**PUPPET**

Installation: Please follow the below steps to install Puppet.

1. **Setup repository on both master and agent nodes.**

rpm -Uvh https://yum.puppetlabs.com/puppet6/puppet6-release-el-7.noarch.rpm

ls -lthr /etc/yum.repos.d

-rw-r--r-- 1 root root 411 Sep 18 16:57 puppet6.repo

**2. Install Puppet Server on the Master Node**

yum install -y puppetserver

1. **Configure Puppet Server**

vi /etc/sysconfig/puppetserver

Change the value shown like below.

**From:**

JAVA\_ARGS="-Xms2g -Xmx2g

**To:**

For 512MB, use below settings.

JAVA\_ARGS="-Xms512m -Xmx512m"

1. **Advanced Configurations (optional)**

Here, I am going to modify the Puppet Master settings for our requirement.

vi /etc/puppetlabs/puppet/puppet.conf

Place the below lines. Modify it according to your environment.

[master]

dns\_alt\_names = server.itzgeek.local,server

[main]

certname = server.itzgeek.local

server = server.itzgeek.local

environment = production

runinterval = 1h

**Start and enable the Puppet Server.**

systemctl start puppetserver

systemctl enable puppetserver

1. **Firewall**

The Puppet Master listens on port 8140, so configure the firewall in such way that managed nodes can connect to the master.

firewall-cmd --permanent --zone=public --add-port=8140/tcp

firewall-cmd --reload

1. **Install Puppet Agent**

yum install -y puppet-agent

1. **Edit the Puppet Agent Configuration**

vi /etc/puppetlabs/puppet/puppet.conf

Set like below.

[main]

certname = client.itzgeek.local

server = server.itzgeek.local

environment = production

runinterval = 1h

You can change the value of **runinterval** depends on the requirement, you can set the **value in seconds**; this controls how long agent should wait between the two catalog requests.

1. **Start puppet agent on the node and make it start automatically on system boot.**

/opt/puppetlabs/bin/puppet resource service puppet ensure=running enable=true

**Output:**

Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'

service { 'puppet':

  ensure => 'running',

  enable => 'true',

}

1. **Sign the Agent Nodes Certificate on Master Server**

In an agent/master deployment, an admin must approve a certificate request coming from each node so that they can fetch the configurations. Agent nodes will request certificates for the first time if they attempt to run.

Log into the puppet master server and run below command to view outstanding requests.

[root@ip-172-31-47-77 yum.repos.d]# /opt/puppetlabs/bin/puppetserver ca list

Requested Certificates:

ec2-34-220-207-19.us-west-2.compute.amazonaws.com (SHA256) 0E:80:4F:D6:9B:0F:18:7C:95:5B:55:B1:FA:EE:4C:A5:2C:05:D2:87:95:D9:D2:2B:1D:80:B6:09:86:B5:8C:EC

[root@ip-172-31-47-77 yum.repos.d]#Run puppet cert sign command to sign a request.

[root@ip-172-31-47-77 yum.repos.d]# /opt/puppetlabs/bin/puppetserver ca sign --certname ec2-34-220-207-19.us-west-2.compute.amazonaws.com

Successfully signed certificate request for ec2-34-220-207-19.us-west-2.compute.amazonaws.com

[root@ip-172-31-47-77 yum.repos.d]#

## Verify the Puppet Client

Once the Puppet master is signed your client certificate, run the following command on the **client machine** to test it.

/opt/puppetlabs/bin/puppet agent --test

**Output:**

Info: Using configured environment 'production'

Info: Retrieving pluginfacts

Info: Retrieving plugin

Info: Caching catalog for client.itzgeek.local

Info: Applying configuration version '1472165304'

