How To Install Puppet To Manage Your Server Infrastructure

Posted Aug 15, 2014

Top of Form

Bottom of Form

73

 201.1kviews [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [Scaling](https://www.digitalocean.com/community/tags/scaling?type=tutorials) [Ubuntu](https://www.digitalocean.com/community/tags/ubuntu?type=tutorials)

**Note:** A newer version of this tutorial, which uses Puppet Server instead of Puppet with Passenger, can be found here: [How To Install Puppet 4 in a Master-Agent Setup on Ubuntu 14.04](https://www.digitalocean.com/community/tutorials/how-to-install-puppet-4-in-a-master-agent-setup-on-ubuntu-14-04).

**Introduction**

Puppet, from Puppet Labs, is a configuration management tool that helps system administrators automate the provisioning, configuration, and management of a server infrastructure. Planning ahead and using config management tools like Puppet can cut down on time spent repeating basic tasks, and help ensure that your configurations are consistent and accurate across your infrastructure. Once you get the hang of managing your servers with Puppet and other automation tools, you will free up time which can be spent improving other aspects of your overall setup.

Puppet comes in two varieties, Puppet Enterprise and open source Puppet. It runs on most Linux distributions, various UNIX platforms, and Windows.

In this tutorial, we will cover how to install open source Puppet in an Agent/Master setup. This setup consists of a central *Puppet Master* server, where all of your configuration data will be managed and distributed from, and all your remaining servers will be *Puppet Agent* nodes, which can be configured by the puppet master server.

Prerequisites

To follow this tutorial, you must have root access to all of the servers that you want to configure Puppet with. You will also be required to create a new Ubuntu 14.04 VPS to act as the Puppet master server. If you do not have an existing infrastructure, feel free to recreate the example infrastructure (described below) by following the prerequisite DNS setup tutorial.

Before we get started with installing Puppet, ensure that you have the following prerequisites:

* **Private Network DNS:** Forward and reverse DNS must be configured, and every server must have a unique hostname. Here is a tutorial to [configure your own private network DNS server](https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04). If you do not have DNS configured, you must use your hosts file for name resolution. We will assume that you will use your private network for communication within your infrastructure.
* **Firewall Open Ports:** The Puppet master must be reachable on port 8140. If your firewall is too restrictive, check out this [UFW Tutorial](https://www.digitalocean.com/community/tutorials/how-to-setup-a-firewall-with-ufw-on-an-ubuntu-and-debian-cloud-server) for instructions on how to allow incoming requests on port 8140.

**Example Infrastructure**

We will use the following infrastructure to demonstrate how to set up Puppet:

| **Hostname** | **Role** | **Private FQDN** |
| --- | --- | --- |
| host1 | Generic Ubuntu 14.04 VPS | host1.nyc2.example.com |
| host2 | Generic Ubuntu 14.04 VPS | host2.nyc2.example.com |
| ns1 | Primary nameserver | ns1.nyc2.example.com |
| ns2 | Secondary nameserver | ns2.nyc2.example.com |

The puppet agent will be installed on all of these hosts. These hosts will be referenced by their private network interfaces, which are mapped to the ".nyc2.example.com" subdomain in DNS. This is the same infrastructure that is described in the prerequisite tutorial: [How To Configure BIND as a Private Network DNS Server on Ubuntu 14.04](https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04).

Once you have all of the prerequisites, let's move on to creating the Puppet master server!

Create Puppet Master Server

Create a new **Ubuntu 14.04** x64 VPS, using "puppet" as its hostname. Add its private network to your DNS with the following details:

| **Hostname** | **Role** | **Private FQDN** |
| --- | --- | --- |
| puppet | Puppet Master | puppet.nyc2.example.com |

If you just set up your DNS and are unsure how to add your host to DNS, refer to the [Maintaining DNS Records](https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04#MaintainingDNSRecords) section of the DNS tutorial. Essentially, you need to add an "A" and "PTR" record, and allow the new host to perform recursive queries. Also, ensure that you configure your search domain so your servers can use short hostnames to look up each other.

Using "puppet" as the Puppet master's hostname simplifies the agent setup slightly, because it is the default name that agents will use when attempting to connect to the master.

Now we need to set up NTP.

