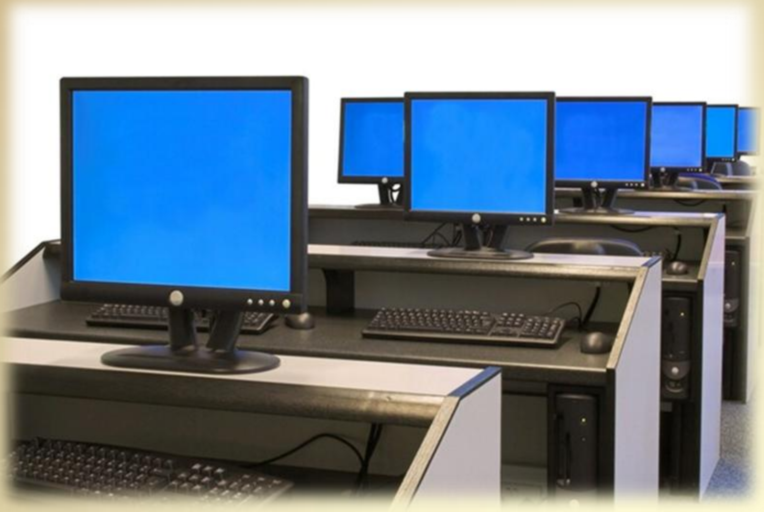


Linux Network Services:

Configuration Management: Puppet

This material is based on work supported by the National Science Foundation under Grant No. 0802551

National Science Foundation
WHERE DISCOVERIES BEGIN

Lesson Overview

System administrators are constantly challenged when managing large enterprise systems using Linux-based operating systems. Administrators need to know a variety of command line differentiations, dependency variations, and support options to support the various computers systems in use. Puppet offers a free, reliable and cross flavor option for remote enterprise computer management.

This lesson will introduce you to the Puppet Administrative tool and provide you with a basic overview on how to use Puppet. Lab activities will provide you with hands-on experience with the Puppet application and assignments and discussion activities will increase your learning on this subject.

Understanding Puppet is important because of its ability to manage enterprise systems. Students hoping to become Linux Administrators must gain mastery of enterprise management tools like Puppet to improve efficiency and productivity.

Objective

You should know what will be expected of you when you complete this lesson. These expectations are presented as objectives. Objectives are short statements of expectations that tell you what you must be able to do, perform, learn, or adjust after reviewing the lesson.

Lesson Objective:

Given five computers that need to be configured, students will assess the shortcomings of scripting that gave rise to configuration management tools and will illustrate the use of one configuration tool for maintenance per industry standards

Lesson Outline

In this lesson, you will explore:

- Introductory Notes
- Puppet Overview
  - What is Puppet?
  - Puppet Terminology
  - Puppet Performance & Future
- Installation & Configuration
  - Ruby Installation
  - Facter
  - Certification
  - Configuration

Notes to Students

The following links are provided for you to review and research additional content on Puppet. You should review these links before starting this lesson to increase your familiarity with available resources on Puppet.

Debian is actually the Mother of Ubuntu! Ubuntu was derived from and modeled after Debian. You may ask, why select Debian instead of Ubuntu for this lesson? The answer is simple. By using Debian, we can quickly install the bare necessities to demonstrate the remote configuration tool called Puppet. Each Debian client installation will be close to 1GB in size, use around 200 MBs of RAM (out of the 400 MBs set aside for it), and use a minimum of CPU time on the host machine. This low Debian footprint will allow us to setup additional Fedora and Debian virtual machines without overwhelming our system resources. Additionally, we will introduce you to a network installation in this lesson as a bonus!

After finishing this lesson, you should be able to setup Puppet, several Debian VMs, and Fedora as well!

Helpful Links

- Debian Linux
- Download Virtual Box
- Virtual Box Manual
- Debian on VirtualBox

Notes to Students (Contd)

Puppet is a fairly new tool and is undergoing constant changes. This lesson has been designed to give a brief introduction to the Puppet application and students need to review all of the listed resources as part of the lesson. Complete books have been written on the Puppet project, and this lesson covers only the basics of the application.

During research and testing for developing this lesson, the author found several real world "scenarios" where he intends to test Puppet as an alternative to commercially available products.

One additional note
During lesson development, the author of this lesson was able to verify that there are immediate openings within the IT Industry for administrators experienced with Puppet. Even the projects web site has information on potential job openings!

Helpful Links
- Debian Linux
- Download Virtual Box
- Virtual Box Manual
- Debian on VirtualBox

Lesson Resources

Before continuing this lesson, review the lab assignments and then download the ISO for a Debian Net Install. The download should take about an hour depending on system throughput.

The image can be found at the link below:

http://cdimage.debian.org/debian-cd/6.0.0/i386/iso-cd/debian-6.0.0-i386-netinst.iso

Once the image has been downloaded, create a backup of the ISO and store it on your computer at a location where it will not be modified. Having a backup ISO image will save you time later if you damage your working ISO image.

Helpful Links

- Puppet documentation
- Puppet FAQ
- About Puppet
- Intro to Puppet
- Use of Puppet
- Puppet Mailing List
- Puppet Recipes
- Puppet Configuration
- Puppet IRC Channel
- Excellent Puppet Videos

Linux Network Admin Tools

System administrators do many tedious tasks that are frequently repeated. Examples of these redundant tasks include:

- Configuring hosts
- Creating users
- Managing applications, daemons, and services

These tasks are often prone to error and waste valuable time due to their inefficiency and potential for human mistakes.

Experienced administrators will automate these tasks using customized scripts and applications. While these meet the objectives of a single user or manager of a small network, transferring these practices to a large enterprise system is not feasible. Additionally, administrator-created scripts are usually designed for one version of Linux and require major tweaks to incorporate them into other Linux flavors.

These edits increase the time and effort required to develop and maintain the very tools you are hoping to use to reduce administrative efforts. Commercial tools are available to help system administrators manage large networks, but they are pricey and lack flexibility, which adds further restrictions.

Required Reading

- Puppet
- Cfengine
- LCFG
- Bcfg2

Network Admin Tools (Contd)

Open Source programmers have solved these challenges by creating several alternative solutions in the typical free natured philosophy enjoyed by Linux users. These tools are not only free, but much more flexible in their use when compared to their commercial "equivalents."

By using open-source products, you and your company can participate and support the Linux philosophy because the source code is at your fingertips, allowing you to develop and enhance/adjust your own code.

Users devoted to Linux maintain a philosophy of supporting their peers and Linux community by actively participating in and testing applications being developed. Puppet is one of those applications.

Required Reading

- Puppet
- Cfengine
- LCFG
- Bcfg2

Introducing Puppet

Puppet is an open source Ruby-based system and configuration management tool that relies on a client-server deployment model. It is licensed using the GPLv2 license and is principally developed by Luke Kanies.

Kanies has been involved in Unix and systems administration since the mid-nineties, and Puppet was developed from his experience. Unsatisfied with existing configuration management tools, Kanies began working in tool development in 2001, and in 2005, he founded Reductive Labs, an open source development house focused on automation tools. Shortly after this, Reductive released their flagship product, Puppet.

Required Reading

- Puppet project
- Puppet Wiki
- Puppet made easy

What Makes Puppet Different?