Notice: Applied catalog in 0.05 seconds

**Install Puppet DB Service in Puppet Master Host**

**Install PostgreSQL**

**1. Download the source pertaining to Postgresql**

wget <https://ftp.postgresql.org/pub/source/v11.5/postgresql-11.5.tar.gz>

**2. Unpack the source code**

gunzip postgresql-11.5.tar.gz

tar xf postgresql-11.5.tar

**3. Install the below dependency packages**

yum install -y readline-devel

yum install -y gcc.x86\_64

yum install -y zlib-devel

**4. Install postgresql**

cd postgresql-11.5

./configure

make

make install

**5. Create a Super User account for postgresql**

adduser postgres

**6. Create a directory to hold PostgreSQL data tree**

mkdir -p ~/pgsql/data

chown postgres ~/pgsql/data

**7.** **Create PostgreSQL cluster**

su - postgres

/usr/local/pgsql/bin/initdb -D ~/pgsql/data

**8. Start up the PostgreSQL server (postmaster process)**

/usr/local/pgsql/bin/pg\_ctl -D /home/postgres/pgsql/data -l logfile start

**9. Create a PostgreSQL database in the cluster**

/usr/local/pgsql/bin/createdb test

**10.** **Log-in to the database**

/usr/local/pgsql/bin/psql test

## Install PuppetDB

/opt/puppetlabs/puppet/bin/puppet resource package puppetdb ensure=latest

## Init Script Config File

To change the JVM heap size for PuppetDB, edit the init script config file (i.e., /etc/sysconfig/puppetdb) by setting a new value for the -Xmx flag in the JAVA\_ARGS variable.

For example, to cap PuppetDB at 192MB of memory:

JAVA\_ARGS="-Xmx192m"

To use 1GB of memory:

JAVA\_ARGS="-Xmx1g"

## The PuppetDB configuration file(s)

The default config directory is /etc/puppetlabs/puppetdb/conf.d

Ensure the below files contain the corresponding content in them, if not add them

vi global.ini

[global]

vardir = /etc/puppetlabs/puppetdb

logging-config = /etc/puppetlabs/puppetdb/logback.xml

vi database.ini

[database]

classname = org.postgresql.Driver

subprotocol = postgresql

subname = //localhost:5432/puppetdb

vi puppetdb.ini

[puppetdb]

certificate-whitelist = /etc/puppetlabs/puppet/ssl/private\_keys

disable-update-checking = false

vi jetty.ini

port = 8080

## Start the PuppetDB service

/opt/puppetlabs/puppet/bin/puppet resource service puppetdb ensure=running enable=true

**Issues:**

1. **Puppet test fails with the error “**is not qualified and no path was specified. Please qualify the command or specify a path**”**

[root@Puppet-Agent ~]# /opt/puppetlabs/bin/puppet agent --test

Info: Using configured environment 'production'

Info: Retrieving pluginfacts

Info: Retrieving plugin

Info: Retrieving locales

Info: Caching catalog for ec2-34-220-207-19.us-west-2.compute.amazonaws.com

Error: Failed to apply catalog: Parameter unless failed on Exec[apache]: 'ping -c1 ec2-35-160-156-203.us-west-2.compute.amazonaws.com' is not qualified and no path was specified. Please qualify the command or specify a path. (file: /etc/puppetlabs/code/environments/production/manifests/init.pp, line: 19)

**Solution:** Mention the absolute path to the executable

**Eg:** unless => '/bin/ping -c1 ec2-35-160-156-203.us-west-2.compute.amazonaws.com',

**Error 2: The file mode specification must be a string, not 'Integer'**

Error: Failed to apply catalog: Parameter mode failed on File[/tmp/test\_script.sh]: The file mode specification must be a string, not 'Integer' (file: /etc/puppetlabs/code/environments/production/manifests/test.pp, line: 4)

**Solution:**

Change mode => 755, to mode => '755',

**Error 3: Could not find resource 'Exec[include nginx]' in parameter 'notify'**

Error: Could not retrieve catalog from remote server: Error 500 on SERVER: Server Error: Could not find resource 'Exec[include nginx]' in parameter 'notify' (file: /etc/puppetlabs/code/environments/production/manifests/test.pp, line: 6) on node hgbupap13vm08.in.oracle.com

Warning: Not using cache on failed catalog

Error: Could not retrieve catalog; skipping run

**Solution:**

**Ensure that the resource is defince within the same manifest file**

**Error 4: Class 'nginx' is already defined**

Error: Could not retrieve catalog from remote server: Error 500 on SERVER: Server Error: Class 'nginx' is already defined (file: /etc/puppetlabs/code/environments/production/manifests/nginx.pp, line: 1); cannot redefine (file: /etc/puppetlabs/code/environments/production/manifests/test.pp, line: 16) on node hgbupap13vm08.in.oracle.com

Warning: Not using cache on failed catalog

Error: Could not retrieve catalog; skipping run

**Solution: Ensure that the class definition is not there in any other manifest file, if so, remove it.**

**Error 5: Error evaluating generate function**

Error: Could not retrieve catalog from remote server: Error 500 on SERVER: Server Error: Evaluation Error: Error while evaluating a Function Call, Generators can only contain alphanumerics, file separators, and dashes (file: /etc/puppetlabs/code/environments/production/manifests/class.pp, line: 1, column: 11) on node hgbupap13vm08.in.oracle.com

**Reason:**

The generate function will allow one to run the command on the puppet server and not on the client servers.