**Install NTP**

Because it acts as a certificate authority for agent nodes, the puppet master server must maintain accurate system time to avoid potential problems when it issues agent certificates--certificates can appear to be expired if there are time discrepancies. We will use Network Time Protocol (NTP) for this purpose.

First, do a one-time time synchronization using the ntpdate command:

sudo ntpdate pool.ntp.org

Your system time will be updated, but you need to install the NTP daemon to automatically update the time to minimize time drift. Install it with the following apt command:

sudo apt-get update && sudo apt-get -y install ntp

It is common practice to update the NTP configuration to use "pools zones" that are geographically closer to your NTP server. In a web browser, go to the [NTP Pool Project](http://www.pool.ntp.org/en/) and look up a *pool zone* that is geographically close the datacenter that you are using. We will go with the United States pool (http://www.pool.ntp.org/zone/us) in our example, because the servers are located in a New York datacenter.

Open ntp.conf for editing:

sudo vi /etc/ntp.conf

Add the time servers from the NTP Pool Project page to the top of the file:

server 0.us.pool.ntp.org

server 1.us.pool.ntp.org

server 2.us.pool.ntp.org

server 3.us.pool.ntp.org

Save and exit. Restart NTP to add the new time servers.

sudo service ntp restart

Now that our server is keeping accurate time, let's install the Puppet master software.

Install Puppet Master

There are a variety of ways to install open source Puppet. We will use the debian package called*puppetmaster-passenger*, which is provided by Puppet Labs. The *puppetmaster-passenger* package includes the Puppet master plus production-ready web server (Passenger with Apache), which eliminates a few configuration steps compared to using the basic *puppetmaster* package.

Download the Puppet Labs package:

cd ~; wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb

Install the package:

sudo dpkg -i puppetlabs-release-trusty.deb

Update apt's list of available packages:

sudo apt-get update

Install the puppetmaster-passenger package:

sudo apt-get install puppetmaster-passenger

The Puppet master, Passenger, Apache, and other required packages are now installed. Because we are using Passenger with Apache, the Puppet master process is controlled by Apache, i.e. it runs when Apache is running.

Before continuing, stop the Puppet master by stopping the apache2 service:

sudo service apache2 stop

Next, we want to lock the version of Puppet.

Lock the Version

Changes from version to version can occasionally cause your Puppet environment to stop working properly. For this reason, you will want to maintain a consistent Puppet version across your entire infrastructure. If you decide to upgrade to a newer version, make sure that you upgrade your *master*before any agent nodes, as the master cannot manage agents that have a higher version number.

Let's look up the version of our Puppet installation with the following command:

puppet help | tail -n 1

During the time of writing, the output from the previous command is Puppet v3.6.2. We can use apt's pin feature to lock our Puppet install to 3.6.\*, which will prevent apt from upgrading Puppet to a higher major release. Create a new file called in the apt preferences directory:

sudo vi /etc/apt/preferences.d/00-puppet.pref

Add the following lines to lock the puppet, puppet-common, and puppetmaster-passenger packages to the 3.6.\* (change this to match your installed version):

# /etc/apt/preferences.d/00-puppet.pref

Package: puppet puppet-common puppetmaster-passenger

Pin: version 3.6\*

Pin-Priority: 501

Save and exit. Your Puppet version is now locked.

The next step is to set up your Puppet master names and certificates.

Set Up Names and Certificates

Puppet uses SSL certificates to authenticate communication between master and agent nodes. The Puppet master acts as a certificate authority (CA), and must generate its own certificates which is used to sign agent certificate requests. We will setup the master's certificates now.

**Delete Existing Certificates**

Delete any existing SSL certificates that were created during the package install. The default location of Puppet's SSL certificates is /var/lib/puppet/ssl:

sudo rm -rf /var/lib/puppet/ssl

**Configure Certificate**

When creating the puppet master's certificate, include every DNS name at which agent nodes can contact the master at. In the case of our example, we will use "puppet" and "puppet.nyc2.example.com", the short hostname and FQDN, respectively.

Edit the master's puppet.conf file:

sudo vi /etc/puppet/puppet.conf

It will look something like the following:

**[main]**

logdir=/var/log/puppet

vardir=/var/lib/puppet

ssldir=/var/lib/puppet/ssl

rundir=/var/run/puppet

factpath=$vardir/lib/facter

templatedir=$confdir/templates

**[master]**

# These are needed when the puppetmaster is run by passenger

# and can safely be removed if webrick is used.

ssl\_client\_header = SSL\_CLIENT\_S\_DN

ssl\_client\_verify\_header = SSL\_CLIENT\_VERIFY

Delete the line with the templatedir option, as that option is deprecated.