Many systems and configuration management products such as cfengine, are similar in functionality. So what makes Puppet different? Puppet's defining characteristic is that it speaks the local language of your target hosts. This capability allows Puppet to define systems administration and configuration tasks with generic instructions on the Puppet server. These instructions are often called **recipes**.

Puppet's recipe syntax allows you to create a single script that allows you to create a user on all your target hosts. In turn, this recipe is interpreted and executed on each target host using the correct local syntax for that host. For instance, if the recipe is executed on a Red Hat Linux server, the user would be created with the **useradd** command. If the same recipe is executed on a FreeBSD target, the **adduser** command would be executed. Because Puppet recipes are so portable, community members and contributors share recipes for a variety of activities on the Puppet website, mailing list, and IRC channel!

The next area Puppet excels in is flexibility. As a result of its open source nature, you always have free access to Puppet's source code, meaning if you have a problem and you have the skills to do so, you can alter or enhance Puppet's code to suit your environment.

Additionally, community developers and contributors regularly enhance and add to the functionality of Puppet. A large community of developers and users also contribute to providing documentation and support for Puppet.

What Makes Puppet Different? (Contd)

Puppet is also readily extensible. Functions to support custom packages and configurations unique to your environment can be quickly and easily added to your Puppet installation. In addition, the Puppet community regularly adds code and packages you can modify or incorporate into your environment.

Finally, Puppet makes use of another Ruby-based tool, Facter.

Facter is a system analysis tool that allows you to query and return information about hosts that you can use in your Puppet configuration as variables.

The ability to use variables means you can write generic configuration instructions and use Facter-returned variables to ensure the right values are configured on the right host. This precludes the need for external databases, configuration files, or directories.

Required Reading
- Facter

“Facter is a lightweight program that gathers basic node information about the hardware and operating system.

Facter is especially useful for retrieving things like operating system names, hardware characteristics, IP addresses, MAC addresses, and SSH keys.”

From puppetlabs.com

How Does Puppet Work?

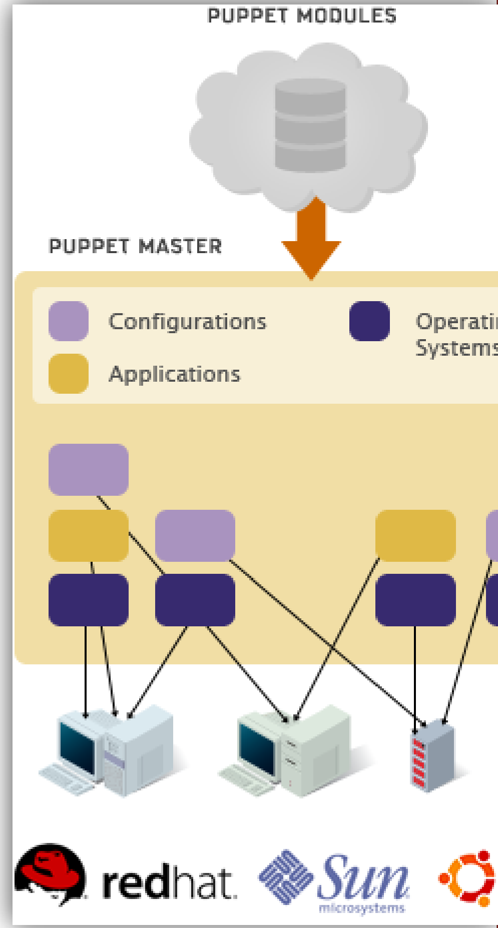
With Puppet, central servers, called *Puppet masters*, are installed and configured. Client software is then installed on the target hosts, called *Puppets* or *nodes*, that you wish to manage. Configuration is defined on the Puppet master, compiled, and then pushed out to the Puppet clients when they connect.

To provide the client-server connectivity, Puppet uses XML-RPC web services running over HTTPS on TCP port 8140. To provide security, the sessions are encrypted and authenticated with internally generated self-signed certificates.

Each Puppet client generates a self-signed certificate that is then validated and authorized on the Puppet master.

Thereafter, each client contacts the server, **by default every half hour**, to confirm that its configuration is up to date. If a new configuration is available or the configuration has changed, it is recompiled and then applied to the client. If required, a configuration update can also be triggered from the server, forcing configuration down to the client. If any existing configuration has varied on the client, it is corrected with the original configuration from the server.

Puppet, however, is more than just a client-server configuration management tool—it is a three-tier architecture that combines a declarative language, transactional layer, and resource abstraction layer.



View complete image

Puppet Resource Types

Puppet is preloaded with a number of resource types by default including types to manage files, services, packages, cron jobs, and file systems. Within each type, you can specify the file that must be managed. For instance you can specify a type for the /etc/passwd file and then set the filename as the title of your resource.

When referring to the resource in other parts of your configuration file, reference the title. Finally, you can set the owner attribute which tells Puppet to set the ownership of the file to the root user.

Resource Type	Description
cron	Manages cron jobs
exec	Executes external scripts
file	Manages files
filebucket	Respository for backing up files
group	Manages groups
host	Manages host entries
interface	Configures interfaces (Redhat & Solaris only)

Puppet Resource Types (Contd)

Table continued from previous . . .

Resource Type	Description
Mailalias	Manages mail aliases
Maillist	Manages mailing lists
Mount	Manages mount entries
Notify	Sends a message to the Puppet log file
Package	Manages packages
Schedule	Defines Puppet scheduling
Service	Manages services
Sshkey	Manages SSH host keys
Tidy	Removes unwanted files
User	Manages users
Yumrepo	Manages YUM repositories
zones	Manages Solaris zones

Required Reading

- Puppet Types

Puppet Collections

Puppet users can move beyond single resources types by using collections. Puppet collections allow you to group together many resources. For example, an application such as Apache is made up of a package, a service, and a number of configuration files. Puppet calls these collections "classes." Each component of an application is represented as a resource (or resources) and then collected together in a class and applied to a node.

The Puppet language also defines the nodes you wish to configure. After a client is connected to Puppet, a node definition can be created that defines what resources and collections of resources are applied to each node.

Node definition allows you to apply appropriate configuration to all nodes running a particular platform or a particular service. For example, you can specify all resources required for Red Hat Enterprise Linux nodes or all configuration required for a database or web server.

The Puppet language allows features usually found in programming languages including:

Required Reading

- Variables
- Arrays
- Conditions

Note: Puppet is rapidly being developed and should be considered a dynamic application.

Puppet Performance

There are two keys to performance management—the number of nodes connected and the amount of configuration defined on each node. As of this writing, there are no known limitations or defined end-points for the number of clients and servers that can be managed with Puppet.

Testing suggests that fifty to one hundred nodes with a moderate amount of configuration can be managed on a single CPU Puppetmaster with 2GBs of RAM. It can be assumed that more nodes will require additional CPU and memory support.

Internally, Puppet uses the WEBbrick web server to interface with clients. The WEBbrick server has some performance limitations and limits administrators to smaller configurations. For better performance, administrators may want to use the Mongrel web server since Puppet has been tested with it.

Generally, the WEBbrick web server will start to experience difficulties in performance at about 50 nodes. This is probably a good breaking point to switch to Mongrel.

