Puppet Workshop

Configuration Management Made Easy

Jeroen van Meeuwen, RHCE

Stefan Hartsuiker, RHCE

Puppet Workshop
Configuration Management Made Easy
Edition 1

Author	Jeroen van Meeuwen, RHCE	*j.van.meeuwen@ogd.nl*
Author	Stefan Hartsuiker, RHCE	*s.hartsuiker@ogd.nl*

This book is a configuration management workshop wrapped around puppet, the next-generation configuration management utility that has proven to be simple, straightforward, flexible, stable, fast, extensible and most importantly, truely Free.

Preface

This is the Configuration Management Workshop reader as provided to you by the Operator Groep Delft. This reader is composed form both an introduction to Configuration Management with Puppet as well as a reference for later use.

1. About the Contributors

Author

Jeroen van Meeuwen (RHCE, LPIC-2, MCP, CCNA) is currently a Senior System Engineer, specialized in Linux systems and Systems Architecture, working for Operator Groep Delft in The Netherlands. His experience with computers goes back to the early '90s, with a Philips P2000T being over a decade old, little tapes containing programs but most importantly games, and 16K memory cartridges. Since 1998, he has been involved with Red Hat Linux (5.2 at that time), and was an early adopter of Fedora Core Linux in November 2003, until his first real contributions to Free and Open Source Software were made in 2005.

As a contributor to Free and Open Source Software within the Fedora community, amongst other programs, Jeroen has developed Revisor, a Python framework to build distributions with. With regards to Configuration Management, Jeroen currently maintains or co-maintains -amongst other packages- the entire stack of packages related to Puppet

2. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

2.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.
>
> A useful shortcut for the above command (and many others) is **`Tab`** completion. Type **`cat my_`** and then press the **`Tab`** key. Assuming there are no other files in the

[1] https://fedorahosted.org/liberation-fonts/

> current directory which begin with 'my_', the rest of the file name will be entered on the command line for you.
>
> (If other file names begin with 'my_', pressing the **`Tab`** key expands the file name to the point the names differ. Press **`Tab`** again to see all the files that match. Type enough of the file name you want to include on the command line to distinguish the file you want from the others and press **`Tab`** again.)

The above includes a file name, a shell command and two key caps, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F1`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F7`** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`Mono-spaced Bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **`ssh`** ***`username`*****`@`*****`domain.name`*** at a shell prompt. If the remote machine is **`example.com`** and your username on that machine is john, type **`ssh john@example.com`**.
>
> The **`mount -o remount`** ***`file-system`*** command remounts the named file system. For example, to remount the **`/home`** file system, the command is **`mount -o remount /home`**.
>
> To see the version of a currently installed package, use the **`rpm -q`** ***`package`*** command. It will return a result as follows: ***`package-version-release`***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

2.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
```

```
        Echo            echo   = home.create();

        System.out.println("Created Echo");

