Quick start » Intro for *nix users

> Included in Puppet Enterprise 2017.2.

Welcome to the Open Source Puppet Quick Start Guide. Whether you're setting up a Puppet installation for a real deployment or simply want to learn some fundamentals of configuration management with Open Source Puppet, this series of guides provides the steps you need to get up and running relatively quickly. We'll walk you through Puppet installation and show you how to automate some basic tasks that sysadmins regularly perform.

The following guides present tasks in the order that you would most likely perform them. See the prerequisite sections in each guide to ensure you have the correct setup to perform the steps as they're provided:

1. Perform pre-install tasks

Follow these instructions to ensure you meet the system requirements for Puppet, to designate servers, to decide on a deployment type, and more.

2. Install Puppet

Next, you'll install and configure your Puppet master and agents.

A computer that runs the Puppet Server is called the "master." Follow these instructions to install and configure Puppet Server.

A computer that runs the Puppet agent is called a "Puppet agent" or simply "agent". The Puppet agent regularly pulls configuration catalogs from a master and applies them to the local system.

Follow these instructions to install a Puppet agent on Windows or *nix.

To learn how to get your Puppet master and agents to communicate with each other and to ensure your Puppet master will receive certificates from its agents, follow the instructions in the Master/Agent Communication Quick Start Guide.

3. Create a user and group

Learn how to create a Puppet user and group with these instructions.

Instructions are available for *nix only.

4. Hello, world!

Modules contain classes, which are named chunks of Puppet code and are the primary means by which Puppet configures and manages nodes. The instructions in the Hello World! Quick Start Guide lead you through the fundamentals of Puppet module writing. You'll write a very simple module that contains classes to manage your message of the day (motd) and create a Hello, World! notification on the command line.

5. Install a module

Next, learn how to install a Puppet module by following the Module Installation Quick Start Guide.

The instructions are written specifically for *nix, but the installation process is the same for Windows.

6. Add classes

Follow the Adding Classes Quick Start Guide to add a class to your module. The class you'll install is derived from the module you installed in the Module Installation Quick Start Guide.

Instructions are available for *nix only.

7. Write modules

Follow the Writing Modules Quick Start Guide for exercises in writing modules to help you become more familiar with Puppet modules and module development.

Instructions are available for *nix only.

Installing Puppet: Pre-install tasks

> Included in Puppet Enterprise 2017.2.

To ease your Puppet installation, complete these tasks before installing Puppet agent.

> **Note:** This document covers open source releases of Puppet. For instructions on installing Puppet Enterprise, see its installation documentation.

1. Decide on a deployment type.

   Puppet usually uses an agent/master (client/server) architecture, but it can also run in a self-contained architecture. Your choice determines which packages you install, and what extra configuration you need to do.

   Additionally, consider using PuppetDB, which enables extra Puppet features and makes it easy to query and analyze Puppet's data about your infrastructure.

   Learn more about Puppet's architectures here.

2. If you choose the standard agent/master architecture, you need to decide which server(s) acts as the Puppet master (and the PuppetDB server, if you choose to use it).

   Completely install and configure Puppet on any Puppet masters and PuppetDB servers before installing on any agent nodes. The master must be running some kind of *nix. Windows machines can't be masters.

   A Puppet master is a dedicated machine, so it must be reachable at a reliable hostname. Agent nodes default to contacting the master at the hostname `puppet`. If you make sure this hostname resolves to the master, you can skip changing the `server` setting and reduce your setup time.

3. Check OS versions and system requirements.

   See the system requirements for the version of Puppet you are installing, and consider the following:

   - Your Puppet master(s) should be able to handle the amount of agents they'll need to serve.
   - Systems we provide official packages for have an easier install path.
   - Systems we don't provide packages for might still be able to run Puppet, as long as the version of Ruby is suitable and the prerequisites are installed, but it means a

more complex and often time consuming install path.

4. Check your network configuration.

   In an agent/master deployment, you must prepare your network for Puppet's traffic.

   - **Firewalls:** The Puppet master server must allow incoming connections on port 8140, and agent nodes must be able to connect to the master on that port.
   - **Name resolution:** Every node must have a unique hostname. **Forward and reverse DNS** must both be configured correctly. If your site lacks DNS, you must write an `/etc/hosts` file on each node.
     - **Note:** The default Puppet master hostname is `puppet`. Your agent nodes can be ready sooner if this hostname resolves to your Puppet master.
5. Set timekeeping on your Puppet master server.

   The time must be set accurately on the Puppet master server that acts as the certificate authority. If the time is wrong, it can mistakenly issue agent certificates from the distant past or future, which other nodes treat as expired. There are modules in the forge, such as the ntp module that can help you with this.

Install Puppet Server before installing Puppet on your agent nodes. If you're using PuppetDB, install it once Puppet Server is up and running. Once you have completed these steps and configured your master, you can install Puppet agent.

Puppet Server: Installing From Packages

> Included in Puppet Enterprise 2017.2.

System Requirements

Puppet Server is configured to use 2 GB of RAM by default. If you'd like to just play around with an installation on a Virtual Machine, this much memory is not necessary. To change the memory allocation, see Memory Allocation.