**Solution:** Use Puppet Facter instead of Generate.

**Error 6: The certificate retrieved from the master does not match the agent's private key**

[root@puppet-agent ~]# /opt/puppetlabs/bin/puppet agent --test

Error: Could not request certificate: The certificate retrieved from the master does not match the agent's private key. Did you forget to run as root?

Certificate fingerprint: 2B:4F:D3:54:5D:E0:68:FA:BA:88:A4:FC:8B:C5:24:0D:57:5D:AF:A9:C8:79:FE:65:7B:10:89:A2:5F:02:3A:32

**To fix this, remove the certificate from both the master and the agent and then start a puppet run, which will automatically regenerate a certificate.**

On the master:

puppetserver ca clean --certname ec2-34-221-81-14.us-west-2.compute.amazonaws.com

On the agent:

1. puppet ssl clean

2. puppet agent -t

Exiting; failed to retrieve certificate and waitforcert is disabled

[root@puppet-agent ~]# /opt/puppetlabs/bin/puppet ssl clean

Notice: Removed private key /etc/puppetlabs/puppet/ssl/private\_keys/ec2-34-221-81-14.us-west-2.compute.amazonaws.com.pem

Notice: Removed public key /etc/puppetlabs/puppet/ssl/public\_keys/ec2-34-221-81-14.us-west-2.compute.amazonaws.com.pem

Notice: Removed certificate /etc/puppetlabs/puppet/ssl/certs/ec2-34-221-81-14.us-west-2.compute.amazonaws.com.pem

[root@puppet-agent ~]# /opt/puppetlabs/bin/puppet agent --test

**Solution:**

**To fix this, remove the certificate from both the master and the agent and then start a puppet run, which will automatically regenerate a certificate.**

On the master:

puppetserver ca clean --certname ec2-34-221-81-14.us-west-2.compute.amazonaws.com

On the agent:

1. puppet ssl clean

2. puppet agent -t

**Puppet Commands:**

puppet help

\* clean:

Use `puppetserver ca clean --certname NAME[,NAME...]`

\* fingerprint:

Use openssl directly:

`openssl x509 -noout -fingerprint -<digest> -inform pem -in certificate.crt`

\* generate:

Use `puppetserver ca generate --certname NAME[,NAME...]`

\* list:

Use `puppetserver ca list [--all]`

\* print:

Use openssl directly:

`openssl x509 -noout -text -in certificate.pem`

\* revoke:

Use `puppetserver ca revoke --cerntname NAME[,NAME...]`

\* sign:

Use `puppetserver ca sign --cerntname NAME[,NAME...]`

\* verify:

Use `puppet ssl verify [--certname NAME]`

\* reinventory:

Removed.puppet agent -t -> 1. To generate a CSR, if one does n t exist.

2. To fetch the generated cert from the Master and

3. To fetch the latest configuration and apply on the Agent.

**Manifest**: Is nothing but the Puppet code.

**Module**: A collection of manifests and data (such as facts, files and templates) and they have a specific directory structure.

**Catalog**: When configuring a node, Puppet Agent uses a document called a catalog, which it downloads from a Puppet Master. The catalog describes the desired state for each resource that should be managed, and may specify dependency information for resources that should be managed in a certain order.

**etckeeper-commit-post:** In this configuration file you can define command and scripts which executes after pushing configuration on Agent.

**etckeeper-commit-pre:** In this configuration file you can define command and scripts which executes before pushing configuration on Agent.

By default every 30 minutes a Puppet Agent will pull the configuration from the Puppet Master.

**Manifest Example:**

vi /etc/puppet/manifests/site.pp

file {‘/etc/inetd.conf’:

ensure => ‘/etc/inetd.conf’,

}

package {‘apache’:

ensure => installed,

}

In order to download any software from the Puppet Forge use the below command

cd /etc/puppet/modules/

puppet module install puppetlabs-mysql -version 3.10.0

Then create a manifest to install the downloaded software

vi /etc/puppet/manifests/site.pp

include ‘::mysql::server’

include ‘::php’