        System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
     }

  }
```

2.3. Notes and Warnings

Finally, we use three distinct visual styles to highlight certain information nuggets.

Note

A note is useful bit of information: a tip or shortcut or an alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

The Important information box highlights details that are easily missed: such as configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring important information won't cause data loss but may cause irritation and frustration.

Warning

A Warning highlights vital information that must not be ignored. Ignoring warnings will most likely cause data loss.

3. Feedback

Should you find any discrepancies or additional information for this documentation, we would appreciate to hear from you.

Our mailing lists are:

http://1lists.fedorahosted.org/1mailman/1listinfo/1courses-users/
Our "users" mailing list where anyone can comment on the course materials offered, provide other means of feedback and ask questions when things appear to not be as clear as they intend to be.

http://1lists.fedorahosted.org/1mailman/1listinfo/1courses-devel/
Our development mailing list for anyone seeking to get involved in the project.

http://1lists.fedorahosted.org/1mailman/1listinfo/1courses-commits/
This mailing list is used to send any changes made to any of the documents to anyone subscribed.

Introduction

Welcome to the Puppet Workshop (or Configuration Management workshop). Today's workshop is comprised of the following topics, in order of appearance:

Topic
Introduction to Configuration Management
Introduction to Puppet
Puppet Terminology
How Puppet Works
Puppet Features
Troubleshooting Puppet
Setting up Puppet
How to use Puppet
Other Things To Do With Puppet
Best Practices

1.1. Target Audience

The primary audience for this book is, of course, Linux system administrators and engineers seeking to implement, further enhance or extend their knowledge about configuration management in general and by using the next-generation configuration management utility Puppet in particular.

Introduction to Configuration Management

2.1. What is Configuration Management?

Within virtually every organization, there's probably a number of systems running Linux, Solaris, Mac OS X or HP-UX. All these machines need to be configured to be able to function properly. Some will need special drivers, and all of them will need correct DNS settings, certain packages installed and certain other packages removed. Most probably, the more systems, the more these diverge in the configuration they need, and potentially diverge in the way this configuration needs to be applied to a given operating system or operating system version.

More specifically, an organization may have a couple of webservers, fileservers, a DNS and a DHCP server, a number of desktop PCs, and a number of laptops. The laptops may need slightly different system configuration (no LDAP authentication, and with a VPN client installed, for example), and the desktop PCs may need different applications installed then the servers, and so forth. Yet, between, say, a hundred desktop PCs, you would want the configuration to be as similar as possible. You may want to diverge between a software developer's desktop PC and a desktop PC in Human Resources, but in essence these are desktop profiles diverging on the application level, applied upon a stable system configuration which remains the same, or similar at least.

By the time the organization grows, replaces the hardware, upgrades to another version of the operating system, or applies changes, the challenge to making everything work yet maintain a similar configuration between all nodes becomes bigger. While every attempt made to control the situation can be called a form of configuration management, the solution without a configuration management framework is often comprised of:

1. a number of scripts (with or without revision control), to move around files, install packages, perform daily check-ups,
2. NFS mounts with programs pre-installed, so that nodes can mount these NFS shares and the software needs to be provided once, in one location, for all to share,
3. file server shares with pre-compiled drivers, or driver sources being compiled on the nodes by scripts running on the nodes,
4. terminal servers or desktop servers like with FreeNX, so that configuration concentrates on a smaller number of boxes

This means that work-arounds for actual (user) problems maybe require an additional if-then-else in one or the other script, and updates to programs installed require manual compilation and installation. The success rate of these solutions never reaches 100%, and as it turns out the longer such a implemented solution runs, the more exotic problems become and the more machines will fail to remain up-to-date regardless of any attempt made to fix the issue; simply because it becomes to diversive and unmaintainable.

2.1.1. Configuration Management

Generally speaking, with configuration management, it's about managing the configuration of one or more organizational resources in order to have it be in a state in which it can perform the operations required by, and possibly critical to, the organization's operations.

In this workshop though, we are not going to explore configuration management of a coffee machine. Instead we look at the computers in a network running any platform but the one from a prominent proprietary North America-based vendor. We are talking automation and further enhancement of Computer Systems Administration.

When managing the operating system and software running on mainframes, servers, desktop PCs and laptops, you may find yourself looking for answers to questions such as:

- How do I manage what packages are installed on a given system?
  - How do I manage the configuration of those packages (this software)?
  - How do I make sure these packages are updated?
- How do I make sure the services that every machine needs to run are actually running?
- How do I manage monitoring the services or a machine's state?
- A job needs to run periodically (maybe via **crontab**), but how do I make sure it is run, and how can I change or remove the job later?
- Given different operating systems and operating system versions, how do I make sure I apply the correct routine for adding a user, starting a service, install/update/remove a package?