Recent reports indicate that when running Puppet with load balancing and Mongrel, node volumes of 5000 or more are feasible with extended memory and resource additions.

Future of Puppet

Lastly, it is very important to remember that Puppet is still in the midst of development and change. The Puppet community is growing quickly, and many new ideas, developments, patches, and recipes appear every day.

But this rapidly changing environment does make it important to keep an eye on the Puppet mailing lists, IRC channel, and #Puppet on Freenode as new enhancements that could help you better manage your configurations appear frequently.

Notes:

IRC information
IRC stands for Internet Relay Chat and is not a secure method of communication. Be careful using scripts and running executables without understanding their origins or intent.

Distinction between client and node

- Client refers to the Puppet client daemon that connects to the Puppet master and retrieves the configuration.
- Node refers to the underlying host to which configuration is applied.

Puppet Installation

Configuration and Testing

Installation Prerequisites (Dependencies)

The process of installing Puppet's master and client components is quick and easy, but you will need to install some prerequisites first. The prerequisites are required for both hosts that run the Puppet master or client. (Remember, your Puppet master can also be a Puppet node). These prerequisites include the Ruby interpreter, select Ruby libraries, and Facter.

Since Puppet is a Ruby-based application, you must first ensure Ruby is installed, and if not, install Ruby and key Ruby libraries. Many Linux and other Unix-like platforms come with a Ruby package, and you can install this package and any required library packages. If your distribution does not have a package, you can install Ruby from source.

Please refer to Documentation and Installation Instructions for your specific version of Linux at the Puppet Project website.

Required Reading

- Puppet Management

View Video
VideoLesson8PuppetInstall
Master(C4L8A2).mp4

Installation (Ruby)

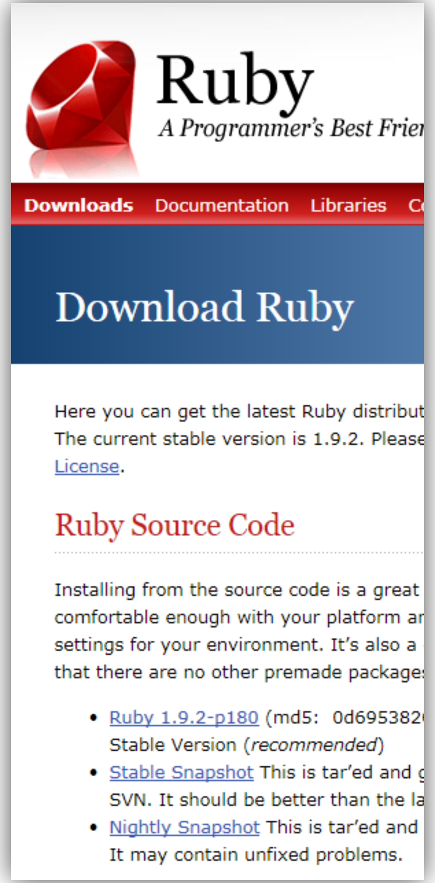
Many Linux distributions and Unix operating systems have Ruby packages available, including Red Hat Enterprise Linux, Fedora, Debian, Ubuntu, SuSE, and Mandriva.

Some distributions bundle all the required Ruby binaries and libraries in a single package. Other distributions separate the core development environment and the libraries into individual packages.

So, if you are installing Ruby and its libraries on a Red Hat Fedora host, you need to use its package management system to install Ruby and Ruby-libs packages. See command line example below for installation of Ruby and Ruby-libs using Yum.

```
# yum install ruby ruby-libs
```

Fortunately, Debian auto-downloads and configures Ruby!



Download Ruby

Here you can get the latest Ruby distribut
The current stable version is 1.9.2. Please
License.

Ruby Source Code

Installing from the source code is a great
comfortable enough with your platform ar
settings for your environment. It's also a
that there are no other premade package

- Ruby 1.9.2-p180 (md5: 0d695382
  Stable Version (*recommended*)
- Stable Snapshot This is tar'ed and g
  SVN. It should be better than the la
- Nightly Snapshot This is tar'ed and
  It may contain unfixed problems.

Ruby Packages on Linux Distros

OS	Ruby	Library	Additional Pkg
Debian	Ruby	libruby	libopenssl-ruby , libxmlrpc-ruby
FreeBSD	Ruby		
Gentoo	Ruby		
Madriva	Ruby		
NetBSD	Ruby		
OpenBSD	Ruby		
Redhat	Ruby	ruby-libs	
Suse Ruby	Ruby		
Ubuntu	Ruby	libruby	libopenssl-ruby , libxmlrpc-ruby

Installation: Missing Files

Installing the Ruby package and libraries may not always install all required libraries.

If the following base libraries are not installed as part of your base Ruby installation, you may need to selectively install the missing libraries. These may include the following files depending your distro and version.

- • base64
- • cgi
- • digest/md5
- • etc
- • fileutils
- • ipaddr
- • openssl
- • strscan
- • syslog
- • uri
- • WEBbrick
- • WEBbrick/https
- • xmlrpc

The good news is that most developers of common distros currently include full Puppet support and have added the packages and their dependencies to the package managers!

Verifying Facter Install

Puppet relies on the Facter tool to provide information about hosts. Facter is also developed by Puppet's developers, Luke Kanies and Reductive Labs, and is written in Ruby.

Facter is a cross-platform Ruby library for returning "facts" about the operating system of a host including IP addresses, and operating system versions. Facter can be installed from source. Some platforms have a Facter package available.

Facter is available as a source package from Reductive Labs. To test if Facter is installed, enter the following at a command line: **# facter --version**

Facter names may differ across distros. For instance:

Platform	Facter Package Name
Debian	facter
Fedora	facter
FreeBSD	facter
Gentoo	facter
OpenBSD	ruby-facter
Ubuntu	facter

Required Reading
- Facter for Puppet

Facter for OS X

There is also a Facter package available for OS X. The package combines both Facter and Puppet and can be downloaded from:

http://reductivelabs.com/downloads/packages/OSX/

Reductive reports that the build appears to be stable, but the OS X version has not been extensively tested.

The Reductive Labs website also reports a working Windows version available through the use of an API call (REST API).

The Windows version of Facter was not tested by the author of this lesson but students wishing to test the API version can find more information at the facter site.

Facter
- Windows Support

Fact Descriptions for Facter

Instructions	Commands
architecture	The architecture of the node, x86_64, for example
domain	The domain name of the node
facterversion	The version of Facter running on the node
fqdn	The fully qualified domain name of the node
hardwaremodel	The model of the hardware, for example, x86_64
hostname	The hostname of the node
id	The user running Facter
ipaddress	The IP address
kernel	The kernel type on the node
kernelrelease	The kernel release of the kernel running on node
lsbdistcodename	The LSB codename of the distribution running on the node
lsbdistdescription	The LSB description of the distribution running on the node

Continued on next . . .

Fact Descriptions (Contd)

Fact	Description
Lsbdistid	The LSB release ID of the distribution running on the node
Lsbdistrelease	The LSB release number of the distribution running on node
Macaddress	The MAC address of the node
Memoryfree	The available memory
Memorysize	The total memory size
Operatingsystem	The node's operating system, for example, Fedora
Operatingsystemrelease	The release of the node's operating system

Continued on next . . .