> **Note:** Puppet masters running Puppet Server 2.6 depend on Puppet Agent 1.6.0 or newer, which installs Puppet 4.6 and compatible versions of its related tools and dependencies on the server. Puppet agents running older versions of Puppet Agent can connect to Puppet Server 2.6 — this requirement applies to the Puppet Agent running on the Puppet Server node *only*.
>
> If you're also using PuppetDB, check its requirements.

Platforms with Packages

Puppet provides official packages that install Puppet Server 2.4 and all of its prerequisites on the following platforms, as part of Puppet Collections.

Red Hat Enterprise Linux

- Enterprise Linux 6
- Enterprise Linux 7

Debian

- Debian 7 (Wheezy)
- Debian 8 (Jessie)

Ubuntu

- Ubuntu 12.04 (Precise)
- Ubuntu 14.04 (Trusty)
- Ubuntu 15.10 (Wily)
- Ubuntu 16.04 (Xenial)

SuSE Linux Enterprise Server

- SLES 12

Quick Start

1. Enable the Puppet package repositories, if you haven't already done so.
2. Stop the existing Puppet master service. The method for doing this varies depending on how your system is set up.

   If you're running a WEBrick Puppet master, use: `service puppetmaster stop`.

   If you're running Puppet under Apache, you'll instead need to disable the puppetmaster vhost and restart the Apache service. The exact method for this depends on what your Puppet master vhost file is called and how you enabled it. For full documentation, see the Passenger guide.

   - On a Debian system, the command might be something like `sudo a2dissite puppetmaster`.
   - On RHEL/CentOS systems, the command might be something like `sudo mv /etc/httpd/conf.d/puppetmaster.conf ~/`. Alternatively, you can delete the file instead of moving it.

   After you've disabled the vhost, restart Apache, which is a service called either `httpd` or `apache2`, depending on your OS.

   Alternatively, if you don't need to keep the Apache service running, you can stop Apache with `service httpd stop` or `service apache2 stop`.

3. Install the Puppet Server package by running:

```
yum install puppetserver
```

```
```

Or

```
apt-get install puppetserver
```

Note that there is no `-` in the package name.

4. Start the Puppet Server service:

```
systemctl start puppetserver
```

Or

```
service puppetserver start
```

Platforms without Packages

For platforms where no official packages are available, you can build Puppet Server from source. Such platforms are not tested, and running Puppet Server from source is not recommended for production use.

For details, see Running from Source.

Memory Allocation

By default, Puppet Server is configured to use 2GB of RAM. However, if you want to experiment with Puppet Server on a VM, you can safely allocate as little as 512MB of memory. To change the Puppet Server memory allocation, you can edit the init config file.

Location

- `/etc/sysconfig/puppetserver` — RHEL
- `/etc/default/puppetserver` — Debian

1. Open the init config file:

```
# Modify this if you'd like to change the memory allocation, en
JAVA_ARGS="-Xms2g -Xmx2g"
```

Replace 2g with the amount of memory you want to allocate to Puppet Server. For example, to allocate 1GB of memory, use `JAVA_ARGS="-Xms1g -Xmx1g"` ; for 512MB, use `JAVA_ARGS="-Xms512m -Xmx512m"` .

For more information about the recommended settings for the JVM, see Oracle’s docs on JVM tuning.

2. Restart the `puppetserver` service after making any changes to this file.

Installing Puppet agent: Linux

> Included in Puppet Enterprise 2017.2.

About release packages

Install the Puppet agent so that your master can communicate with your Linux nodes.

Before you begin: Review the pre-install tasks and installing Puppet Server. If you're familiar with Puppet 3 and earlier, learn about new locations for many of the files and directories by reading a summary of changes in Puppet 4 or referring to the full specification of Puppet directories.

1. Install a release package to enable Puppet Collection repositories.
2. Confirm that you can run Puppet executables.

   The location for Puppet's executables is `/opt/puppetlabs/bin/` , which is not in your `PATH` environment variable by default.

   The executable path doesn't matter for Puppet services — for instance, `service puppet start` works regardless of the `PATH` — but if you're running interactive `puppet` commands, you must either add their location to your `PATH` or execute them using their full path.

   To quickly add the executable location to your `PATH` for your current terminal session, use the command `export PATH=/opt/puppetlabs/bin:$PATH` . You can also add this location wherever you configure your `PATH` , such as your `.profile` or `.bashrc` configuration files.

   For more information, see details about files and directories moved in Puppet 4.
3. Install the `puppet-agent` package on your Puppet agent nodes using the command appropriate to your system:
   - Yum – `sudo yum install puppet-agent` .
   - Apt – `sudo apt-get install puppet-agent` .
4. (Optional) Configure agent settings.

   For example, if your master isn't reachable at the default address, `server = puppet` , set the `server` setting to your Puppet master's hostname.

   For other settings you might want to change, see a list of agent-related settings.

5. Start the `puppet` service: `sudo /opt/puppetlabs/bin/puppet resource service puppet ensure=running enable=true`.
6. (Optional) To see a sample of Puppet agent's output and verify any changes you may have made to your configuration settings in step 5, manually launch and watch a Puppet run: `sudo /opt/puppetlabs/bin/puppet agent --test`
7. Sign certificates on the certificate authority (CA) master.