Go to Puppet Agent and run ‘puppet agent -cist’

service{‘httpd’:

ensure => running,

}

**To get detailed error message**

journalctl -u puppetserver

**Sample Class with exec resource**

class apache {

package {'httpd':

ensure => latest, # Using the class parameter from above

before => File['/etc/httpd.conf'],

}

file {'/etc/httpd.conf':

ensure => file,

owner => 'httpd',

content => template('apache/httpd.conf.erb'), # Template from a module

}

service {'httpd':

ensure => running,

enable => true,

subscribe => File['/etc/httpd.conf'],

}

exec { 'apache':

unless => '/bin/ping -c1 ec2-35-160-156-203.us-west-2.compute.amazonaws.com',

command => 'class { "apache": }'

}

}

“If” statements

“If” statements take a [boolean](https://puppet.com/docs/puppet/5.3/lang_data_boolean.html) condition and an arbitrary block of Puppet code, and will only execute the block if the condition is true. They can optionally include elsif and else clauses.

Syntax

if $facts['is\_virtual'] {

# Our NTP module is not supported on virtual machines:

warning('Tried to include class ntp on virtual machine; this node might be misclassified.')

}

elsif $facts['os']['name'] == 'Darwin' {

warning('This NTP module does not yet work on our Mac laptops.')

}

else {

# Normal node, include the class.

include ntp

}

**Puppet Logs:**

/var/log/puppetlabs/puppetserver/puppetserver.log

**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': }**

**What is a Catalog?**

In Puppet, the combined configuration to be applied to a host is called a catalog, and the process of applying it is called a run.

**Puppet Agent:**

The Puppet client software is called the agent. Puppet calls the definition of

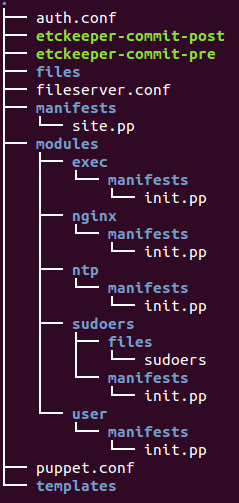
the host itself a node. The Puppet server is called the master.

**Modules and Classes:**

1. A module is a directory.
2. The module's name must be the name of the directory.
3. It contains a manifests directory, which can contain any number of .pp files.
4. The manifests directory should always contain an init.pp file.
5. This file must contain a single class definition. The class's name must be the same as the module's name.

A good module is made up of small, self-contained classes that each do only one thing. Classes within a module are similar to functions in programming, using parameters to perform related steps that create a coherent whole.

A typical module structure is depicted in the below diagram:



1. In general, files must have the same named as the class or definition that it contains, and classes must be named after their function. The one exception to this rule is the main class of a module, which is defined in the init.pp file, but is called by the same name as the module. Generally, a module includes:

* The <MODULE> class: The main class of the module shares the name of the module and is defined in the init.pp file.
* The install class: Contains all of the resources related to installing the software that the module manages.
* The config class: Contains resources related to configuring the installed software.
* The service class: Contains service resources, as well as anything else related to the running state of the software.

### Write your first module:

A [module](https://puppet.com/docs/puppet/5.4/modules_fundamentals.html) in the puppet master contains a bunch of code and resources which can be applied to puppet agents. Modules are shareable, reusable and anyone can write their own module.

To create your own module, navigate to /etc/puppet/code/environments/production/modules directory and create a folder with a valid module name(mymodule).

mkdir /etc/puppet/code/environments/production/modules/mymodule  
cd /etc/puppet/code/environments/production/modules/mymodule  
mkdir files  
mkdir manifests

Put the file(test.txt) inside files folder. Then, create a init.pp file inside manifests directory.

nano manifests/init.pp

Modules have a specific [directory structure](https://puppet.com/docs/puppet/5.4/modules_fundamentals.html#module-structure). Here, we use the following structure where you only have the “files” and “manifests” subfolders.

.  
└── mymodule  
 ├── files  
 │ └── test.txt  
 └── manifests  
 └── init.pp

The “files” folder is to keep the file that the agent is going to download, and the “manifests” to place the init.pp file. The init.pp file contains the definition of the main class of the module.

Include the following in your init.pp file.

class mymodule {

file { '/home/ubuntu':  
 ensure => directory,  
 owner => 'ubuntu',  
 group => 'ubuntu',  
 }

file { "/home/ubuntu/test.txt":  
 mode => "0644",  
 owner => 'ubuntu',  
 group => 'ubuntu',  
 source => 'puppet:///modules/mymodule/test.txt',  
 }  
}

Note:

1. Here, the file will be copied to /home/ubuntu/ directory in the puppet agent node. Make sure the path where you want to copy the file exists.
2. Also, when you give the source, make sure you don’t give the absolute path to the file. Puppet only needs the path to the location of the module which has the file. That is /modules/mymodule/ and then just give the file path as above. If you give the file path as below, you will get an error.