Add the following two lines to the end of the [main] section (replace the highlighted text with the private FQDN):

certname = puppet

dns\_alt\_names = puppet,puppet.nyc2.example.com

It is important to use assign certname to "puppet" because the Apache/Passenger configuration expects the certificate to be named "puppet". If you decide that you want a different certname setting, be sure to edit the Apache config file (/etc/apache2/sites-available/puppetmaster.conf) to change the name of the SSL certificate path.

Save and exit.

**Generate New Certificate**

Now create new CA certificates by running the following command:

sudo puppet master --verbose --no-daemonize

You will see several lines of output saying that SSL keys and certificates are being created. Once you seeNotice: Starting Puppet master version 3.6.2, the certificate setup is complete. Press CTRL-C to return to the shell.

Sample Output:

Info: Creating a new SSL key for ca

Info: Creating a new SSL certificate request for ca

Info: Certificate Request fingerprint (SHA256): EC:7D:ED:15:DE:E3:F1:49:1A:1B:9C:D8:04:F5:46:EF:B4:33:91:91:B6:5D:19:AC:21:D6:40:46:4A:50:5A:29

Notice: Signed certificate request for ca

...

Notice: Signed certificate request for puppet

Notice: Removing file Puppet::SSL::CertificateRequest puppet at '/var/lib/puppet/ssl/ca/requests/puppet.pem'

Notice: Removing file Puppet::SSL::CertificateRequest puppet at '/var/lib/puppet/ssl/certificate\_requests/puppet.pem'

Notice: Starting Puppet master version 3.6.2

If you want to look at the cert info of the certificate that was just created, type in the following:

sudo puppet cert list -all

The previous command actually lists all of the signed certificates and unsigned certificate requests. Currently, only the master cert will display, because no other certificates have been added yet:

+ "puppet" (SHA256) 05:22:F7:65:64:CF:46:0E:09:2C:5D:FD:8C:AC:9B:31:17:2B:7B:05:93:D5:D1:01:52:72:E6:DF:84:A0:07:37 (alt names: "DNS:puppet", "DNS:puppet.nyc2.example.com")

Our Puppet master service is almost ready to be started. Let's take a look at the master configuration first.

Configure Puppet Master

The main puppet configuration file, puppet.conf, consists of three sections: [main], [master], and[agent]. As you may have guessed, the "main" section contains global configuration, the "master" section is specific to the puppet master, and the "agent" is used to configure the puppet agent. Aside from the changes we made earlier, the defaults will work fine for a basic setup.

The configuration file has many options which might be relevant to your own setup. A full description of the file is available at Puppet Labs: [Main Config File (puppet.conf)](https://docs.puppetlabs.com/puppet/latest/reference/config_file_main.html).

If you want to edit it, run this command:

sudo vi /etc/puppet/puppet.conf

Let's take a look at the main manifest file.

**Main Manifest File**

Puppet uses a domain-specific language to describe system configurations, and these descriptions are saved to files called "manifests", which have a .pp file extension. The default main manifest file is located at /etc/puppet/manifests/site.pp. We will cover some basic manifests later, but we will create a placeholder file for now:

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

**Start Puppet Master**

We are now ready to start the Puppet master. Start it by running the apache2 service:

sudo service apache2 start

Your Puppet master is running, but it isn't managing any agent nodes yet. Let's learn how to install and add Puppet agents!

Install Puppet Agent

The Puppet agent must be installed on any server that the Puppet master will manage. In most cases, this includes every server in your infrastructure. As mentioned in the introduction, the Puppet agent can run on all major Linux distributions, some UNIX platforms, and Windows. Because the installation varies on each OS slightly, we will only cover the installation on Ubuntu and Debian servers.

Instructions to install Puppet on other platforms are located in the [Puppet Labs Docs](https://docs.puppetlabs.com/guides/install_puppet/pre_install.html#next-install-puppet)--be sure to follow the "Install Puppet on Agent Nodes" section for your OS of choice.

**Note:** It is assumed that all of your Puppet nodes, including agent nodes, are configured to use your DNS. If you are creating a brand new server, make sure to [add it to your DNS](https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04#MaintainingDNSRecords) before installing the Puppet agent.