   On the Puppet master:

   a. Run `sudo /opt/puppetlabs/bin/puppet cert list` to see any outstanding requests.
   b. Run `sudo /opt/puppetlabs/bin/puppet cert sign <NAME>` to sign a request.

   As each Puppet agent runs for the first time, it submits a certificate signing request (CSR) to the CA Puppet master. You must log into that server to check for and sign certificates. After an agent's certificate is signed, it regularly fetches and applies configuration catalogs from the Puppet master.

About release packages

Release packages configure your system to download and install appropriate versions of the `puppetserver` and `puppet-agent` packages. These packages are grouped into a Puppet Collection repository comprised of compatible versions of Puppet tools.

> **Note:** This document covers the Puppet Collection repository of open source Puppet 4-compatible software packages.
>
> - For Puppet 3.8 open source packages, see its repository documentation.
> - For Puppet Enterprise installation tarballs, see its installation documentation.

Puppet maintains official package repositories for several operating systems and distributions. To make the repositories more predictable, we version them as "Puppet Collections" — each collection has all of the software you need to run a functional Puppet deployment, in versions that are known to work well with each other. Each collection is opt-in, and you must choose one (and on some operating systems, install a package on Puppet-managed systems) to install software and receive updates.

Collection repositories are organized into two tiers that correspond to Puppet Enterprise releases, which are downstream from the collection's open-source components:

- **Numbered collections, such as Puppet Collection 1 (PC1),** are long-lived, stable repositories from which long term support (LTS) Puppet Enterprise releases are built. Numbered collections maintain the same major version of each component package during its lifetime, which delivers bug fixes and minimizes breaking changes, but also introduces fewer new features.
- **The "latest" collection** follows every release of Puppet Enterprise, including versions not considered LTS releases, and is updated with new major-version releases that might

introduce breaking changes.

Puppet publishes updates for operating systems starting from the time a package is first published for that operating system to a collection repository, and stops updating those packages 30 days after the end of the operating system's vendor-determined lifespan.

See The Puppet Enterprise Lifecycle for information about phases of the Puppet Support Lifecycle.

To receive the most up-to-date Puppet software without introducing breaking changes, use the `latest` collection, pin your infrastructure to known versions, and update the pinned version manually when you're ready to update. For example, if you're using the `puppetlabs-puppet_agent` module to manage the installed `puppet-agent` package, use this resource to pin it to version 1.7.0:

```
class { '::puppet_agent':
  collection      => 'latest',
  package_version => '1.7.0',
}
```

When `puppet-agent` 2.0.0 is released, update `package_version` when you're ready to upgrade to that version:

```
class { '::puppet_agent':
  collection      => 'latest',
  package_version => '2.0.0',
}
```

Package	Contents
`puppet-agent`	Puppet, Facter, Hiera, MCollective, `pxp-agent`, root certificates, and prerequisites like Ruby and Augeas
`puppetserver`	Puppet Server; depends on `puppet-agent`
`puppetdb`	PuppetDB
`puppetdb-termini`	Plugins to let Puppet Server talk to PuppetDB

Yum-based systems:

To enable the Puppet Collection 1 repository, first choose the package based on your operating system and version. The packages are located in the `yum.puppetlabs.com` repository and named using the following convention:

```
puppetlabs-release-<COLLECTION>-<OS ABBREVIATION>-<OS VERSION>.noar
```

For instance, the package for Puppet Collection 1 on Red Hat Enterprise Linux 7 (RHEL 7) is `puppetlabs-release-pc1-el-7.noarch.rpm`.

Next, use the `rpm` tool as root with the `upgrade` (`-U`) flag, and optionally the `verbose` (`-v`), and `hash` (`-h`) flags:

```
sudo rpm -Uvh https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7
```

The `rpm` tool outputs its progress:

```
Retrieving https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.nc
Preparing...                          ###############################
Updating / installing...
1:puppetlabs-release-pc1-0.9.2-1.el##################################
```

Note: We only provide the `puppet-agent` package for recent versions of Puppet on RHEL 5, and to install it you must first download the package as `rpm` on RHEL 5, as it doesn't support installing packages from a URL.

Apt-based systems:

To enable the Puppet Collection 1 repository, first choose the package based on your operating system and version. The packages are located in the `apt.puppetlabs.com` repository and named using the following convention:

```
puppetlabs-release-<COLLECTION>-<VERSION CODE NAME>.deb
```

For instance, the release package for Puppet Collection 1 on Debian 7 "Wheezy" is `puppetlabs-release-pc1-wheezy.deb`. For Ubuntu releases, the code name is the adjective, not the animal.

Next, download the release package and install it as root using the `dpkg` tool and the `install` flag (`-i`):

```
wget https://apt.puppetlabs.com/puppetlabs-release-pc1-wheezy.deb
sudo dpkg -i puppetlabs-release-pc1-wheezy.deb
```

Finally, run `apt-get update` after installing the release package to update the `apt` package lists.