Fact Descriptions (Contd)

Continued from previous . . .

Instructions	Commands
Processor	The make of each processor, includes an entry for each processor, incremented from 0
Processorcount	The total processor count
Puppetversion	The version of Puppet on the node
Rubyversion	The version of Ruby on the node
sshdsakey	The node's public DSA key
Sshrsakey	The node's public RSA key
swapfree	The available swap space
swapsize	The total swap size

Installing Ruby Documentation

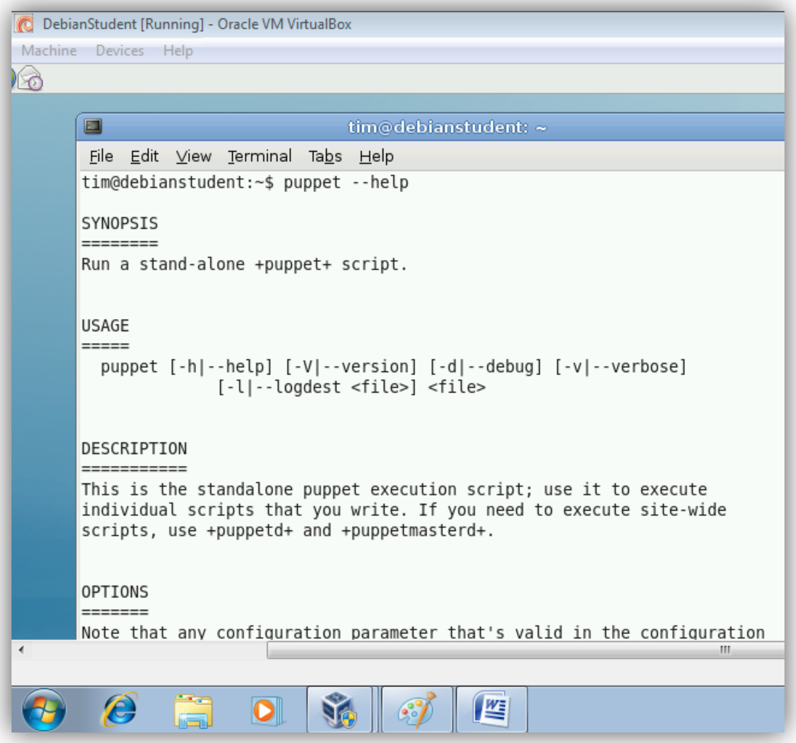
You may want to optionally install the RDoc package. RDoc is the Ruby Standard Documentation System that allows Ruby apps to return help text and documentation when prompted.

RDoc may come with Ruby versions later than 1.8.2, but several platforms still have a separate package for RDoc. If you do not have RDoc installed, your Facter or Puppet binaries will not return help text when prompted, for example, if you enter:

# facter --help

You will receive the response:

No help available unless you have RDoc::usage installed.


DebianStudent [Running] - Oracle VM VirtualBox
Machine Devices Help
tim@debianstudent: ~
File Edit View Terminal Tabs Help

```
tim@debianstudent:~$ puppet --help

SYNOPSIS
========
Run a stand-alone +puppet+ script.


USAGE
=====
  puppet [-h|--help] [-V|--version] [-d|--debug] [-v|--verbose]
              [-l|--logdest <file>] <file>


DESCRIPTION
===========
This is the standalone puppet execution script; use it to execute
individual scripts that you write. If you need to execute site-wide
scripts, use +puppetd+ and +puppetmasterd+.


OPTIONS
=======
Note that any configuration parameter that's valid in the configuration
```

Installing Puppet: OS Support

Puppet currently recognizes and supports the following versions of Linux:

Linux
CentOS
Debian 3.1 and later
Fedora Core 2-6
Fedora 7 and later
Gentoo Linux
Mandriva Corporate Server 4
RHEL 3 and later
Oracle Linux
SuSE Linux 8 and later
Ubuntu 7.04 and later
ArchLinux

BSD
FreeBSD 4.7 and later
OpenBSD 4.1 and later

Other Unix
Macintosh OS X
Sun Solaris 2.6
Sun Solaris 7 and later
AIX
HP-UX

Windows
Windows (Supported in Puppet 2.6 & later)

Required Reading
- Puppet Platform guide

Installing Puppet: Packages

Puppet is available as a package for a number of platforms, but not all of them yet. During the development of this lesson, packages were available for Debian, FreeBSD and OpenBSD, Gentoo, Red Hat Fedora, and Ubuntu as part of their package management systems.

Some packages contain both the server and the client, while others have separate packages for each.

Platform	Puppet Server	Puppet Client
Debian	Puppetmaster	Puppet
Fedora	Puppet-server	Puppet
FreeBSD		Puppet
Gentoo		Puppet
OpenBSD		Ruby-Puppet
Ubuntu		Puppet

Required Reading

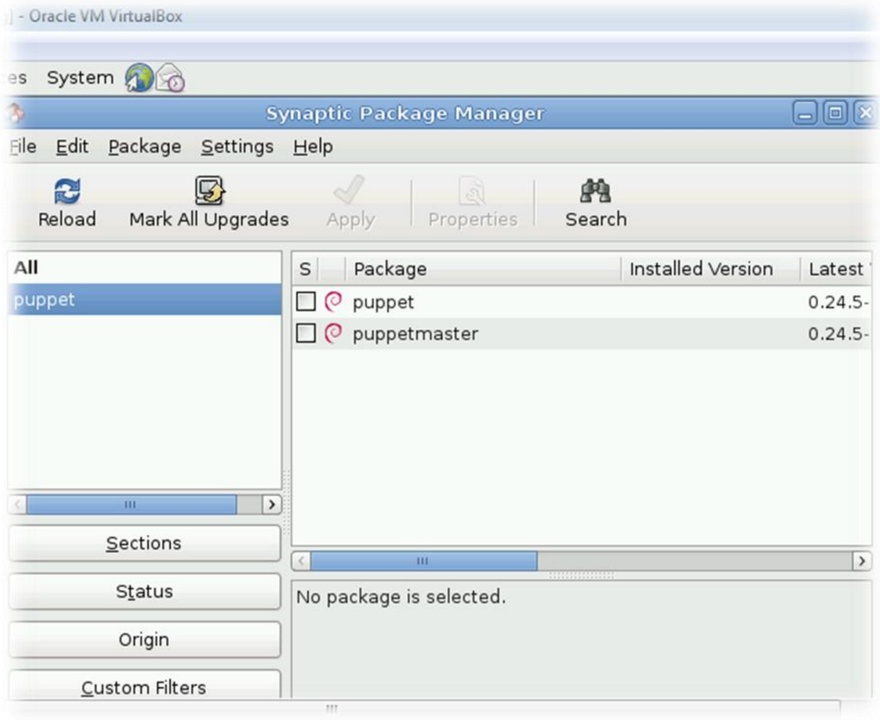
- Puppet Platform guide

Installing Puppet: Package Mgmt

Puppet is now offered as a package utility from several Synaptic Package Managers. Simply access the Synaptic Package Management GUI and search for “Puppet.”

Some flavors will offer at least two selections. The first will be the Puppet Client and the second will be Puppet Master.

Make sure you select the right option (normally client). Choose your desired package and select **Apply**.