Source:

source => 'puppet:///modules/mymodule/files/test.txt',

Error:

Error:/Stage[main]/Mymodule/File[/home/ubuntu/test.txt]: Could not evaluate: Could not retrieve information from environment production source(s) puppet:///modules/mymodule/files/test.txt

Next, navigate to the [main manifest](https://puppet.com/docs/puppet/5.4/dirs_manifest.html). The main starting point of Puppet is called the main manifest or the site manifest. Compilation of puppet starts with either a single manifest file or a set of manifests in the manifests directory that get treated like a single file.

Create the site.pp file inside main manifests.

cd /etc/puppet/code/environments/production/manifests  
nano /etc/puppet/code/environments/production/manifests/site.pp

Include the following in your site.pp file.

node "puppet-agent" {  
 include mymodule  
}

Note: Here, the “puppet-agent” means the hostname of the puppet agent. You can find out the hostname of the puppet agent either by typing `hostname` in your puppet agent terminal or by checking the /etc/hostname file in your puppet agent. Also, you can use “default” instead of “puppet-agent”. Then the module will be applied to all the existing nodes.

Now you are all set. Try your module by running the following command in your puppet agent terminal and check your /home/ubuntu/ directory to find the test.txt file copied into it.

puppet agent --test

### Deleting the file:

Change your init.pp to delete the file you just copied as follows and run the above command again to delete the file.

class mymodule {

file { "/home/ubuntu/test.txt":  
 ensure => absent,  
 }  
}

**Classes:**

Puppet classes are defined as a collection of resources, which are grouped together in order to get a target node or machine in a desired state. These classes are defined inside Puppet manifest files which is located inside Puppet modules.

1. Defining a class makes it available by name, but doesn't automatically evaluate the code inside it.  
Before we can use a class, we must define it, which is done with the class keyword, a name, curly braces, and a block of code:

1. class my\_class {
2. ... puppet code ...
3. }

This manifest does nothing.

1. Declaring a class evaluates the code in the class, and applies all of its resources.  
   This one actually does something.
2. class my\_class {
3. ... puppet code ...
4. }
5. include my\_class

**Example:** If condition within Puppet exec

exec { 'apply\_new\_cert':

command => "cat /some/file > /other/file",

onlyif => "if [[ `cat /some/file` == `cat /other/file` ]]; then exit 0 ; else exit 1; fi;",

require => Exec['new\_cert'],

path => ['/bin', '/usr/bin', '/sbin', '/usr/sbin', '/usr/loca/bin','],

provider => 'shell',

}

**Assign the output of a unix command to a Puppet Variable and print it.**

$arr = generate('/bin/ls')

notify{"The value is: ${$arr}": } -> Print the Puppet Variable

**Example: Execute a class only if a condition is met, if not notify a message**

if 1 == 2 {

include myclass

}

else {

notify { 'notify user':

message=> 'The UC Master is not reachable',

}

}

class myclass {

notify { 'success':

message => 'UC Master is available and the execution is successful',

}

}

**Puppet Metaparameters:**

Puppet uses four [metaparameters](https://puppet.com/docs/puppet/6.0/lang_resources.html) to establish relationships, and you can set each of them as an attribute in any resource. The value of any relationship metaparameter should be a [resource reference](https://puppet.com/docs/puppet/6.0/lang_data_resource_reference.html) (or [array](https://puppet.com/docs/puppet/6.0/lang_data_array.html) of references) pointing to one or more **target resources**.

* before — Applies a resource **before** the target resource.
* require — Applies a resource **after** the target resource.
* notify — Applies a resource **before** the target resource. The target resource [refreshes](https://puppet.com/docs/puppet/6.0/lang_relationships.html#refreshing-and-notification) if the notifying resource changes.
* subscribe — Applies a resource **after** the target resource. The subscribing resource [refreshes](https://puppet.com/docs/puppet/6.0/lang_relationships.html#refreshing-and-notification) if the target resource changes.

If two resources need to happen in order, you can either put a before attribute in the prior one or a require attribute in the subsequent one; either approach creates the same relationship. The same is true of notify and subscribe.