Open source Puppet quick start guide series

> Included in Puppet Enterprise 2017.2.

Overview

This guide walks you through the process to make sure your Puppet master and agents are able to communicate. This involves modifying the `/etc/hosts` file on your master and agents, and also opening the firewall to your master so that it is able to sign certificates from the agents.

> **Prerequisites**: This guide assumes you've already installed Puppet, and have installed at least one *nix agent.
>
> For this walk-through, log in as root or administrator on your nodes.

Modifying the `/etc/hosts` files

To make sure your Puppet master and agents communicate, update the `/etc/hosts` file on each so that they're aware of each other. First, use your text editor to open `/etc/hosts` on your Puppet master. Add each of your agents by IP address and name below the existing text. It should look something like this:

```
192.168.33.11    agent1.example.com
```

Next, add the name and IP address of your Puppet master to each of your Puppet agents. Use your text editor to open `/etc/hosts` on your Puppet agent and add the IP address and name of your Puppet master below the existing text, as well as the alias `puppet`. It should look similar to this:

```
192.168.33.10    master.example.com puppet
```

Repeat this step for all of your Puppet agents.

> Congratulations! You've successfully made sure your Puppet master and agents can communicate.

Opening port 8140 on your firewall

For your Puppet master to sign an agent certificate, the agent needs to be able to connect to the master's firewall through port 8140. You will learn to set full firewall rules later in the Quick Start Guide.

WARNING: These next steps open the port 8140 in your firewall. This does create a security risk, as you will need to keep port 8140 open so that the master and agents can continue to communicate.

From the command line on your Puppet master, run:

```
iptables -I INPUT -m state --state NEW -m tcp -p tcp --dport
```

From the command line on each Puppet agent, run `puppet agent -t`.

From your Puppet master, run `puppet cert list` and then `puppet cert sign <AGENT NAME>` to sign the certificates of your Puppet agents.

> That's it! Your Puppet configuration is ready to go.

Users and groups quick start guide

Included in Puppet Enterprise 2017.2.

Before you begin

Prerequisites: This guide assumes you've already installed Puppet, and have installed at least one *nix agent.

For this walk-through, log in as root or administrator on your nodes.

Create a user and group

Puppet uses some defaults for unspecified user and group attributes, so all you'll need to do to create a new user and group is set the 'ensure' attribute to 'present'. This 'present' value tells Puppet to check if the resource exists on the system, and to create the specified resource if it does not.

1. To create a user named `jargyle`, on your Puppet master, run `puppet apply -e "user { 'jargyle': ensure => present, }"`. The result should show, in part, `Notice: /Stage[main]/Main/User[jargyle]/ensure: created`.
2. To create a group named `web`, on your Puppet master, run `puppet apply -e "group { 'web': ensure => present, }"`. The result should show, in part, `Notice: /Stage[main]/Main/Group[web]/ensure: created`.

That's it! You've successfully created the Puppet user `jargyle` and the Puppet group `web`.

Add the group to the main manifest

1. From the command line on your Puppet master, run `puppet resource -e group web`. This opens a file in your text editor with the following content:

   ```
   group { 'web':
                                 ensure => 'present',
                                 gid    => '502',
   }
   ```

   > **Note**: Your gid (the group ID) might be a different number than the example shown in this guide.

2. Copy the lines of code, and save and exit the file.
3. Navigate to your main manifest: `cd /etc/puppetlabs/code/environments/production/manifests`.
4. Still using the Puppet master, paste the code you got from Steps 1 and 2 into the default node `site.pp`, then save and exit.
5. From the command line on your Puppet master, run `puppet parser validate site.pp` to ensure that there are no errors. The parser will return nothing if there are no errors.
6. From the command line on your Puppet agent, use `puppet agent -t` to trigger a Puppet run.

> That's it! You've successfully added your group, `web`, to the main manifest.

Add the user to the main manifest

1. From the command line on your Puppet master, run `puppet resource -e user jargyle`. This opens a file in your text editor with the following content:

   ```
   user { 'jargyle':
                                 ensure           => 'present',
     gid              => '501',
     home             => '/home/jargyle',
     password         => '!!',
     password_max_age => '99999',
     password_min_age => '0',

     shell            => '/bin/bash',
     uid              => '501',
   }
   ```

```
      }
```

> **Note**: Your uid (the user ID), or gid (the group ID) might be different numbers than the examples shown in this guide.

2. Add the following Puppet code to the file:

```
        comment             => 'Judy Argyle',
        groups              => 'web',
```

3. **Delete** the following Puppet code from the file:

```
          gid                 => '501',
```

4. Copy all of the code, and save and exit the file.
5. Paste the code from Step 10 into your default node in `site.pp`. It should look like this:

```
      user { 'jargyle':
                                    ensure            => 'present',
        home                => '/home/jargyle',
        comment              => 'Judy Argyle',
        groups               => 'web',
        password            => '!!',
        password_max_age => '99999',
        password_min_age => '0',
        shell               => '/bin/bash',
        uid                 => '501',
      }
```

6. From the command line on your Puppet master, run `puppet parser validate site.pp` to ensure that there are no errors. The parser will return nothing if there are no errors.
7. From the command line on your Puppet agent, use `puppet agent -t` to trigger a Puppet run.