S	Package	Installed Version	Latest
☐	puppet		0.24.5-
☐	puppetmaster		0.24.5-

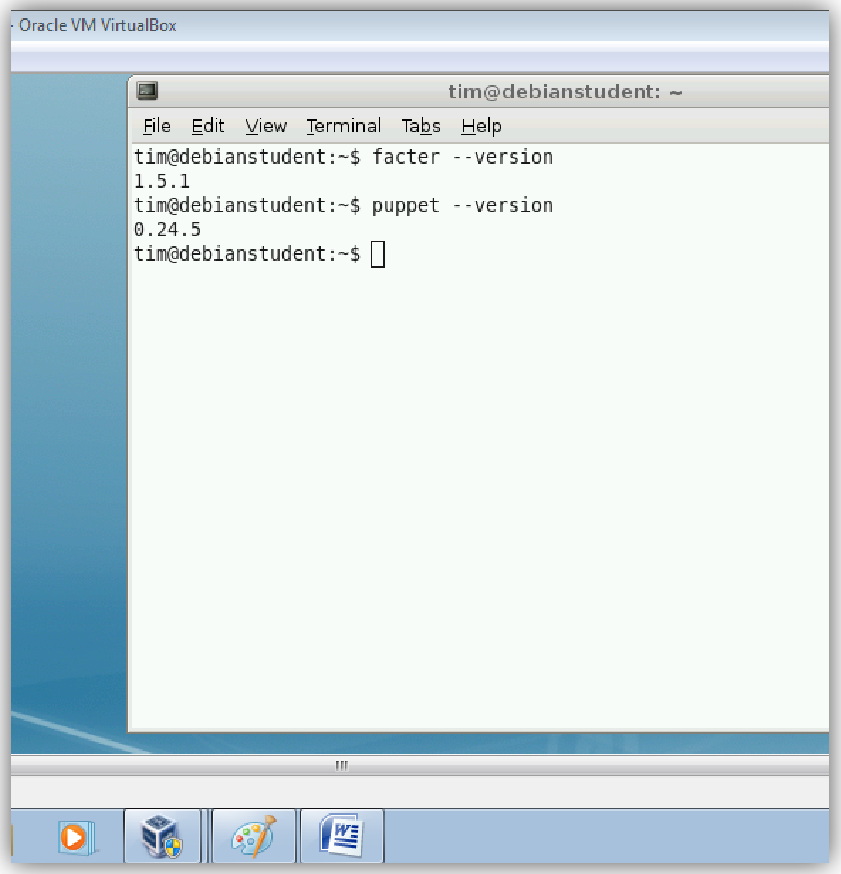
No package is selected.

Installing Puppet: Confirmation

To confirm Puppet is installed, enter the following command:

# Puppet –version

You will get a response similar to:
0.24.5



```
tim@debianstudent:~$ facter --version
1.5.1
tim@debianstudent:~$ puppet --version
0.24.5
tim@debianstudent:~$
```

Location of Puppet Files

The default folder setup for Puppet is as follows:

Files	Description
/etc/Puppet -	Basic Puppet configuration information
/etc/Puppet/manifests	Node config mappings
/etc/Puppet/manifests/filetypes/*	Various filetype definitions
/etc/Puppet/manifests/nodes/*	Server lists and the classes they use
/etc/Puppet/manifests/server-groups	Maps services with a server type
/etc/Puppet/manifests/service-types	Contains each service and requirements for that service
/etc/Puppet/manifests/site.pp	Contains the 'root' config file which includes other config files
/var/lib/Puppet/	Puppet files
/var/lib/Puppet/config	Config files for the actual nodes (e.g. httpd.conf)
/etc/lib/Puppet/bucket	Backup of overwritten config files

Note: *Some of these folders will be created as part of your configuration and setup and may not be available initially.*

Configuring Puppet Daemon

Now let's look at the Puppet master daemon by running it and adding our first node. One of the strengths of the Puppet infrastructure is that most of the functionality will run with a default configuration without any changes. The only two resources you need to run Puppet are a user and group and a basic configuration to apply to your first node.

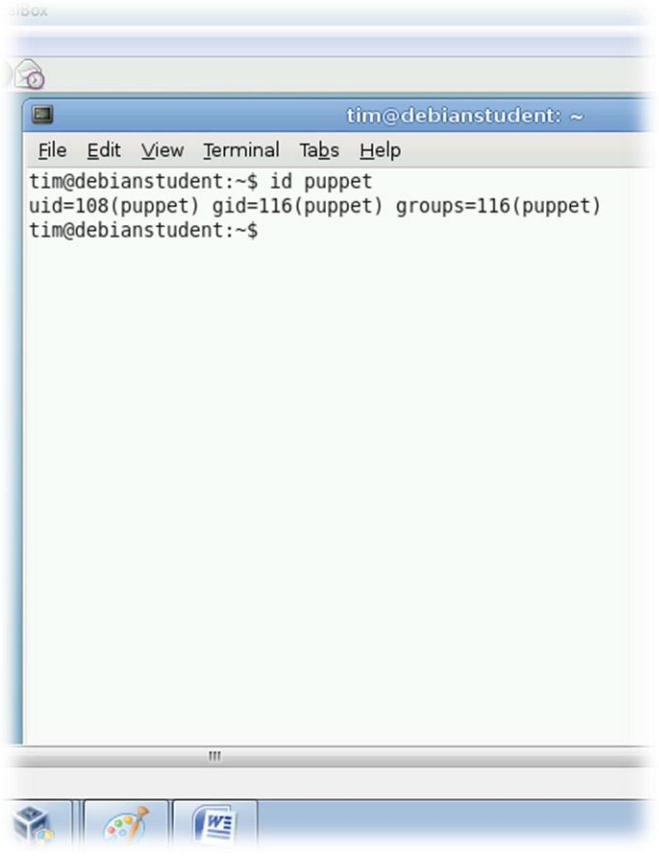
We will create a user and group, and then start the Puppet master daemon for the first time using the basic configuration.

If you installed Puppet from a package, a user and a group (usually called Puppet), was already created for you. You can check for this user by using the "id" command as shown:

# **id Puppet**

The response below indicates that a user and a group exist:

uid=108(Puppet) gid=116(Puppet) groups=116(Puppet)



```
tim@debianstudent:~$ id puppet
uid=108(puppet) gid=116(puppet) groups=116(puppet)
tim@debianstudent:~$
```

Puppet Daemon (Create User)

If you do not see a user and a group, you will need to create them. You should name the user and group with the default name **Puppet**. On a Red Hat host, use the following commands:

```
# groupadd Puppet
# useradd –M –g Puppet Puppet
```

In Debian/Ubuntu:

```
Sudo addgroup accounts
Sudo adduser Puppet
```

View Video
VideoLesson8PuppetConfig
(C4L8A3).mp4

Starting Puppet Master

Now that you have a user and group to run the Puppet master server, you can start it using the Puppetmasterd binary as shown below.

```
# Puppetmasterd
```

Note: The Manifest /etc/Puppet/manifests/site.pp must exist or you will get an error message.

You may see that trying to start Puppetmasterd has resulted in an error message stating that the manifest (/etc/Puppet/manifests/site.pp) must exist. A *manifest* is Puppet's term for a text document that defines a particular configuration or configurations. These manifests are then compiled and applied to a Puppet node to set the desired configuration on the node.