The two examples below create the same ordering relationship:

package { 'openssh-server':

ensure => present,

before => File['/etc/ssh/sshd\_config'],

}

file { '/etc/ssh/sshd\_config':

ensure => file,

mode => '0600',

source => 'puppet:///modules/sshd/sshd\_config',

require => Package['openssh-server'],

}

The two examples below create the same notifying relationship:

file { '/etc/ssh/sshd\_config':

ensure => file,

mode => '0600',

source => 'puppet:///modules/sshd/sshd\_config',

notify => Service['sshd'],

}

service { 'sshd':

ensure => running,

enable => true,

subscribe => File['/etc/ssh/sshd\_config'],

}

Since an array of resource references can contain resources of differing types, these two examples also create the same ordering relationship:

service { 'sshd':

ensure => running,

require => [

Package['openssh-server'],

File['/etc/ssh/sshd\_config'],

],

}

package { 'openssh-server':

ensure => present,

before => Service['sshd'],

}

file { '/etc/ssh/sshd\_config':

ensure => file,

mode => '0600',

source => 'puppet:///modules/sshd/sshd\_config',

before => Service['sshd'],

}

# How To Set Up a Masterless Puppet Environment on RHEL 7

### Introduction

In the modern world of cloud computing, configuration management is a crucial step. Configuration management tools allow you to reliably deploy configurations to your servers. One of the more mature configuration management tools in this space is [Puppet](https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules).

In a typical Puppet environment, a user writes Puppet modules on their workstation, pushes the modules to a version control server (e.g. Git), then pulls those modules down to a Puppet master. A server running the Puppet client periodically connects to the Puppet master to see if anything has changed, and applies the changes if so.

This scenario works just fine until you have to start scaling up how many servers are checking in or the modules become fairly complex. At that point you have two options: cluster your Puppet Master to handle the load (which will likely require you to buy the commercial version of Puppet), or just drop the Puppet master altogether. This article will look into the second option.

A masterless Puppet setup requires a copy of all Puppet modules to be copied to each node via Git and then have Puppet apply the changes locally. The disadvantage with this method is that each server downloads all of the modules, then applies what is relevant, so it's not the best choice for e.g. setups with sensitive information. However, running without a Puppet master gives you a lot of flexibility and works great without having to scale your infrastructure.

### Prerequisites

If you are new to Puppet, then you may want to pause here to read [this article on Puppet](https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules) first, as this tutorial assumes a working knowledge of the tool. If you're new to Git, you can check out [this introduction to Git series](https://www.digitalocean.com/community/tutorial_series/introduction-to-git-installation-usage-and-branches), too.

In this tutorial, we'll be working with two Droplets: one running as a Git server, and the other that we'll be applying changes to via Puppet. We'll refer to the IP addresses of these Droplets with your\_git\_server\_ip and your\_puppet\_server\_ip respectively.

So, to follow this tutorial, you will need:

* One Ubuntu 14.04 Droplet with a [sudo non-root user](https://www.digitalocean.com/community/tutorials/how-to-add-and-delete-users-on-an-ubuntu-14-04-vps) and [SSH keys added](https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server).
* Another Ubuntu 14.04 Dropet with SSH keys added and Git Labs installed.

The easiest way to set up Git Labs is to use the one click image: on the Droplet creation page under **Select Image**, click the **Applications** tab, then click **GitLab 7.10.0 CE on 14.04**. You can also follow [this tutorial](https://www.digitalocean.com/community/tutorials/how-to-set-up-gitlab-as-your-very-own-private-github-clone) to set up Git Labs manually.

## Step 1 — Creating a Git Repository

The first step is to create a repository where all of our Puppet modules and manifests will be stored.

First, open the Git Labs UI by going to http://your\_git\_server\_ip in your favorite browser. Create an account by filling in the details on the right under **New user? Create an account** and pressing the green **Sign up** button. You'll receive an account activation email, and after activating your account, you'll be able to sign in on the main page.

Click on the green **+ New Project** button on the main page. Enter "puppet" for the **Project path**, and click **Create project**. Enter "puppet" in the **Project path** field, and choose **Public** for the **Visibility Level**, then click the green **Create Project** button.

Make sure you copy the SSH URL, which you'll see toward the top of the project screen, as we'll need it in a later step. It'll look something like git@your\_git\_server\_ip:username/puppet.git.

## Step 2 — Adding an SSH Key to Git Labs

In this step, we will create an SSH key on the Puppet server, then add that key to the Git Labs server.

Log in to the Puppet server as **root**. (Because Puppet's files will be owned by root, we need to have rights to setup the initial Git repo in the Puppet folder.)

Create an SSH key for the root user. Make sure not to enter a passphrase because this key will be used by scripts, not a user.