**Ubuntu / Debian Agent Node**

**Note**: All of our example agent nodes, *host1*, *host2*, *ns1*, and *ns2*, are Ubuntu 14.04 VPS. We will repeat this step for each server, so each can be managed by the Puppet master.

On your Puppet agent node, download the Puppet Labs package:

cd ~; wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb

Install the package:

sudo dpkg -i puppetlabs-release-trusty.deb

Update apt's list of available packages:

sudo apt-get update

Install the Puppet agent package (puppet):

sudo apt-get install puppet

The puppet agent is disabled by default. To change this, update its default file:

sudo vi /etc/default/puppet

And change value of START to "yes":

START=yes

Save and exit /etc/default/puppet.

**Lock the Version**

As with the Puppet master, we will want to use the apt pin feature to lock the version of the Puppet agent:

sudo vi /etc/apt/preferences.d/00-puppet.pref

Add the following lines to lock the puppet and puppet-common packages to the 3.6.\* (change this to match your installed version):

# /etc/apt/preferences.d/00-puppet.pref

Package: puppet puppet-common

Pin: version 3.6\*

Pin-Priority: 501

Save and exit. Your Puppet version is now locked.

**Configure Agent**

Before running the agent, we must make a few configuration changes.

Edit the agent's puppet.conf:

sudo vi /etc/puppet/puppet.conf

It will look exactly like the Puppet master's initial configuration file.

Again, delete the templatedir line. Then delete the [master] section, and all of the lines below it.

Assuming that the Puppet master is reachable at "puppet", the agent should be able to connect to the master. If the master is not available at "puppet", you will need to add the Puppet master's FQDN. We recommend configuring this regardless (substitute the FQDN with your own):

[agent]

server = puppet.nyc2.example.com

Save and exit.

The Puppet agent is ready to run. Do so by running the following command:

sudo service puppet start

If everything is configured properly, you should not see any output. The first time you run the Puppet agent, it generates an SSL certificate and sends a signing request to the Puppet master. After the Puppet master signs the agent's certificate, it will be able to communicate with the agent node.

**Note**: If this is your first Puppet agent, it is recommended that you attempt to sign the certificate on the Puppet master before adding your other agents. Once you have verified that everything works properly, then you can go back and add the remaining agent nodes with confidence.

Sign Request On Master

The first time Puppet runs on an agent node, it will send a certificate signing request to the Puppet master. Before the master will be able to communicate and control the agent node, it must sign that particular agent node's certificate. We will describe how to sign and check for signing requests.

**List Current Certificate Requests**

On the Puppet master, run the following command to list all unsigned certificate requests:

sudo puppet cert list

If you just set up your first agent node, you will see one request. It will look something like the following, with the agent node's FQDN as the hostname:

"host1.nyc2.example.com" (SHA256) B1:96:ED:1F:F7:1E:40:53:C1:D4:1B:3C:75:F4:7C:0B:A9:4C:1B:5D:95:2B:79:C0:08:DD:2B:F4:4A:36:EE:E3

Note that there is no + in front of it. This indicates that it has not been signed yet.

**Sign A Request**

To sign a certificate request, use the puppet cert sign command, with the hostname of the certificate you want to sign. For example, to sign host1.nyc2.example.com, you would use the following command:

sudo puppet cert sign host1.nyc2.example.com

You will see the following output, which indicates that the certificate request has been signed:

Notice: Signed certificate request for host1.nyc2.example.com

Notice: Removing file Puppet::SSL::CertificateRequest host1.nyc2.example.com at '/var/lib/puppet/ssl/ca/requests/host1.nyc2.example.com.pem'

The Puppet master can now communicate and control the node that the signed certificate belongs to.

If you want to sign all of the current requests, use the -all option, like so:

sudo puppet cert sign --all

**Revoke Certificates**

You may want to remove a host from Puppet, or rebuild a host then add it back to Puppet. In this case, you will want to revoke the host's certificate from the Puppet master. To do this, you will want to use the cleanaction:

sudo puppet cert clean hostname

The specified host's associated certificates will be removed from Puppet.

**View All Signed Requests**

If you want to view all of the requests, signed and unsigned, run the following command:

sudo puppet cert list --all

You will see a list of all of the requests. Signed requests are preceded by a + and unsigned requests do not have the +.