2.1.2. Configuration Management Requirements

This section is about what you would want Configuration Management to do for you:

Maintain consistency across systems

Consistency across systems is key in understanding where a problem might come from. If each and every system is unique, you may end up searching for unique aspects of the system's configuration in order to determine the cause of a problem, while if systems are consistent to some extend, you may have found the problem even before your users report it.

Consistency !== Equality

Of course keeping system consistent in their configuration doesn't say all your systems should be entirely equal, because that would not be feasible for many organizations and defeat the purpose of configuration management. Needless to say though, having all systems be entirely unique defeats part of the purpose of configuration management as well.

Categorize systems

Categorizing systems into categories like (for example) *desktop*, *server* and/or *laptop*, helps in applying changes to one category, such as installing **GNOME** or keeping systems up-to-date according to a schedule that may (servers) or may not (desktops, laptops) need a service or maintenance window.

Different profiles

More generally speaking, different profiles for each of these categories may be defined as well, of course. A developer's desktop most likely has different requirements then a publicly accessible booth at the reception desk.

- **Version Control**

  Version control lets you keep track of changes applied to the overall configuration management framework, which is important because since you are managing different aspects of a number of systems, if something goes wrong the changes applied to the configuration of puppet will most likely be the first clue as to what caused the new problem and lets you recover relatively fast.

- **Overview of systems' tasks and services**

  Being able to quickly tell what a system does exactly, and how it differs from another system not only aids in performing risk assessments (impact of a given change), but may also help in determining the impact of a change beforehand, as well as determine the impact of an unexpected system interruption. Providing an example to the latter I suppose if you update httpd across systems (whether tested or untested), but the new software version doesn't work as expected, a configuration management framework should be able to quickly give you an overview of impacted systems and services.

- **Updating systems**

  Some systems can be updated irregularly, such as desktop PCs, but need to be kept up-to-date nonetheless. Other systems need to have service and/or maintenance windows, such as servers.

2.2. Problems without Configuration Management

There's a number of challenges in applying configuration management, such as:

1. **Different operating systems**

   If you have a diverse organization in terms of the operating systems your systems run, applying the same configuration items to a set of different operating systems is challenging in that adding a user or setting a password on one operating system is not the same as adding a user or setting a password on another operating system. The same applies to installing, updating or removing a package, and so forth. Additionally the more different operating systems you have, the harder managing any given system resource becomes. Some commands for day-to-day administrative tasks may be equal, or similar, but most of them are and/or behave different.

2. **Different distributions**

   Although an organization may not have different distributions running right now, sooner or later, an organization will migrate from one distribution to another; That is practically inevitable. If an organization does have different distributions running, practical problems such as the location of certain files become evident, as well as different interfaces to resource-management (like adding a user with **useradd** or **adduser**).

3. **Different versions of distributions**

   Different versions of distributions, or more accurately the different versions of the utilities, as well as the configuration settings for updated programs that come with the distributions, can form a challenge when or if the organization does not have a proper configuration management framework in place. Note that even though an organization may not have different versions of a distribution right now, at some point the organization will need to upgrade to the next available release.

4. **Different tasks to perform**

   Each different system in an organization is performing one or more tasks that may be unique to the system or may be shared between a group of systems, but with many different tasks being performed throughout the organization's infrastructure, keeping track of what system performs which task, keeping these systems up to date and configuring them to have the required packages installed for each of the tasks they perform, tackling the problem becomes harder.

5. **Different ways to perform a task**

   Within an organization that has multiple servers performing the same task, keeping a similar state or perform a task in a similar manner is challenging in that without configuration management, you are most likely to find three or more ways to purge old files from `/tmp/` and `/var/tmp/`, for example. The same differentiation may apply to how webservers' VirtualHost's are configured, or how a NFS share is mounted (mount options in particular).

6. **Different nodes**

   This one goes to hardware-specific needs and configuration. When each of the systems in an organization are not all of the same brand, make and model, or each system has different harddisk layouts, or needs different videocard drivers, you are basically keeping lists and making choices based on this list.

7. **Different services**

   Different services of course are configured differently, as far as configuration file locations and syntax are concerned. However, figuring out the best way to apply certain configuration to a system for each service is less efficient without configuration management. You might adjust a script or two and/or adjust the source repository from which you pull updates to each machine, but the changes may turn out to only apply to that system that needed the exception to the rule instead of focussing on a more general solution to the problem once, and apply that solution multiple times, over and over again.

