Example 1: Install LAMP with a Single Manifest

If you have not ever written a Puppet manifest before, this example is a good place to start. The manifest will be developed on a Puppet agent node, and executed via puppet apply, so an agent/master setup is not required.

You will learn how to write a manifest that will use following types of resource declarations:

* **exec**: To execute commands, such as apt-get
* **package**: To install packages via apt
* **service**: To ensure that a service is running
* **file**: To ensure that certain files exist

**Create Manifest**

On a fresh *lamp-1* VPS, create a new manifest:

sudo vi /etc/puppet/manifests/lamp.pp

Add the following lines to declare the resources that we just determined we wanted. The inline comments detail each resource declaration:

# execute 'apt-get update'

exec { 'apt-update': # exec resource named 'apt-update'

command => '/usr/bin/apt-get update' # command this resource will run

}

# install apache2 package

package { 'apache2':

**require** => Exec['apt-update'], # require 'apt-update' before installing

**ensure** => installed,

}

# ensure apache2 service is running

service { 'apache2':

**ensure** => running,

}

# install mysql-server package

package { 'mysql-server':

**require** => Exec['apt-update'], # require 'apt-update' before installing

**ensure** => installed,

}

# ensure mysql service is running

service { 'mysql':

**ensure** => running,

}

# install php5 package

package { 'php5':

**require** => Exec['apt-update'], # require 'apt-update' before installing

**ensure** => installed,

}

# ensure info.php file exists

file { '/var/www/html/info.php':

**ensure** => file,

content => '<?php phpinfo(); ?>', # phpinfo code

**require** => Package['apache2'], # require 'apache2' package before creating

}

Save and exit.

**Apply Manifest**

Now you will want to use the puppet apply command to execute the manifest. On *lamp-1*, run this:

sudo puppet apply --test

You will see many lines of output that show how the state of your server is changing, to match the resource declarations in your manifest. If there were no errors, you should be able to visit the public IP address (or domain name, if you set that up), and see the PHP info page that indicates that Apache and PHP are working. You can also verify that MySQL was installed on your server (it has not been secured, but we're not going to worry about that for now). Congrats! You set up a LAMP stack with Puppet.

This particular setup isn't too exciting, because we did not take advantage of our agent/master setup. The manifest is currently not available to other agent nodes, and Puppet is not continuously checking (every 30 minutes) that our server is in the state that the manifest described.

Now we want to convert the manifest that we just developed into a module, so it can be used by your other Puppet nodes.

Example 2: Install LAMP by Creating a New Module

Now let's create a basic module, based on the LAMP manifest that was developed in example 1. We will do this on the Puppet *master* node this time. To create a module, you must create a directory (whose name matches your module name) in Puppet's modules directory, and it must contain a directory calledmanifests, and that directory must contain an init.pp file. The init.pp file must only contain a Puppet class that matches the module name.

**Create Module**

On the Puppet *master*, create the directory structure for a module named lamp:

cd /etc/puppet/modules

sudo mkdir -p lamp/manifests

Now create and edit your module's init.pp file:

sudo vi lamp/manifests/init.pp

Within this file, add a block for a class called "lamp", by adding the following lines:

**class** **lamp** {

}

Copy the contents of LAMP manifest that you created earlier (or copy it from example 1 above) and paste it into the *lamp* class block. In this file, you created a class definition for a "lamp" class. The code within the class is will not be evaluated at this time, but it is available to be declared. Additionally, because it complies with the Puppet conventions for defining a module, this class can be accessed as a module by other manifests.

Save and exit.

**Use Module in Main Manifest**

Now that we have a basic lamp module set up, let's configure our main manifest to use it to install a LAMP stack on *lamp-1*.

On the Puppet *master*, edit the main manifest:

sudo vi /etc/puppet/manifests/site.pp

Assuming the file is empty, add the following *node* blocks (replace "lamp-1" with the hostname of the Puppet agent that you want to install LAMP on):

node default { }

node 'lamp-1' {

}

A node block allows you to specify Puppet code that will only apply to certain agent nodes. The *default*node applies to every agent node that does not have a node block specified--we will leave it empty. The*lamp-1* node block will apply to your *lamp-1* Puppet agent node.

In the *lamp-1* node block, add the following code to use the "lamp" module that we just created:

**include** lamp

Now save and exit.

The next time your *lamp-1* Puppet agent node pulls its configuration from the master, it will evaluate the main manifest and apply the module that specifies a LAMP stack setup. If you want to try it out immediately, run the following command on the *lamp-1* agent node:

sudo puppet agent --test

Once it completes, you will see that a basic LAMP stack is set up, exactly like example 1. To verify that Apache and PHP are working, go to *lamp-1*'s public IP address in the a web browser:

http://lamp\_1\_public\_IP/info.php

You should see the information page for your PHP installation.

Note that you can reuse the "lamp" module that you created by declaring it in other node blocks. Using modules is the best way to promote Puppet code reuse, and it is useful for organizing your code in a logical manner.

Now we will show you how to use pre-existing modules to achieve a similar setup.