"ns2.nyc2.example.com" (SHA256) E4:F5:26:EB:B1:99:1F:9D:6C:B5:4B:BF:86:14:40:23:E0:50:3F:C1:45:D0:B5:F0:68:6E:B2:0F:41:C7:BA:76

+ "host1.nyc2.example.com" (SHA256) 71:A2:D3:82:15:0D:80:20:D4:7E:E3:42:C2:35:87:83:79:2B:57:1D:D5:5A:EC:F6:8B:EE:51:69:53:EB:6B:A1

+ "host2.nyc2.example.com" (SHA256) F4:79:A6:7C:27:0C:EA:8E:BC:31:66:FF:F2:01:AB:B1:35:7B:9F:5E:C8:C9:CE:82:EE:E8:2F:23:9F:0C:2B:ED

+ "puppet" (SHA256) 05:22:F7:65:64:CF:46:0E:09:2C:5D:FD:8C:AC:9B:31:17:2B:7B:05:93:D5:D1:01:52:72:E6:DF:84:A0:07:37 (alt names: "DNS:puppet", "DNS:puppet.nyc2.example.com")

Congrats! Your infrastructure is now ready to be managed by Puppet!

Getting Started with Puppet

Now that your infrastructure is set up to be managed with Puppet, we will show you how to do a few basic tasks to get you started.

**How Facts Are Gathered**

Puppet gathers facts about each of its nodes with a tool called *facter*. Facter, by default, gathers information that is useful for system configuration (e.g. OS names, hostnames, IP addresses, SSH keys, and more). It is possible to add custom facts if you need other facts to perform you configurations.

The facts gathered can be useful in many situations. For example, you can create an web server configuration template and automatically fill in the appropriate IP addresses for a particular virtual host. Or you can determine that your server's OS is "Ubuntu", so you should run the apache2 service instead ofhttpd. These are basic examples, but they should give you an idea of how facts can be used.

To see a list of facts that are automatically being gathered on your agent node, run the following command:

facter

**How The Main Manifest Is Executed**

The puppet agent periodically checks in with the puppet master (typically every 30 minutes). During this time, it will send facts about itself to the master, and pull a current catalog--a compiled list of resources and their desired states that are relevant to the agent, determined by the main manifest. The agent node will then attempt to make the appropriate changes to achieve its desired state. This cycle will continue as long as the Puppet master is running and communicating with the agent nodes.

**Immediate Execution on a Particular Agent Node**

It is also possible initiate the check for a particular agent node manually, by running the following command (on the agent node in question):

puppet agent --test

Running this will apply the main manifest to the agent running the test. You might see output like the following:

Info: Retrieving pluginfacts

Info: Retrieving plugin

Info: Loading facts in /var/lib/puppet/lib/facter/pe\_version.rb

Info: Loading facts in /var/lib/puppet/lib/facter/puppet\_vardir.rb

Info: Loading facts in /var/lib/puppet/lib/facter/root\_home.rb

Info: Loading facts in /var/lib/puppet/lib/facter/facter\_dot\_d.rb

Info: Caching catalog for host1.nyc2.example.com

Info: Applying configuration version '1407966707'

This command is useful for seeing how the main manifest will affect a single server immediately.

**One-off Manifests**

The puppet apply command allows you to execute manifests that are not related to the main manifest, on demand. It only applies the manifest to the node that you run the apply from. Here is an example:

sudo puppet apply /etc/puppet/modules/test/init.pp

Running manifests in this fashion is useful if you want to test a new manifest on an agent node, or if you just want to run a manifest once (e.g. to initialize an agent node to a desired state).

A Simple Manifest

As you may recall, the main manifest file on the Puppet master is located at/etc/puppet/manifests/site.pp.

On the *master*, edit it now:

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

Now add the following lines to describe a file resource:

file {'/tmp/example-ip': # resource type file and filename

**ensure** => present, # make sure it exists

mode => 0644, # file permissions

content => "Here is my Public IP Address: ${ipaddress\_eth0}.\n", # note the ipaddress\_eth0 fact

}

Now save and exit. The inline comments should explain the resource that we are defining. In plain English, this will make *ensure* that all agent nodes will have a file at /tmp/example-ip, with -rw-r--r--permissions, and text that contains the node's public IP address.