> Success! You have created a user, `jargyle`, and added jargyle to the group with `groups => web`. For more information on users and groups, check out the documentation for Puppet resource types regarding users and groups. With users and groups, you can assign different permissions for managing Puppet.

Next: Hello, world!

Hello world! Quick start guide

Included in Puppet Enterprise 2017.2.

Overview

The following Quick Start Guide introduces the essential components of Puppet module writing. In this guide, you will write a simple *nix-based module that contains two classes—one that manages your message of the day (motd) and one that creates a notification on the command line when you run Puppet.

While the module you'll write doesn't have an incredible amount of functionality, you'll learn the basic module directory structure and how to apply classes to the main manifest. You'll encounter more complex module writing scenarios in other quick start guides.

For this walk-through, log in as root or administrator on your nodes.

Write the `helloworld` class

Some modules can be large, complex, and require a significant amount of trial and error. This module will be a very simple module to write; it contains just two classes.

A quick note about modules

By default, Puppet keeps modules in an environment's `modulepath`, which for the production environment defaults to `/etc/puppetlabs/code/environments/production/modules`. This includes modules that Puppet installs, those that you download from the Forge, and those you write yourself.

Note: Puppet also creates another module directory: `/opt/puppetlabs/puppet/modules`. Don't modify or add anything in this directory, including modules of your own.

Modules are directory trees. For this task, you'll create the following structure and files:

- `helloworld/` (the module name)
  - `manifests/`
    - `init.pp` (manifest file that contains the `helloworld` class)
    - `motd.pp` (manifest file that contains a file resource that ensures the creation of the motd)

Every manifest (.pp file) in a module contains a single class. File names map to class names in a predictable way, described in the Autoloader Behavior documentation. The `init.pp` file is a special case that contains a class named after the module, `helloworld`. Other manifest files contain classes called `<MODULE NAME>::<FILE NAME>`, or in this case, `helloworld::motd`.

- For more on how modules work, see Module Fundamentals in the Puppet documentation.
- For more on best practices, methods, and approaches to writing modules, see the Beginners Guide to Modules.

To write the `helloworld` class:

1. From the command line on the Puppet master, navigate to the modules directory: `cd /etc/puppetlabs/code/environments/production/modules`.
2. Run `mkdir -p helloworld/manifests` to create the new module directory and its manifests directory.
3. In the `manifests` directory, use your text editor to create the `init.pp` file, and edit it so that it contains the following Puppet code:

   ```
   class helloworld {
      notify { 'hello, world!': }
   }
   ```

4. Save and exit the file.
5. In the `manifests` directory, use your text editor to create the `motd.pp` file, and edit it so that it contains the following Puppet code:

   ```
   class helloworld::motd {

      file { '/etc/motd':
      owner  => 'root',
      group  => 'root',
      mode    => '0644',
      content => "hello, world!\n",
      }

   }
   ```

6. Save and exit the file.

> Hooray! You've written a module that contains two classes that will, once applied, show a notification message when Puppet runs, and manage the motd on your server.

Add the `helloworld` and `helloworld::motd` classes to the main manifest

For this procedure, you're going to add the `helloworld` classes to the default node in the main manifest. You will be using the default node throughout the Quick Start Guide.

The default node is a special value for node names. If no node statement matching a given node name can be found, the default node will be used, making it an easy way to ensure compilation for any node will be successful. In Puppet, a given agent will only get the contents of one node definition. In order to simplify this process, and ensure that compilations are always successful, this guide will consistently use the `default` node in the `site.pp` manifest. The default node's properties apply to all the agents which have not had definitions applied to them yet, so in the case of this guide, the contents of the default node will apply to all of your agents.

To create the default node

1. From the command line on the Puppet master, navigate to the main manifest: `cd /etc/puppetlabs/code/environments/production/manifests`.
2. Use your text editor to create the `site.pp` file, and edit it so that it contains the following Puppet code:

```
node default {

}
```

3. Add the following Puppet code within `node default { }`:

```
class { 'helloworld': }
class { 'helloworld::motd': }
```

4. Save and exit the file.
5. Ensure that there are no errors in the Puppet code by running `puppet parser validate site.pp` on your Puppet master. The parser will return nothing if there are no errors. If it does detect a syntax error, open the file again and fix the problem before continuing.
6. From the CLI of your Puppet agent, use `puppet agent -t` to trigger a Puppet run.

Viewing the results

After you kick off the puppet run, you will see the following on the command line as the `helloworld` class is applied:

```
        [root@agent1 ~]# puppet agent -t
        Info: Retrieving pluginfacts
        Info: Retrieving plugin
        Info: Loading facts
        Info: Caching catalog for agent1.example.com
        Info: Applying configuration version '1437172035'
        Notice: hello, world!
        Notice: /Stage[main]/Main/Node[default]/Notify[hello, world!]/message: de
        Notice: Applied catalog in 1.25 seconds
```

From the command line of your agent, run `cat /etc/motd`. The result should show `hello, world!`

Other resources

There are plenty of resources about modules and the creation of modules that you can reference. Check out Module Fundamentals, the Beginner's Guide to Modules, and the Puppet Forge.