Example 3: Install LAMP with Pre-existing Modules

There is a repository of publically-available modules, at [the Puppet Forge](https://forge.puppetlabs.com/), that can be useful when trying to develop your own infrastructure. The Puppet Forge modules can be quickly installed with built-inpuppet module command. It just so happens that modules for installing and maintaining Apache and MySQL are available here. We will demonstrate how they can be used to help us set up our LAMP stack.

**Install Apache and MySQL Modules**

On your Puppet *master*, install the puppetlabs-apache module:

sudo puppet module install puppetlabs-apache

You will see the following output, which indicates the modules installed correctly:

Notice: Preparing to install into /etc/puppetlabs/puppet/modules ...

Notice: Downloading from https://forgeapi.puppetlabs.com ...

Notice: Installing -- do not interrupt ...

/etc/puppet/modules

└─┬ puppetlabs-apache (v1.0.1)

├── puppetlabs-concat (v1.0.0) [/etc/puppet/modules]

└── puppetlabs-stdlib (v3.2.0) [/etc/puppet/modules]

Also, install the puppetlabs-mysql module:

sudo puppet module install puppetlabs-mysql

Now the *apache* and *mysql* modules are available for use!

**Edit the Main Manifest**

Now let's edit our main manifest so it uses the new modules to install our LAMP stack.

On the Puppet *master*, edit the main manifest:

sudo vi /etc/puppet/manifests/site.pp

Assuming the file is empty, add the following node blocks (if you followed example 2, just delete the contents of the *lamp-1* node block):

node default { }

node 'lamp-1' {

}

Within the *lamp-1* node block, use a resource-like class declaration to use the *apache* module (the in-line comments explain each line):

**class** { '**apache**': # use the "apache" module

default\_vhost => **false**, # don't use the default vhost

default\_mods => **false**, # don't load default mods

mpm\_module => 'prefork', # use the "prefork" mpm\_module

}

**include** apache::mod::php # include mod php

apache::vhost { 'example.com': # create a vhost called "example.com"

port => '80', # use port 80

docroot => '/var/www/html', # set the docroot to the /var/www/html

}

The *apache* module can be passed parameters that override the default behavior of the module. We are passing in some basic settings that disable the default virtual host that the module creates, and make sure we create a virtual host that can use PHP. For complete documentation of the PuppetLabs-Apache module, check out its [readme](https://forge.puppetlabs.com/puppetlabs/apache).

Using the MySQL module is similar to using the Apache module. We will keep it simple since we are not actually using the database at this point. Add the following lines within the node block:

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

root\_password => 'password',

}

Like the Apache module, the MySQL module can be configured by passing parameters ([full documentation here](https://forge.puppetlabs.com/puppetlabs/mysql).

Now let's add the file resource that ensures info.php gets copied to the proper location. This time, we will use the *source* parameter to specify a file to copy. Add the following lines within the node block:

file { 'info.php': # file resource name

path => '/var/www/html/info.php', # destination path

**ensure** => file,

**require** => Class['apache'], # require apache class be used

source => 'puppet:///modules/apache/info.php', # specify location of file to be copied

}

This file resource declaration is slightly different from before. The main difference is that we are specifying the *source* parameter instead of the *content* parameter. *Source* tells puppet to copy a file over, instead of simply specifying the file's contents. The specified source, puppet:///modules/apache/info.php gets interpreted by Puppet into /etc/puppet/modules/apache/files/info.php, so we must create the source file in order for this resource declaration to work properly.

Save and exit site.pp.

Create the info.php file with the following command:

sudo sh -c 'echo "<?php phpinfo(); ?>" > /etc/puppet/modules/apache/files/info.php'

The next time your *lamp-1* Puppet agent node pulls its configuration from the master, it will evaluate the main manifest and apply the module that specifies a LAMP stack setup. If you want to try it out immediately, run the following command on the *lamp-1* agent node:

sudo puppet agent --test

Once it completes, you will see that a basic LAMP stack is set up, exactly like example 1. To verify that Apache and PHP are working, go to *lamp-1*'s public IP address in the a web browser:

http://lamp\_1\_public\_IP/info.php

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

# **Puppet creating and managing user accounts with SSH access:-**

Security and access control

To have good security and access control practices, we need to use the following policies:

1. Everyone who needs access to a machine has his/her own user account with an SSH key (not a password).
2. Access to special-purpose accounts, such as those used to deploy and run applications, or a database, is controlled by authorizing specific SSH keys, rather than using passwords.
3. Accounts that need certain, specific superuser privileges can get them via the sudo mechanism.
4. The root account is not accessible via the network (but there is secure, out-of-band access to the system console).
5. Third parties, such as contractors and support staff, get temporary access with limited privileges, which can be revoked once a job is finished.

Setting up policies listed above, while highly desirable from a security point of view, is time-consuming to do by hand and difficult to maintain. If a new user arrives, someone has to add and configure his account on every server. If a user leaves, the accounts have to be removed or locked everywhere.

Puppet can make it quicker and easier to manage user accounts securely across a large network. We can add or remove individual and shared accounts, control their access via SSH, manage their privileges via sudo, and have the changes immediately applied to every machine under Puppet's control, all without logging into a single server.