You can either wait until the agent checks in with the master automatically, or you can run the puppet agent --test command (from one of your agent nodes). Then run the following command to print the file:

cat /tmp/example-ip

You should see output that looks like the following (with that node's IP address):

Here is my Public IP Address: 128.131.192.11.

**Specify a Node**

If you want to define a resource for specific nodes, define a node in the manifest.

On the master, edit site.pp:

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

Now add the following lines:

node 'ns1', 'ns2' { # applies to ns1 and ns2 nodes

file {'/tmp/dns': # resource type file and filename

**ensure** => present, # make sure it exists

mode => 0644,

content => "Only DNS servers get this file.\n",

}

}

node default {} # applies to nodes that aren't explicitly defined

Save and exit.

Now Puppet will ensure that a file at /tmp/dns will exist on *ns1* and *ns2*. You may want to run the puppet agent --test command (from ns1 or ns2), if you do not want to wait for the scheduled Puppet agent pull.

Note that if you do not define a resource, Puppet will do its best not to touch it. So if you deleted these resources from the manifest, Puppet will not delete the files it created. If you want to have it delete the files, change ensure to absent.

These examples don't do anything useful, but they do prove that Puppet is working properly.

Using a Module

Now let's use a module. Modules are useful for grouping tasks together. There are many modules available in the Puppet community, and you can even write your own.

On the Puppet master, install the puppetlabs-apache module from forgeapi:

sudo puppet module install puppetlabs-apache

**Warning**: Do not use this module on an existing Apache setup. It will purge any Apache configurations that are not managed by Puppet.

Now edit site.pp:

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

Now add the following lines to install Apache on *host2*:

node 'host2' {

**class** { '**apache**': } # use apache module

apache::vhost { 'example.com': # define vhost resource

port => '80',

docroot => '/var/www/html'

}

}

Save and exit. Now the next time Puppet updates host2, it will install the Apache package, and configure a virtual host called "example.com", listening on port 80, and with a document root /var/www/html.

On *host2*, run the following command:

sudo puppet agent --test

You should see a bunch of output indicating that Apache is being installed. Once it is complete, go to*host2*'s public IP address. You should see the default Apache welcome page.

Congrats! You have used your first Puppet module!

Conclusion

Now that you have a basic agent/master Puppet installation, you are now ready to learn more about how to use Puppet to manage your server infrastructure. Check out the following tutorial: [Getting Started With Puppet Code: Manifests and Modules](https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules).

Getting Started With Puppet Code: Manifests and Modules

Posted Aug 21, 2014

Top of Form

Bottom of Form

27

 153.9kviews [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [Ubuntu](https://www.digitalocean.com/community/tags/ubuntu?type=tutorials)

Tutorial Series

This tutorial is part 2 of 3 in the series: [How To Use Puppet To Manage Your Servers](https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules#tutorial_series_28)

**Introduction**

After setting up Puppet in an agent/master configuration, you may need some help writing Puppet manifests and modules. In order to use Puppet effectively, you must understand how manifests and modules are constructed. This tutorial covers Puppet code basics, and will show you how to construct manifests and modules that will help you get started with using Puppet to manage your server environment. We will show three different ways to use Puppet to configure a LAMP stack on an Ubuntu 14.04 VPS.

Prerequisites

Before starting this tutorial, you must have a working agent/master Puppet setup. If you do not already have this, follow this tutorial: [How To Install Puppet To Manage Your Server Infrastructure](https://www.digitalocean.com/community/tutorials/how-to-install-puppet-to-manage-your-server-infrastructure).

You also need to be able to create at least one new VPS to serve as the Puppet agent node that the Puppet master will manage.

**Create a New Agent Node**

Create a new Ubuntu 14.04 VPS called "lamp-1", [add it as a Puppet agent node](https://www.digitalocean.com/community/tutorials/how-to-install-puppet-to-manage-your-server-infrastructure#InstallPuppetAgent), and sign its certificate request on the Puppet master.

Puppet Code Basics

Before getting into writing Puppet code that will configure our systems, let's step back and review some of the relevant Puppet terminology and concepts.

**Resources**

Puppet code is composed primarily of *resource declarations*. A resource describes something about the state of the system, such as a certain user or file should exist, or a package should be installed. Here is an example of a user resource declaration:

user { 'mitchell':

**ensure** => present,

uid => '1000',

gid => '1000',

shell => '/bin/bash',

home => '/home/mitchell'

}

Resource declarations are formatted as follows:

resource\_type { 'resource\_name'

attribute => value

...