Puppet requires a central manifest file, called the *site manifest*, before the master daemon can be started. By default, the site manifest file is called site.pp and is located in the /etc/Puppet/manifests directory. You will learn how to reconfigure this location later in this lesson. This central manifest will ultimately contain all the configuration information required to configure all your nodes, either directly in the file or by including and importing other files.

Note: If you added the puppetmaster from a Package Manager, puppetmaster was probably already started as part of the installation process.

Creating a Site Manifest

Puppet will not start without a central manifest. Consequently, you just want to create a simple site.pp file to get Puppet started.

First, create the directory:

```
# mkdir –p /etc/Puppet/manifests
```

Then create the file:

```
file { "/etc/passwd":
owner => "root",
group => "bin",
mode => 644,
}
```

This site.pp file is very simple: it sets the user and group ownership of the /etc/passwd file as well as its permissions. Indeed, your first site.pp file could do anything; you just need a file with correct syntax to start the Puppet daemon.

Starting the Master Daemon

To start the master daemon type:

# Puppetmasterd --verbose --no-daemonize

We started Puppetmasterd with the --verbose and --no-daemonize options.

The **--verbose** option turns on verbose logging, and the **--no-daemonize** option forces the master daemon to run in the foreground. This mode is ideal for troubleshooting your master daemon.

Note: The --no-daemonize option was introduced in version 0.24 of Puppet. Prior to this, the –verbose option by it caused the Puppetmasterd not to daemonize.

In most cases, puppetmaster daemon starts as part of the installation process.

```
info: Starting server for Puppet version 0.23.0

info: Parsed manifest in 0.01 seconds

info: Listening on port 8140

notice: Starting Puppet server version 0.23.0
```

Display text as seen when Puppetmaster Daemon starts

Starting Master Daemon (Contd)

Once started, Puppet expects to find each node defined in a manifest, either directly in the site.pp file or in another file imported into the site manifest. The node definitions tell Puppet about each host to be configured and exactly what configuration applies to each. For example, you might have configuration specific to Debian hosts, or to web servers, or hosts in a specific location. When you are using node definitions, only the configuration defined to a particular node will be applied to that node.

Puppet detects if you have nodes defined or not. If you do not have defined nodes, Puppet turns off node designation. With node designation turned off, all configuration resources defined will be applied to all nodes that connect to the master. Since we do not have nodes nor any substantive configuration in our example, it is best to turn off nodes until we are ready to define our first node.

In our current example, the master daemon has started and is listening on TCP port 8140. You will need to open this port in any firewall you are running on the local host. If the port is open and the master daemon has started without error messages, you are now ready to connect your first node.

Starting the Puppet Client

Unlike the Puppet master daemon, the Puppet client daemon runs as the root user, allowing it to perform the required configuration actions on your Puppet node. The first time you start a node, it will generate a local self-signed certificate, connect to a master server (which, in addition to distributing configuration to nodes, also acts as a Certificate Authority) you specify, and request that the certificate be signed.

Note: *Puppet relies on SSL to communicate between client and server. You need to ensure that the date/time on your server and client is correct and appropriately synchronized to ensure SSL functions correctly.*

Once the certificate is signed, the node will request whatever configuration is specified for that node. The master server will then compile and deliver the configuration. The configuration is then implemented on the node. The Puppet client will then periodically check the master to see whether the configuration defined there is unchanged. By default, this periodic check occurs every 30 minutes. If it has changed, the client will request a recompilation of the configuration, and the new configuration will be implemented on the node.

Note: *If you are running the Puppet client on the same host as the server, your certificate will be automatically signed.*

Starting the Puppet Client (Contd)

To start the Puppet client:

# **Puppetd --server Puppetmaster.localhost.net --verbose --waitforcert 60**

notice: Did not receive certificate

We started the Puppet client daemon with three options: **--server**, **--verbose**, and **--waitforcert**.

The --server option tells the client the name of the server to which it must connect. You should specify the server in the form of a fully qualified domain name. The –verbose option enables verbose output for the client and stops it going into the background and daemonizing.

The last option, --waitforcert, tells the client to check every 60 seconds to see whether a signed certificate is returned from the server. This option is generally only used when you are connecting a new node and tells the client daemon to keep checking the server for a signed certificate.

Important:

If you should happen to see the following readout:

notice: Did not receive certificate

Check your master daemon and you should see a corresponding log message:

notice: Host node1.localnet.net has a waiting certificate request

This message indicates that the client's request to have a certificate signed has been received, and now you need to act on it.

Signing Your Client Certificate

So how does the node acquire a signed certificate, receive authentication, and deliver its configuration?

Certificate signing is done on the master server by the Puppetca tool. The Puppetca tool controls the Puppet Certificate Authority and allows certificate requests to be signed or revoked.

Note: You can configure Puppet to automatically sign all incoming certificate requests (known as *autosign*), either from every node or using coarse-grained authentication to selectively sign node requests based on hostname or domain. Using both forms of autosign poses a serious security risk as they bypass Puppet's security controls.

Using autosign is not recommended. But if you do, you can see more details about autosign and Puppet’s certificate management at:

http://www.reductivelabs.com/trac/Puppet/wiki/CertificatesAndSecurity.

Signing Your Client Certificate (Contd)

Use the command below to list all the waiting certificate signing requests:

```
# Puppetca --list
```

If you followed this example precisely, you will receive this response: *node1.localnet.net*
The --list option listed our node's signing request. Now, if you want to sign it, you can use the Puppetca command again as shown below:

```
# Puppetca --sign node1.localnet.net
```

Similarly, the system will respond with: *Signed node1.localnet.net*
We specified the option **--sign** together with the hostname of the node whose certificate we wish to sign, in this case, node1.localnet.net. You should now see that the command has returned a message indicating the certificate is now signed. The node is now authenticated to the server.

If we go back to the client daemon, we will see logging messages indicating the certificate has been returned, and the client has been started as shown below:

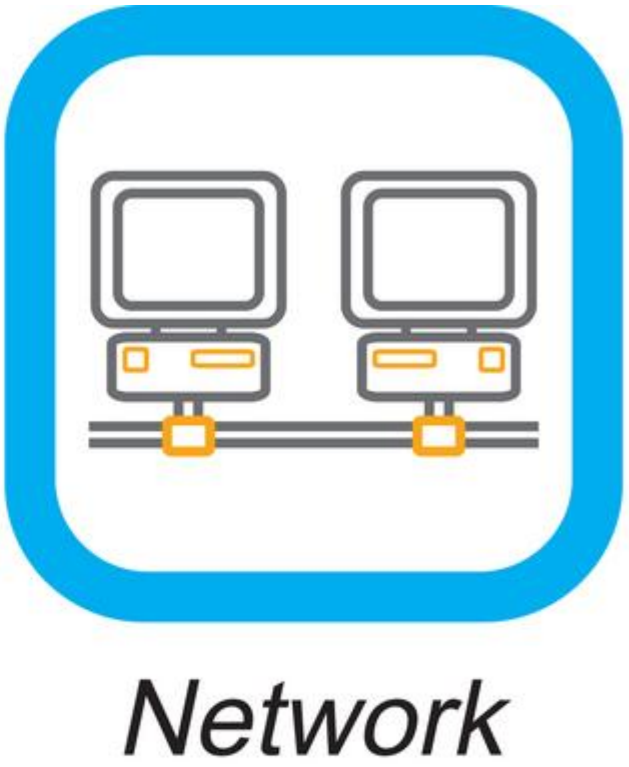
```
notice: Got signed certificate
notice: Starting Puppet client version 0.24.0
```

Then server will now compile and deliver any configuration for that node to the client daemon to be applied.