Puppet provides a couple of ways to help us manage users. The user resource type controls user accounts, and the ssh\_authorized\_key resource type controls SSH access to accounts. We can use Puppet to control user privileges by managing the **sudoers** file.

Creating a user

Edit our **manifests/site.pp** file as follows:

node 'puppet-agent' {

include user

}

Also, we need to edit our **modules/user/manifests/init.pp** file:

class user {

user { 'k':

ensure => present,

comment => 'bogo user',

home => '/home/k',

managehome => true

}

}

Run puppet:

ubuntu@puppet-agent:~$ sudo puppet agent --test

Info: Retrieving plugin

Info: Caching catalog for puppet-agent.ec2.internal

Info: Applying configuration version '1419727849'

Notice: /Stage[main]/User/User[k]/ensure: created

Notice: Finished catalog run in 0.22 seconds

ubuntu@puppet-agent:~$ cd /home

ubuntu@puppet-agent:/home$ ls

k ubuntu

Puppet's user resource type creates a user or modifies it if the user already exists. The following line declares a user whose login name is 'k':

user { 'k':

The user should be present:

ensure => present,

We can also specify here some information about the user:

comment => 'bogo user',

The comment attribute sets the user's full name.

home => '/home/k',

The home attribute sets the path to the user's home directory. Puppet will not create this directory for us unless we also set the managehome attribute:

managehome => true,

So the manifest says that a user named 'k' should exist, whose full name is 'bogo user', and that the home directory should be '/home/k', and that that directory should exist. Note that we have not specified a password for the user, and as a result 'k' will not yet be able to log in. Although Puppet can set passwords for users, SSH authentication is recommended.

Removing a user

To remove a user from the system altogether, use the **ensure => absent** attribute:

user { 'k':

ensure => absent,

}

When we run Puppet, the 'k' account will be removed though k's home directory and any files he owned will remain.

Access control

Now that we've created the user's account, we now need to provide a secure way for a user to log in. We can do this using the SSH protocol.

In this section, the Puppet master will put a public key of my labtop into **authorized\_keys** of Puppet agent so that I can login to the agent node from my labtop computer via ssh.

Puppet can manage SSH public keys and authorize them for user accounts, using the**ssh\_authorized\_key** resource type.

We'll need our own SSH public key for this. If we already have one on our own computer, display the contents:

If we don't have an SSH key, we can generate one for this exercise:

k@laptop:~/.ssh$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/home/k/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/k/.ssh/id\_rsa.

Your public key has been saved in /home/k/.ssh/id\_rsa.pub.

The key fingerprint is:

bf:af:d1:85:af:a6:5f:f9:19:ad:cf:94:df:7d:21:d1 k@laptop

The key's randomart image is:

+--[ RSA 2048]----+

| |

| |

| . |

| o E |

| S . o |

| . . + oo|

| o . =o+|

| o.o =B|

| +\*+ .oO|

+-----------------+

Now display the **id\_rsa.pub** file to see the public key:

k@laptop:~$ cat ~/.ssh/id\_rsa.pub

ssh-rsa AAAAB3...jjQfJ7 k@laptop

The key itself is the long string of numbers and letters, without the **ssh-rsa** part at the beginning, or the **k@laptop** part at the end. It's this string we'll put into the Puppet manifest in the next step.

Edit our **modules/user/manifests/init.pp** file as follows using our own key string as the value for key:

class user {

user { 'k':

ensure => present,

comment => 'bogo user',

home => '/home/k',

managehome => true

}

ssh\_authorized\_key { 'k\_ssh':

user => 'k',

type => 'rsa',

key => 'AAAA...GjjQfJ7',

}

}

Run Puppet:

root@puppet-agent:~# puppet agent --test

Info: Retrieving plugin

Info: Caching catalog for puppet-agent.ec2.internal

Info: Applying configuration version '1419754797'

Notice: /Stage[main]/User/Ssh\_authorized\_key[k\_ssh]/key: key changed 'AAA...iQ8JkZV1F' to 'AAAA...jjQfJ7'

Notice: Finished catalog run in 0.03 seconds

Actually, we got the output after changing the pub key.

At this point, Puppet has added the key to the file **/home/k/.ssh/authorized\_keys** on Puppet agent node. When we try to log in to k's account via SSH, the system will look in this file to see if our private key matches any of the public keys listed there. Assuming it does, we'll be able to log in to Puppet agent node of AWS via ssh:

k@laptop:~/.ssh$ ssh k@54.88.104.246

Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-36-generic x86\_64)

\* Documentation: https://help.ubuntu.com/

System information as of Sat Dec 27 19:11:48 UTC 2014

System load: 0.0 Processes: 104

Usage of /: 11.8% of 7.74GB Users logged in: 0

Memory usage: 11% IP address for eth0: 172.31.43.38

Swap usage: 0%

...

$ uname -a

Linux puppet-agent 3.13.0-36-generic #63-Ubuntu SMP Wed Sep 3 21:30:07 UTC 2014 x86\_64 x86\_64 x86\_64 GNU/Linux

$ cd /home

$ ls

k ubuntu