}

Therefore, the previous resource declaration describes a user resource named 'mitchell', with the specified attributes.

To list all of the default resource types that are available to Puppet, enter the following command:

puppet resource --types

We will cover a few more resource types throughout this tutorial.

**Manifests**

Puppet programs are called manifests. Manifests are composed of puppet code and their filenames use the .pp extension. The default main manifest in Puppet installed via apt is/etc/puppet/manifests/site.pp.

If you have followed the prerequisite Puppet tutorial, you have already written a manifest that creates a file and installs Apache. We will also write a few more in this tutorial.

**Classes**

In Puppet, classes are code blocks that can be called in a code elsewhere. Using classes allows you reuse Puppet code, and can make reading manifests easier.

**Class Definition**

A class definition is where the code that composes a class lives. Defining a class makes the class available to be used in manifests, but does not actually evaluate anything.

Here is how a class **definition** is formatted:

**class** **example\_class** {

...

code

...

}

The above defines a class named "example\_class", and the Puppet code would go between the curly braces.

**Class Declaration**

A class declaration occurs when a class is called in a manifest. A class declaration tells Puppet to evaluate the code within the class. Class declarations come in two different flavors: normal and resource-like.

A **normal class declaration** occurs when the include keyword is used in Puppet code, like so:

**include** example\_class

This will cause Puppet to evaluate the code in *example\_class*.

A **resource-like class declaration** occurs when a class is declared like a resource, like so:

**class** { '**example\_class**': }

Using resource-like class declarations allows you to specify *class parameters*, which override the default values of class attributes. If you followed the prerequisite tutorial, you have already used a resource-like class declaration ("apache" class) when you used the PuppetLabs Apache module to install Apache on*host2*:

node 'host2' {

**class** { '**apache**': } # use apache module

apache::vhost { 'example.com': # define vhost resource

port => '80',

docroot => '/var/www/html'

}

}

Now that you know about resources, manifests, and classes, you will want to learn about modules.

**Modules**

A module is a collection of manifests and data (such as facts, files, and templates), and they have a specific directory structure. Modules are useful for organizing your Puppet code, because they allow you to split your code into multiple manifests. It is considered best practice to use modules to organize almost all of your Puppet manifests.

To add a module to Puppet, place it in the /etc/puppet/modules directory.

We will cover the details necessary to write your own basic module. If you want to learn more details, check out the [PuppetLabs Module Fundamentals](https://docs.puppetlabs.com/puppet/latest/reference/modules_fundamentals.html) reference guide.

Developing a Manifest

To demonstrate how to write a Puppet manifests, classes, and modules, we will use Puppet to set up LAMP stack on Ubuntu (similar to the setup in [this tutorial](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-14-04)). If you have never set up a LAMP stack before, you will want to run through the linked tutorial to familiarize yourself with how to set it up manually.

From the LAMP stack tutorial, we know that we want an Ubuntu 14.04 server with the following *resources*:

* Apache package (apache2) installed
* Apache service (apache2) running
* MySQL Server package (mysql-server) installed
* MySQL Server service (mysql) running
* PHP5 package (php5) installed
* A test PHP script file (info.php)
* Update apt before installing packages

The following three sections will show different ways to use Puppet to achieve similar results, a working LAMP server. The first example will show how to write a basic manifest that is all in one file. The second example will show how to build and use a class and module, building upon the manifest developed in the first example. Finally, the third example will show how to use pre-existing, publicly available modules to quickly and easily set up a similar LAMP stack. If you want to try all three examples, for learning purposes, we recommend starting with a fresh VPS (as described in the [prerequisites](https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules)) each time.

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

You should see the information page for your PHP installation.

Conclusion

Congratulations! You have used Puppet to set up an Ubuntu 14.04 LAMP stack.

Now that you are familiar with the basics of Puppet code, and are able to write basic manifests and modules, you should try to use Puppet to configure other aspects of your environment.

A good place to start is to use Puppet to manage your system users and your application configuration files. Remember that if you use Puppet to manage resources you must make changes to those particular resources on your Puppet master server, or they will be overwritten the next time your agent nodes do their periodic catalog pull request.