfamily == 'Debian' {  file { "/etc/apache2/sites-available/$servername.conf":  ensure => file,  content => template('apache/vhosts-deb.conf.erb'),  }  file { "/var/www/$servername":  ensure => directory,  }  file { "/var/www/html/$servername/public\_html":  ensure => directory,  }  file { "/var/www/html/$servername/logs":  ensure => directory,  }  } else {  print "This is not a supported distro."  }  } |

**Test and Run the Module**

1. From within the apache/manifests/ directory, run the puppet parser on all files to ensure the Puppet coding is without error:
2. sudo puppet parser validate init.pp params.pp vhosts.pp

It should return empty, barring any issues.

1. Navigate to the examples directory within the apache module. Create an init.pp file and include the created classes. Provide variables for servername and adminemail:

/etc/puppetlabs/code/environments/production/modules/apache/examples/init.pp

|  |
| --- |
| $serveremail = 'webmaster@example.com'  $servername = 'example.com'  include apache  include apache::vhosts |

1. Test the module by running puppet apply with the --noop tag:
2. sudo puppet apply --noop init.pp

It should return no errors, and output that it will trigger refreshes from events. To install and configure apache on the Puppet master, this can be run again without --noop , if so desired.

1. Navigate back to the main Puppet directory and then to the manifests folder (**not** the one located in the Apache module). If you are continuing this guide from the Puppet Install guide, you should have a site.pp file already created. If not, create one now.
2. Open site.pp and include the Apache module for each agent node. Also input the variables for the adminemail and servername parameters. If you followed the [Puppet Setup](https://www.linode.com/docs/applications/puppet/set-up-puppet-master-agent/) guide, a single node configuration within site.pp will resemble the following:

/etc/puppetlabs/code/environments/production/manifests/site.pp

|  |  |
| --- | --- |
|  | node 'ubuntuhost.example.com' {  $adminemail = 'webmaster@example.com'  $servername = 'hostname.example.com'  include accounts  include apache  include apache::vhosts  resources { 'firewall':  purge => true,  }  Firewall {  before => Class['firewall::post'],  require => Class['firewall::pre'],  }  class { ['firewall::pre', 'firewall::post']: }  }  node 'centoshost.example.com' {  $adminemail = 'webmaster@example.com'  $servername = 'hostname.example.com'  include accounts  include apache  include apache::vhosts  resources { 'firewall':  purge => true,  }  Firewall {  before => Class['firewall::post'],  require => Class['firewall::pre'],  }  class { ['firewall::pre', 'firewall::post']: }  } |

1. To run the new module on your agent nodes, log in to the nodes and run:
2. sudo puppet agent -t

**Using the MySQL Module**

Many modules needed to run a server already exist within Puppet Lab’s [Puppet Forge](https://forge.puppetlabs.com). These can be configured just as extensively as a module that you created and can save time since the module need not be created from scratch.

Install the [Puppet Forge’s MySQL module](https://forge.puppetlabs.com/puppetlabs/mysql) by PuppetLabs:

sudo puppet module install puppetlabs-mysql

This will also install any prerequisite modules.

**Use Hiera to Create Databases**

Before you begin to create the configuration files for the MySQL module, consider that you may not want the same values to be used across all agent nodes. To supply Puppet with the correct data per node, Hiera is used. In this instance, you will be using a different root password per node, thus creating different MySQL databases.

1. Navigate to /etc/puppet and create Hiera’s configuration file hiera.yaml in the main puppet directory:

/etc/puppet/hiera.yaml

|  |
| --- |
| :backends:  - yaml  :yaml:  :datadir: /etc/puppet/hieradata  :hierarchy:  - "nodes/%{::fqdn}"  - common |

The value under :backends: defines that you are writing data in YAML, whereas :datadir: calls to the directory where the Hiera data will be stored. The :hierarchy: section denotes that your data will be saved in files under the node directory, as a file named after the node’s FQDN. A common file will also contain default variables.