* ssh-keygen -t rsa

Next, display your public key with the following command.

* cat ~/.ssh/id\_rsa.pub

Copy this key. It will look something like ssh-rsa long\_alphanumeric\_string root@hostname.

Now, on your Git Labs Dashboard page, click on the **Profile settings** icon on the top bar, second from the right. In the left menu, click **SSH Keys**, then click the green **Add an SSH Key** button. In the **Title**, field add a description of the key (like "Root Puppet Key"), and paste your public key into the **Key** field. Finally, click **Add key**.

## Step 3 — Installing Puppet and Git

In this step, we will install Puppet and Git.

On the Puppet server, first download the Puppet package for Ubuntu 14.04.

* wget https://yum.puppetlabs.com/puppetlabs-release-el-7.noarch.rpm

Install the package.

* puppetlabs-release-el-7.noarch.rpm

Update your system's package list.

* apt-get update

Finally, install Puppet and git.

* apt-get install puppet git-core

At this point, you should configure your Git environment by following the instructions in [this tutorial](https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-14-04#how-to-set-up-git).

## Step 4 — Pushing the Initial Puppet Configuration

With Puppet and Git installed, we are ready to do our initial push to our Puppet repository.

First, move to the /etc/puppet directory, where the configuration files live.

* cd /etc/puppet

Initialize a git repository here.

* git init

Add everything in the current directory.

* git add .

Commit these changes with a descriptive comment.

* git commit -m "Initial commit of Puppet files"

Add the Git project we created earlier as origin using the SSH URL you copied in Step 1.

* git remote add origin git@your\_server\_ip:username/puppet.git

And finally, push the changes.

* git push -u origin master

## Step 5 — Cleaning Up Puppet's Configuration

Now that Puppet is installed, we can put everything together. At this point, you can log out as root and instead log in as the sudo non-root user you created during the prerequisites. It isn't good practice to operate as the root user unless absolutely necessary.

To get the foundation in place, we need to make a couple of changes. First, we are going to clean up the /etc/puppet/puppet.conf file. Using your favorite editor (vim, nano, etc.) edit /etc/puppet/puppet.conf with the following changes.

Let's start by making a few changes to the /etc/puppet/puppet.conf file for our specific setup. Open the file using nano or your favorite text editor.

* sudo nano /etc/puppet/puppet.conf

The file will look like this:

Original /etc/puppet/puppet.conf

[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

First, remove everything from the [master] line down, as we aren't running a Puppet master. Also delete the last line in the [main] section which begins with templatedir, as this is deprecated. Finally, change the line which reads factpath=$vardir/lib/facter to factpath=$confdir/facter instead. $confdir is equivalent to /etc/puppet/, i.e. our Puppet repository.

Here is what your puppet.conf should look like once you're finished with the above changes.

Modified /etc/puppet/puppet.conf

[main]

logdir=/var/log/puppet

vardir=/var/lib/puppet

ssldir=/var/lib/puppet/ssl

rundir=/var/run/puppet

factpath=$confdir/facter

## Step 6 — Adding a Puppet Module

Now Puppet is set up, but it's not doing any work. The way Puppet works is by looking at files called manifests that define what it should do, so in this step, we'll create a useful module for Puppet to run.

Our first module, which we will call cron-puppet, will deploy Puppet via Git. It'll install a Git hook that will run Puppet after a successful merge (e.g. git pull), and it'll install a cron job to perform a git pull every 30 minutes.

First, move into the Puppet modules directory.

* cd /etc/puppet/modules

Next, make a cron-puppet directory containing manifests and files directories.

* sudo mkdir -p cron-puppet/manifests cron-puppet/files

Create and open a file called init.pp in the manifests directory.

* sudo nano cron-puppet/manifests/init.pp

Copy the following code into init.pp. This is what tells Puppet to pull from Git every half hour.

init.pp

class cron-puppet {

file { 'post-hook':

ensure => file,

path => '/etc/puppet/.git/hooks/post-merge',

source => 'puppet:///modules/cron-puppet/post-merge',

mode => 0755,

owner => root,

group => root,

}

cron { 'puppet-apply':

ensure => present,

command => "cd /etc/puppet ; /usr/bin/git pull",

user => root,

minute => '\*/30',

require => File['post-hook'],

}

}

Save and close the file, then open another file called post-merge in the files directory.