Check out the remainder of the Quick Start Guide series for additional module writing exercises.

Next: Installing Modules (*nix)

Module installation quick start guide

> Included in Puppet Enterprise 2017.2.

Overview

In this guide, you'll install the puppetlabs-apache module, a Puppet-supported module. Modules contain **classes**, which are named chunks of Puppet code and are the primary means by which Puppet configures and manages nodes. In the Module Writing Basics for Linux Quick Start Guide you'll learn more about modules, including customizing and writing your own modules on *nix platforms.

The process for installing a module is the same on both Windows and *nix operating systems.

> **Prerequisites**: This guide assumes you've already installed Puppet, and have installed at least one *nix agent.

> Before starting this walk-through, complete the Hello World exercise in the introductory quick start guide. You should still be logged in as root or administrator on your nodes.

Installing a Forge module

1. **On the Puppet master**, run `puppet module search apache`. This command searches for modules from the Puppet Forge with `apache` in their names or descriptions.

   The search results will display:

```
Searching http://forgeapi.puppetlabs.com ...
NAME                    DESCRIPTION                             AU
puppetlabs-apache       Puppet module for apache                @pu
```

To view detailed information about the module, see the Apache module on Forge.

2. To install the Apache module, run: `puppet module install puppetlabs-apache` . The result looks like this:

   ```
   Preparing to install into /etc/puppetlabs/code/environments/pro
   Notice: Downloading from http://forgeapi.puppetlabs.com ...
   Notice: Installing -- do not interrupt ...
   /etc/puppetlabs/code/environments/production/modules
   └── puppetlabs-apache (v1.1.1)
   ```

That's it! You have installed a Puppet module. All of the classes in the module are now available to be assigned to nodes.

A quick note about module directories

By default, Puppet keeps modules in an environment's `modulepath` , which for the production environment defaults to `/etc/puppetlabs/code/environments/production/modules` . This includes modules that Puppet installs, those that you download from the Forge, and those you write yourself.

Note: Puppet also creates another module directory: `/opt/puppetlabs/puppet/modules` . Don't modify or add anything in this directory, including modules of your own.

`puppetlabs-apache` is a PE-supported module. It is tested and maintained by Puppet, and Puppet Enterprise users are able to file support escalations on these modules.

Next: Adding classes to Puppet agents (*nix)

Adding Classes Quick Start Guide

> Included in Puppet Enterprise 2017.2.

Overview

Every module contains one or more **classes**. Classes are named chunks of Puppet code and are the primary means by which Puppet configures nodes. The puppetlabs-apache module you installed in the Module Installation Quick Start Guide contains a class called `apache`. In this example, you will:

- Use the `apache` class to launch the default Apache virtual host
- Edit class parameters in the main manifest

> **Prerequisites**: This guide assumes you've already installed Puppet, and have installed at least one *nix agent and the puppetlabs-apache module.

> Before starting this walk-through, complete the previous exercises in the introductory quick start guide. You should still be logged in as root or administrator on your nodes.

Add Apache to the main manifest

1. From the command line of your Puppet master, navigate to the main manifest directory: `cd /etc/puppetlabs/code/environments/production/manifests`.
2. Use your text editor to open the `site.pp` file, and edit it so that it contains the following Puppet code:

```
node default {
  include apache
}
```

> **Note**: If you have already created the default node class, simply add `include apache` to it. Code from the Hello World! exercise does not need to be removed, but a class cannot be declared twice. We will explore this later in the guide.

3. Ensure that there are no errors in the Puppet code by running `puppet parser validate site.pp` on the command line of your Puppet master. The parser will return nothing if there are no errors. If it does detect a syntax error, open the file and fix the problem before continuing.
4. From the command line of your Puppet agent, run `puppet agent -t` to trigger a Puppet run.

Create the index.html file

1. **On the Puppet agent**, navigate to `/var/www/html`, and create a file called `index.html` if it does not already exist.
2. Open `index.html` in your text editor and fill it with some content (for example, "Hello World") or edit what is already there.
3. From the command line of your Puppet agent, run `puppet agent -t`.
4. Open a web browser and enter the IP address for the Puppet agent, adding port 80 on the end, as in `http://myagentnodeIP:80/`.

   You will see the contents of `/var/www/html/index.html` displayed.

Editing class parameters in the main manifest

You can edit the parameters of a class in `site.pp` as well. Parameters allow a class to request external data. If a class needs to configure itself with data other than Puppet facts, provide that data to the class via a parameter.

To edit the parameters of the `apache` class:

1. From the command line of your Puppet master, navigate to `/etc/puppetlabs/code/environments/production/manifests`.
2. Use your text editor to open `site.pp`.
3. Replace the `include apache` command with the following Puppet code:

```
class { 'apache':
        docroot => '/var/www'
}
```

> **Note**: You must remove `include apache` because Puppet will only allow you to declare a class once.