1. Ensure you are in the /etc/puppet/ directory, then create a directory for hieradata and nodes:
2. sudo mkdir -p hieradata/nodes
3. Navigate to the nodes directory:
4. cd hieradata/nodes
5. Use the puppet cert command to list what nodes are available, then create a YAML file for each, using the FQDN as the file’s name:
6. sudo puppet cert list --all
7. sudo touch {ubuntuhost.example.com.yaml,centoshost.example.com.yaml}
8. Open the first node’s configuration file to define the first database. In this example, the database is called webdata1, with username and password self-defined. The grant value is granting the user all access to the webdata1 database:

/etc/puppet/hieradata/nodes/ubuntuhost.example.com.yaml

|  |
| --- |
| databases:  webdata1:  user: 'username'  password: 'password'  grant: 'ALL' |

Repeat with the second server. In this example, the database is called webdata2:

/etc/puppet/hieradata/nodes/centoshost.example.com.yaml

|  |
| --- |
| databases:  webdata2:  user: 'username'  password: 'password'  grant: 'ALL' |

Save and close the files.

1. Return to the hieradata directory and create the file common.yaml. It will be used to define the default root password for MySQL:

/etc/puppet/hieradata/common.yaml

|  |  |
| --- | --- |
|  | mysql::server::root\_password: 'password' |

The common.yaml file is used when a variable is not defined elsewhere. This means all servers will share the same MySQL root password. These passwords can also be hashed to increase security.

1. Puppet now needs to know to use the information input in Hiera to create the defined database. Move to the mysql module directory and within the manifests directory create database.pp. Here you will define a class that will link the mysql::db resource to the Hiera data. It will also call the mysql::server class, so it will not have to be included later:

/etc/puppetlabs/code/environments/production/modules/mysql/manifests/database.pp

|  |
| --- |
| class mysql::database {  include mysql::server  create\_resources('mysql::db', hiera\_hash('databases'))  } |

1. Include include mysql::database within your site.pp file for both nodes.

**Create the PHP Module**

1. Create the php module in the modules path, generating the files, manifests, templates, and examples sub-folders afterward:
2. cd /etc/puppetlabs/code/environments/production/modules
3. puppet module generate --skip-interview test-php
4. Create and open init.pp. Because all that needs to be done is to install and ensure the PHP service is on and able to start on boot, all code will be contained in this file.
5. Two packages will be installed: The PHP package and the PHP Extension and Application Repository. Use the package resource for this:

/etc/puppetlabs/code/environments/production/modules/php/manifests/init.pp

|  |
| --- |
| class php {  package { 'php':  name: $phpname,  ensure: present,  }  package { 'php-pear':  ensure: present,  }  } |

Because the php package has different names on Ubuntu and CentOS, it will once again need to be defined with a parameter. However, because this is the only parameter we will be needing, it will be added directly to the init.pp file:

/etc/puppetlabs/code/environments/production/modules/php/manifests/init.pp

|  |
| --- |
| class php {  $phpname = $osfamily ? {  'Debian' => 'php5',  'RedHat' => 'php',  default => warning('This distribution is not supported by the PHP module'),  }  package { 'php':  name => $phpname,  ensure => present,  }  package { 'php-pear':  ensure => present,  }  } |

1. Use the service resource to ensure that PHP is on and set to start at boot:

/etc/puppetlabs/code/environments/production/modules/php/manifests/init.pp

|  |
| --- |
| class php {  $phpname = $osfamily ? {  'Debian' => 'php5',  'RedHat' => 'php',  default => warning('This distribution is not supported by the PHP module'),  }  package { 'php':  name => $phpname,  ensure => present,  }  package { 'php-pear':  ensure => present,  }  service { 'php-service':  name => $phpname,  ensure => running,  enable => true,  }  } |

1. Add include php to the hosts in your sites.pp file and run puppet agent -t on your agent nodes to pull in any changes to your servers.