* sudo nano cron-puppet/files/post-merge

Copy the following bash script into post-merge. This bash script will run after a successful Git merge, and logs the result of the run.

post-merge

#!/bin/bash -e

## Run Puppet locally using puppet apply

/usr/bin/puppet apply /etc/puppet/manifests/site.pp

## Log status of the Puppet run

if [ $? -eq 0 ]

then

/usr/bin/logger -i "Puppet has run successfully" -t "puppet-run"

exit 0

else

/usr/bin/logger -i "Puppet has ran into an error, please run Puppet manually" -t "puppet-run"

exit 1

fi

Save and close this file

Finally, we have to tell Puppet to run this module by creating a global manifest, which is canonically found at /etc/puppet/manifests/site.pp.

* sudo nano /etc/puppet/manifests/site.pp

Paste the following into site.pp. This creates a node classification called 'default'. Whatever is included in the 'default' node will be run on every server. Here, we tell it to run our cron-puppet module.

site.pp

node default {

include cron-puppet

}

Save and close the file. Now, let's make sure our module works by running it.

* sudo puppet apply /etc/puppet/manifests/site.pp

After a successful run you should see some output ending with a line like this.

...

Notice: Finished catalog run in 0.18 seconds

Finally, let's commit our changes to the Git repository. First, log in as the root user, because that is the user with SSH key access to the repository.

Next, change to the /etc/puppet directory.

* cd /etc/puppet

Add everything in that directory to the commit.

* git add .

Commit the changes with a descriptive message.

* git commit -m "Added the cron-puppet module"

Finally, push the changes.

* git push -u origin master

## Conclusion

To add more servers, simply follow step 3 above to install Puppet and Git on the new server, then clone the Git repository to /etc/puppet and apply the site.pp manifest.

You can even automate this installation by using [user data](https://www.digitalocean.com/community/tutorials/an-introduction-to-droplet-metadata) when you create a Droplet. Make sure you use an SSH key when you create the Droplet, and have that SSH key added to your GitLab server. Then just tick the **Enable User Data** checkbox on the Droplet creation screen and enter the following bash script, replacing the variables highlighted in red with your own.

#!/bin/bash -e

## Install Git and Puppet

wget -O /tmp/puppetlabs.deb http://apt.puppetlabs.com/puppetlabs-release-`lsb\_release -cs`.deb

dpkg -i /tmp/puppetlabs.deb

apt-get update

apt-get -y install git-core puppet

# Clone the 'puppet' repo

cd /etc

mv puppet/ puppet-bak

git clone http://your\_git\_server\_ip/username/puppet.git /etc/puppet

# Run Puppet initially to set up the auto-deploy mechanism

puppet apply /etc/puppet/manifests/site.pp

That's all! You now have a masterless Puppet system, and can spin up any number of additional servers without even having to log in to them.

**Steps to update the sudoers file via Puppet**

**Install the saz-sudo module**:

As the root user on the Puppet master, run

/opt/puppetlabs/bin/puppet module install saz-sudo

You should see output similar to the following:

Preparing to install into /etc/puppetlabs/code/environments/production/modules …

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

Notice: Installing -- do not interrupt ...

/etc/puppetlabs/puppet/modules

└── saz-sudo (v2.3.6)

└── puppetlabs-stdlib (3.2.2) [/opt/puppet/share/puppet/modules]

That’s it! You’ve just installed the saz-sudo module.

cd /etc/puppetlabs/code/environments/production/modules

mkdir -p privileges/manifests

cd privileges/manifests

vi init.pp

class privileges {

sudo::conf { 'admins':

ensure => present,

content => '%admin ALL=(ALL) ALL',

}

}

Save and exit the file.

That’s it! You’ve written a module that contains a class that, once applied, ensures that your agents have the correct sudo privileges set for the root user and the “admins” and “wheel” groups.

Note the following about the resource in the privileges class:

* The sudo::conf ‘admins’ line creates a sudoers rule to ensure that members of the admins group have the ability to run any command using sudo. This resource creates configuration fragment file to define this rule in /etc/sudoers.d/. It will be called something like 10\_admins.

cd /etc/puppetlabs/code/environments/production/manifests

vi site.pp

class { 'sudo': }

sudo::conf { 'web':

content => "web ALL=(ALL) NOPASSWD: ALL",

}

class { 'sudoprivilege': }

sudo::conf { 'jargyle':

priority => 60,

content => "jargyle ALL=(ALL) NOPASSWD: ALL",

}

puppet parser validate site.pp

puppet agent -t

useradd raju

passwd raju

groupadd aws

usermod -a raju -G aws