Running Puppet Daemons

Like most Unix and Linux applications, the Puppet daemons, Puppetmasterd and Puppetd, can be started and stopped using your platform's standard spawn process.

If you've installed Puppet from a package, you'll usually find that the package installation process has added the appropriate links and scripts to start the daemons when your host boots.



Configuring Puppet

Your Puppet installation comes with a number of binaries that run the various Puppet functions and daemons. You have already reviewed Puppetd, Puppetmasterd, and Puppetca binaries, but you will explore them in more detail in the next few screens. we'll go into more detail on them in the sections that follow. You will not learn every configuration option, just the main ones. For a full reference to every command-line and configuration file option, you can find a guide at:

http://www.reductivelabs.com/trac/Puppet/wiki/ConfigurationReference.

Each Puppet binary can be configured via the command line or via a configuration file or files. A list of some of the key binaries are listed on the next screen.

Each binary has a different set of command-line options you can use to run and configure it. The easiest way to see the configuration options used for each binary is by executing the binary with the --help option as shown below:

```
# Puppet –help
```

Note: To get the --help text, you need to have the RDoc library installed as discussed earlier in this lesson.

Configuring Puppet (Contd)

Puppet binaries and description . . .

Binary	Description
Puppet	A local configuration script interpreter and executor
Puppetd	The Puppet client daemon that runs on the managed host
Puppetmasterd	The Puppet master daemon that manages the nodes
Puppetca	The Puppet Certificate Authority server used to authenticate nodes to the master server
Puppetrun	A tool that can connect to clients and force them to run their configurations
Filebucket	A client to send files to a Puppet file bucket
Ralsh	An interactive Puppet shell for converting current state into Puppet configuration code
Pi	Tool to output documentation about Puppet types and providers
Puppetdoc	Tool that prints Puppet reference documentation (generally only used within other Puppet tools)

Puppet Configuration Files

Puppet configuration can also be managed via the configuration file. Puppet's configuration file model is similarly to INI files. Each file is divided into namespace sections, and each section name is enclosed in parentheses and named for the Puppet function it configures. For example, the namespace used to configure the Puppet client daemon is called **[Puppetd]**.

The use of namespaces means options can be used in multiple namespaces if the option is relevant to the binary being configured. For example, you can specify the same option twice, with different values, in the [Puppetd] and [Puppetmasterd] namespaces, and each binary will use only the configuration option contained in its own namespace.

Puppet also has support for multiple environments including production, testing, and development. There is some documentation available describing multiple environments at http://reductivelabs.com/trac/Puppet/wiki/UsingMultipleEnvironments.

You should view the configuration information located on the Puppet website for more detailed configuration instructions.

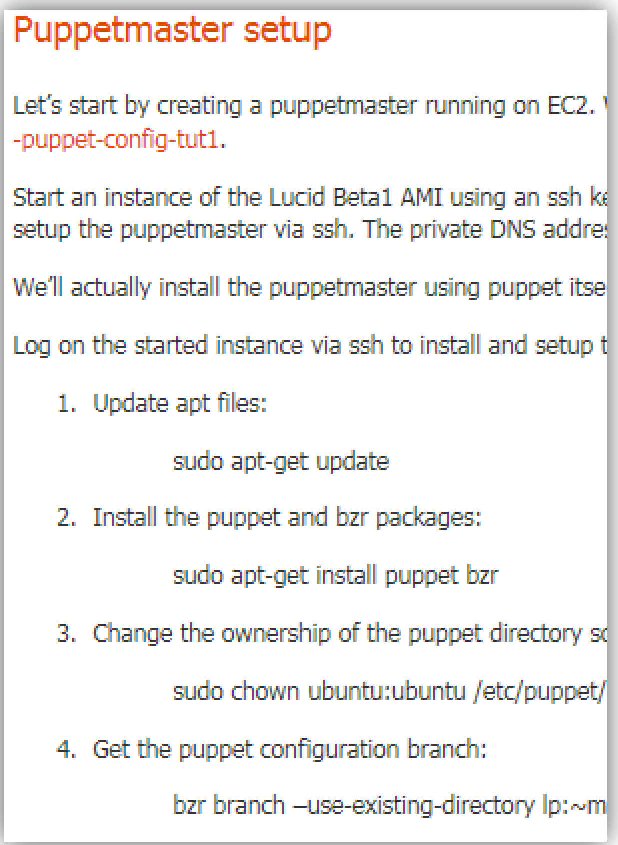
http://docs.Puppetlabs.com/guides/configuring.html

Configuring Puppetmasterd

The Puppet master daemon is initiated by the puppermasterd binary. This is the core of the Puppet client-server model; the server compiles and provides the compiled configuration to the nodes.

In this section, we'll look at some of the command-line flags and configuration file options that can be used to configure the Puppet master daemon.

Notes: Puppetmaster is an ongoing application in progress. Please refer to the Project website for the most current information on your version.



Puppetmaster setup

Let's start by creating a puppetmaster running on EC2. \
-puppet-config-tut1.

Start an instance of the Lucid Beta1 AMI using an ssh ke
setup the puppetmaster via ssh. The private DNS addre

We'll actually install the puppetmaster using puppet itse

Log on the started instance via ssh to install and setup t

1. Update apt files:

```
sudo apt-get update
```

2. Install the puppet and bzr packages:

```
sudo apt-get install puppet bzr
```

3. Change the ownership of the puppet directory so

```
sudo chown ubuntu:ubuntu /etc/puppet/
```

4. Get the puppet configuration branch:

```
bzr branch –use-existing-directory lp:~m
```

From ubuntu.com

Puppetmasterd Flags

Flag	Description
--daemonize | -D	Daemonize the process (default)
--no-daemonize	Do not daemonize the process
--debug | -d	Enable debugging (leaves process in the foreground)
--logdest | -1 *file* | console | syslog	Specify logging destination (defaults to syslog)
--mkusers	Creates the initial set of users and directories
--verbose | -v	Enable verbose output (leaves process in the foreground)
--help | -h	Print help text
--version | -v	Print the version

Puppetmasterd Flags Explanation

--daemonize tells the Puppet master daemon to daemonize the process and is the default behavior of the Puppetmasterd binary when executed

--no-daemonize option flag prevents the process being daemonized and leaves it running in the foreground

--debug option causes the process to output debugging data. This is useful for troubleshooting.

--logdest flag lets you tell the master daemon where to output logging data. You have the choice of specifying a file name, syslog output, or the console. It defaults to syslog output.

--mkusers flag only needs to be run once when you first install Puppet. It creates the required Puppet user and group for Puppet to run as (if they were not yet created).

--verbose option outputs all logging messages to the command line.

--help and **--version** options print the help text and version, respectively.