> That's it! You have set the Apache web server's root directory to `/var/www` instead of its default `/var/www/html`. If you refresh `http://myagentnodeIP:80/` in your web browser, it shows the list of files in `/var/www`. If you click `html`, the browser will again show the contents of `/var/www/html/index.html`.
>
> **Note:** If you have Puppet Enterprise, you can do the steps in this guide through the PE web UI, the console.

Next: Quick Start: Writing Modules

Module writing basics for *nix

> Included in Puppet Enterprise 2017.2.

Welcome to the Module Writing section of the Quick Start Guide series. This walk-through will help you become more familiar with Puppet modules and module development. Follow along to learn how to:

- Modify a module obtained from the Forge
- Write your own Puppet module
- Create a site module that composes other modules into machine roles

> Before starting this walk-through, complete the previous exercises in the introductory quick start guide. These steps assume that you have installed Puppet and Puppet agents, and have installed the latest version of the puppetlabs-apache module. You should still be logged in as root or administrator on your nodes.

Editing a Forge module

Although many Forge modules are exact solutions that fit your site, many are *almost* but not quite what you need. Sometimes you will need to edit some of your Forge modules.

Module basics

> #### About module directories
>
> By default, Puppet keeps modules in an environment's `modulepath`, which for the production environment defaults to `/etc/puppetlabs/code/environments/production/modules`. This includes modules that Puppet installs, those that you download from the Forge, and those you write yourself. In a fresh installation, you need to create this `modules` subdirectory yourself by navigating to `/etc/puppetlabs/code/environments/production` and running `mkdir modules`.
>
> **Note:** Puppet also creates another module directory: `/opt/puppetlabs/puppet/modules`. Don't modify or add anything in this directory, including modules of your own.
>
> There are plenty of resources about modules and the creation of modules that you can reference. Check out Module Fundamentals, the Beginner's Guide to Modules, and the Puppet Forge.

Modules are directory trees. For these exercises you'll use the following files:

- `apache/` (the module name)
  - `manifests/`
    - `init.pp` (contains the `apache` class)
  - `templates/`
    - `vhost/`
      - `_file_header.erb` (contains the vhost template, managed by Puppet)

Every manifest (.pp file) in a module contains a single class. File names map to class names in a predictable way, described in the Autoloader Behavior documentation. The `init.pp` file is a special case that contains a class named after the module, `apache`. Other manifest files contain classes called `<MODULE NAME>::<FILE NAME>` or `<MODULE NAME>::<FOLDER>::<FILE NAME>`. Many modules, including Apache, contain directories other than `manifests` and `templates`. For simplicity's sake, we do not cover them in this introductory guide.

- For more on how modules work, see Module Fundamentals in the Puppet documentation.
- For more on best practices, methods, and approaches to writing modules, see the Beginners Guide to Modules.

Writing a Puppet module

In this simplified exercise, you'll modify a template from the Puppet Apache module, specifically `vhost.conf.erb`, to include some simple variables that will be populated by facts (using Puppet's implementation of Facter) about your node.

1. **On the Puppet master,** navigate to the modules directory by running `cd /etc/puppetlabs/code/environments/production/modules`.
2. Run `ls` to view the currently installed modules, and note that `apache` is present.
3. Open `apache/templates/vhost/_file_header.erb` in a text editor. Avoid using Notepad because it can introduce errors. `_file_header.erb` contains the following header:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
# ************************************
```

4. Collect the following facts about your agent:
   - on your Puppet agent, run `facter osfamily`. This returns your agent's OS.
   - on your Puppet agent, run `facter id`. This returns the id of the currently logged in user.
5. Edit the header of `_file_header.erb` so that it contains the following variables for Facter lookups:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
#
# This file is authorized for deployment by <%= scope.lookupvar('::id') %>
#
# This file is authorized for deployment ONLY on <%= scope.lookupvar('::os
#
# Deployment by any other user or on any other system is strictly prohibit
# ************************************
```

6. From the command line of your Puppet agent, run `puppet agent -t` to trigger a Puppet run.

At this point, Puppet configures Apache and starts the httpd service. When this happens, a default Apache virtual host is created based on the contents of `_file_header.erb`.

1. **On the agent**, navigate to one of the following locations based on your operating system:
   - Redhat-based: `/etc/httpd/conf.d`
   - Debian-based: `/etc/apache2/sites-available`
2. View `15-default.conf`. Depending on the node's OS, the header will show some variation of the following contents:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
#
# This file is authorized for deployment by root.
#
# This file is authorized for deployment ONLY on Redhat 6.
#
# Deployment by any other user or on any other system is strictly prohibit
# ************************************
```

As you can see, Puppet has used Facter to retrieve some key facts about your node, and then used those facts to populate the header of your vhost template.

But now, let's see what happens when you write your own Puppet code.

Writing a Puppet module

Puppet modules save time, but at some point you may need to write your own modules.

Writing a class in a module

In this exercise, you will create a class called `puppet_quickstart_app` that will manage a PHP-based web app running on an Apache virtual host.

1. **On the Puppet master**, make sure you're still in the modules directory (`cd /etc/puppetlabs/code/environments/production/modules`) and then run `mkdir -p puppet_quickstart_app/manifests` to create the new module directory and its manifests directory.
2. Use your text editor to create and open the `puppet_quickstart_app/manifests/init.pp` file.
3. Edit the `init.pp` file so it contains the following Puppet code, and then save it and exit the editor:

```
class puppet_quickstart_app {

  class { 'apache':
    mpm_module => 'prefork',
  }

  include apache::mod::php

  apache::vhost { 'puppet_quickstart_app':
    port     => '80',
    docroot  => '/var/www/puppet_quickstart_app',
    priority => '10',
  }

  file { '/var/www/puppet_quickstart_app/index.php':
    ensure  => file,
    content => "<?php phpinfo() ?>\n",
    mode    => '0644',
  }

}
```

You have written a new module containing a new class that includes two other classes: `apache` and `apache::mod::php`.

Note the following about your new class:

- The class `apache` has been configured to include the `mpm_module` attribute. This attribute determines which multi-process module is configured and loaded for the Apache (HTTPD) process. In this case, the value is set to `prefork`.
- `include apache::mod::php` indicates that your new class relies on those classes to function correctly. Puppet understands that your node needs to be classified with these classes and will take care of that work automatically when you classify your node with the `puppet_quickstart_app` class. In other words, you don't need to worry about classifying your nodes with Apache and Apache PHP.
- The `priority` attribute of `10` ensures that your app has a higher priority on port 80 than the default Apache vhost app.
- The file `/var/puppet_quickstart_app/index.php` contains whatever is specified by the `content` attribute. This is the content you will see when you launch your app. Puppet uses the `ensure` attribute to create that file the first time the class is applied.

Using your custom module in the main manifest

1. From the command line on the Puppet master, navigate to the main manifest (`cd /etc/puppetlabs/code/environments/production/manifests`).
2. With your text editor, open `site.pp` and add the following Puppet code to your default node. **Remove the apache class you added previously.** Your site.pp file should look like this after you make your changes (although you may have portions from earlier in the Quick Start Guide):

   ```
   node default {
     class { 'puppet_quickstart_app': }
   }
   ```

   Note: Since the `puppet_quickstart_app` includes the `apache` class, you need to remove the first `apache` class you added the master node, as Puppet will only allow you to declare a class once.

3. From the command line on your agent, run `puppet agent -t` to trigger a Puppet run.

   When the Puppet run is complete, you will see in the agent's log that a vhost for the app has been created and the Apache service (httpd) has been started.

4. Use a browser to navigate to port 80 of the IP address for your node, as in `http://<yournodeip>:80`.

   Tip: Be sure to use `http` instead of `https`.

Congratulations! You have created a new class from scratch and used it to launch a Apache PHP-based web app. Needless to say, in the real world, your apps will do a lot more than display PHP info pages. But for the purposes of this exercise, let's take a closer look at how Puppet is managing your app.

Using Puppet to manage your app

1. **On the Puppet agent**, open `/var/www/puppet_quickstart_app/index.php` , and change the content to something like, "THIS APP IS MANAGED BY PUPPET!"
2. Refresh your browser, and notice that the PHP info page has been replaced with your new message.
3. Run `puppet agent -t --onetime` on your Puppet agent.
4. Refresh your browser, and notice that Puppet has reset your web app to display the PHP info page. (You can also see that the contents of `/var/www/puppet_quickstart_app/index.php` has been reset to what was specified in your manifest.)

Using a site module

Many users create a "site" module. Instead of describing smaller units of a configuration, the classes in a site module describe a complete configuration for a given *type* of machine. For example, a site module might contain:

- A `site::basic` class, for nodes that require security management but haven't been given a specialized role yet.
- A `site::webserver` class for nodes that serve web content.
- A `site::dbserver` class for nodes that provide a database server to other applications.

Site modules hide complexity so you can more easily divide labor at your site. System architects can create the site classes, and junior admins can create new machines.

- **On the Puppet master**, create `/etc/puppetlabs/code/environments/production/modules/site/manifests/basic.pp` , and edit the file to contain the following:

```
class site::basic {
  if $::kernel == 'Linux' {
    include puppet_quickstart_app
  }
  elsif $::kernel == 'windows' {
    include registry::compliance_example
  }
}
```

This class declares other classes with the `include` function. Note the "if" conditional that sets different classes for different kernels using the `$kernel` fact. In this example, if an agent is a Linux machine, Puppet will apply your `puppet_quickstart_app` class. If it is a Windows machine, Puppet will apply the `registry::compliance_example` class.

1. From the command line on the Puppet master, navigate to the main manifest: `cd /etc/puppetlabs/code/environments/production/manifests` .
2. Add the following Puppet code to the default node in `site.pp` , retaining the classes you have already added:

```
class { 'site::basic': }
```

3. Save and exit, then run `puppet agent -t` from the command line of your Puppet agent.

Summary

You have now performed the core workflows of an intermediate Puppet user. In the course of their normal work, intermediate users:

- Download and modify Forge modules to fit their deployment’s needs.
- Create new modules and write new classes to manage many types of resources, including files, services, and more.
- Build and curate a site module to safely empower junior admins and simplify the decisions involved in deploying new machines.