Namespace Options

In the Puppet configuration file, there are some useful options for the [Puppetmasterd] namespace that you can use to configure the Puppet master daemon. For example:

Option	Description
user	The user who should run the Puppet master daemon
group	The group who should run the Puppet master daemon
manifestdir	The directory to store configuration manifests, defaults to $confdir/manifests
manifest	The name of the site manifest file, defaults to $manifestdir/site.pp
bindaddress	The interface to which to bind the daemon
masterport	The port to run the Puppet master daemon on

The user and group options tell Puppetmasterd what user and group to run as; this defaults to Puppet in both cases.

The manifestdir and manifest options specify the directory for storing manifests and the name of the site manifest file, which defaults to **/etc/Puppet/manifests** and **/etc/Puppet/manifests/site.pp**, respectively.

The bindaddress and masterport options allow you to control what interface and port to which the daemon must bind. These default to binding to all interfaces and to port 8140.

Puppet Client Daemon Options

The command-line operation of the Puppet client daemon is very similar to the operation of the master daemon. It can be configured both from the command line and via a configuration file. In this section, we will look at the options typically specified for the daemon.

Flag	Description
--daemonize | -D	Daemonize the process (default)
--no-daemonize	Do not daemonize the process
--server *name*	Name of the Puppet master server to which it must connect
--waitforcert | -w *seconds*	Time in seconds between certificate signing requests
--onetime | -o	Connect and pull down the configuration once and then exit
--noop	Run in NOOP or dry-run mode
--disable	Temporarily disable the Puppet client
--test | -t	Enable some common testing options
--debug | -d	Enable debugging (leaves process in the foreground)
--verbose | -v	Enable verbose output (leaves process in the foreground)
--logdest | -1 *file* | console | syslog	Specify logging destination (defaults to syslog)
--help | -h	Print help text
--version | -v	Print the version

Daemon Options Explained

--daemonize option is the default action for the Puppetd process. If executed without options, it will run in the background as a daemon

--no-daemonize option flag prevents the process being daemonized and leaving it running in the foreground

--server option is used to specify the name of the Puppet master to which it must connect. It should be specified as a fully qualified domain name

--waitforcert option only applies, as discussed in the "Starting the Puppet Client" section, for Puppet nodes without a certificate. It indicates the time in seconds in between certificate signing requests to a Puppet master. Once the node has a signed certificate, this option does nothing

--onetime option connects the client to the master, requests the node configuration, applies it, and then exits

--noop option allows dry runs of configuration without actually applying the configuration. This allows you to see what the new configuration will do without actually making any changes to the node

--verbose option will output logging messages with the proposed changes that you can verify for accuracy

Daemon Options Explained (Contd)

On the following line, you can see an example of typical noop output:
notice: //File[/etc/group]/mode: is 644, should be 640 (noop)

The notice indicates that the /etc/group file's permissions are 644, but the configuration would change that to 640. The (noop) at the end of the message indicates that no change has been made.

--disable and **--enable** options allow you to turn on and off the Puppet client.

--disable option sets a lock file that prevents the Puppet client from running. The same lock file is set by the Puppet client when running as a daemon to prevent the client from running twice.

--enable option removes the lock file and allows the client to run again on its normal schedule, by default checking half-hourly.

--test option applies a number of common testing options including verbose logging, running in the foreground, and exits after running the configuration once (the --onetime option).

--debug and **--verbose** options enable debug and verbose output from the daemon, and the **--logdest** option allows you to specify where log data will be outputted: console, file, or syslog. The option defaults to syslog output.

--help and **--version** print the help text and the version information.

Puppetd Daemon Configuration Options

There are also some options you can specify in the configuration file to configure the Puppetd daemon:

Option	**Description**
server *Puppet*	The Puppet master server to connect to, defaults to Puppet
runinterval *seconds*	The interval between Puppet applying configuration in seconds, defaults to 1800 seconds, or a half-hour
Puppetdlockfile *file*	The location of the Puppet lock file
Puppetport *port*	The port that the client daemon listens on, defaults to 8139

The server option is the configuration file equivalent of the command-line –server option and allows you to specify the Puppet master server to which you must connect. (It defaults to Puppet.)

The **runinterval** option controls how often configuration is applied to the Puppet node. It is from this option that Puppet gets the default half-hourly application of configuration. The option is in seconds and defaults to 1800 seconds.

The **Puppetdlockfile** option specifies the location of the lock file used by the –disable option to control the running of the Puppet client. The option defaults to $statedir/Puppetdlock. The Puppetdport option controls the port to which the client daemon binds. By default this is 8139.

Configuring Puppetca

The Puppetca binary's primary purpose is to control and interact with the Puppetmasterd's built-in certificate authority. Its principal purpose, if you do not use the automatic signing of certificates (which is turned off by default), is to sign incoming requests from Puppet clients to authenticate new nodes.

Note: *As discussed, autosigning of certificates is dangerous, as anyone can authenticate to your Puppet master. If you want to autosign certificates, use per-host authentication to authenticate only those hosts you identify.*

You have already seen Puppetca's primary function when we connected our first node to Puppet. Puppetca listed and signed certificate requests of new nodes using the --list and –sign options.

```
# Puppetca --sign node1.localnet.net
```

You can specify more than one node on the command line, and you can also sign all outstanding certificate requests by specifying the --all keyword. See example below:

```
# Puppetca --sign –all
```

Puppetca Flags

Flag	**Description**
--revoke | -r *host*	Revoke a node’s certificate
--clean | -c *host*	Remove a node’s certificate from the master
--generate | -g *host*	Generate a client key/certificate pair
--debug | -d	Enable debugging (leaves process in the foreground)
--verbose | -v	Enable verbose output (leaves process in the foreground)
--help | -h	Print help text
--version | -v	Print the version

The **--revoke** option revokes a client’s certificate. You can specify a decimal number, the certificate’s hexadecimal code, or the hostname of the client node. The certificate is added to Puppet’s Certificate Revocation List (CRL). You can specify the CRL using the **cacrl** option in the Puppetmasterd namespace. The master daemon needs to be restarted to update the CRL with the revoked certificate.

The **--clean** option removes all files related to a particular node from the Puppet Certificate Authority. This option is most useful for rebuilding nodes. It removes traces of the old certificate and allows you to submit a new certificate signing request from the client.

The **--generate** option generates a certificate and key pair for the node or nodes specified on the command line.

Puppet Resources

We’ve looked at installing and configuration Puppet in this Lesson, and there are a number of useful resources and documentation online that can also help with this process:

Puppet’s trac site
Bug reports
Puppet support
Installation guide
Configuration reference
Mailing list
Language tutorial
Type references
Function reference
Style Guide
Best practices

Additional Reading

- Puppet modules
- File Serving
- Tips & Tricks
- Best Practices
- Troubleshooting

Lesson Summary

This lesson was created introduce Linux Administrators to a remote system administrative tool named Puppet.

Puppet is a relatively new tool, still in the growing stages, but showing great promise to being a heavy hitter in the Linux enterprise world. Puppet is currently used by several fortune 500 companies for managing and maintaining cross platform Linux environments.

As this is a new application to the Linux community, students familiar with Puppet have a good start in the competitive IT world. With newly created Windows and Apple support, Puppet is predicted to be a top contender for enterprise System management in the future.