8. **Interfaces to a system resource**

   This is probably the hardest one if you are not using any configuration management framework. Given different operating systems, distributions and/or distribution versions, in which case any combination of the three only makes the problem harder to solve, you are most likely to encounter so many different ways to manage a given system resource, that a simple script or routine cannot cover all of them -and remain comprehensible and maintainable. One example is adding a user to the system, and making the user a group member of several groups. You may find routines ranging from using **useradd** or **adduser** depending on the distribution used, to writing out ldifs from a template and using **ldapadd** or **ldapmodify** depending on whether the user already exists or not.

2.3. Not So Technical Aspects

In addition to the problems you may encounter with or without configuration management, there's a number of problems or challenges that are not so technical, but you may want to see resolved by a configuration management utility;

1. **Applying changes**

   Applying changes to multiple machines at once may become a problem depending on the size of the organization or the amount of control that you have over systems, remotely. There was a time when changing the DNS servers for a set of systems required one to log on to the console of each system and edit `/etc/resolv.conf` manually. You can see the problem become bigger if the organization does not have 20 systems, but 1200.

2. **Keeping track of changes**

   Another challenge is keeping track of the changes applied to each system. Even with configuration management, errors can be made and systems might behave unexpectedly, in which case you will want to know what changed on these systems, and how to recover to an operational state. Keeping track of changes without a configuration management framework however is a little harder, but with configuration management, you have reports (changes applied to a system in a nice overview), and most advisebly you have the configuration for Puppet stored in a Source Control Management system, or SCM system, like CVS, SVN, Mercurial, or GIT.

3. Staging changes

Staging changes is a huge must-have in case changes are radical or might destroy a normal system's operation (even if temporary). For such changes, you would want to test the changes first, and with Puppet, you get this in the form of *environments*. Additionally, in case any critical component needs to change, proper Change Management then requires you to Build & Test the solution prior to implementation, often not a very bad idea to relieve stress in case the implemented solution does not work, especially if the change is time-constrained such as with service windows.

Introduction To Puppet

Puppet is a solution to many of the problems set forth in *Section 2.2, “Problems without Configuration Management”*, and thus perfect for a workshop on Configuration Management.

Another solution may be *CFEngine*. We have chosen not to use CFEngine for several reasons:

- Puppet has an open development model, whereas CFEngine has not. This means that the changes and bugfixes, and more importantly innovation and development is in the hands of you and me.

- The level of abstraction of system resources that Puppet enables you to use allows you to concentrate on the bigger picture, rather then needing to figure out again and again, and then specify again and again, how a certain task is to be performed on a given operating system, distribution and/or specific distribution version. CFEngine however is a very low-level utility, perfect for keeping 800 identical machines in shape, but becomes worse with any desirable discrepancy between systems because of that low-level management.

For a more detailed CFEngine vs. Puppet poem, visit *http://1reductivelabs.com/1trac/1puppet/1wiki/1CfengineVsPuppet*.

3.1. What Does Puppet Do?

Puppet offers a high-level abstraction of system resources like you would encounter on any given system, such as users, services and packages. Seeing as how different operating systems and different distributions each have different interfaces (*providers* in puppet terms), to these system resources, managing a package to be installed, updated, removed or be of a certain version includes a lot of `if-then-else` statements in a script you would write to manage that particular system resource; one package.

On Debian, Ubuntu and derivative distributions for example, the package provider may be **apt**, **dpkg**, **smart**, **alien**, **PackageKit**, while on Fedora, Red Hat and it's derivatives, the package provider may be **rpm**, **yum**, **PackageKit**, **apt** or **smart**. Although some of these package managers can be combined, while others can not, and systems usually stick to their natively integrated package manager, figuring out such while actually trying to manage the result of what a package manager does could be seen as a lot of work for little gain.

Another difference between distributions is how services can be started, or configured to start up when the machine boots. A **service** script may be available, or **`/etc/init.d/`** may contain scripts to start and stop a service. Also, some of these service providers may have `status`, `reload` and `restart` command parameters, whereas others may not have. Additionally, using **chkconfig** to configure the runlevels the service should be enabled or disabled in may not be available on all systems.

By abstracting these system resources into *types*, Puppet takes on the headaches for most operating system and distribution specific interfaces to managing these system resources. It knows, or figures out all by itself, what provider to use given a *type*.

Abstraction of system resources

Abstraction of the system resources into so-called *types* causes the administrator to only need to configure a type, such as *package*, *user*, *cron*, and so forth. The configuration management utility itself will figure out what package manager backend to use, whether it's apt, yum, rpm, dpkg, smart or PackageKit.

Puppet example to ensure user *sysadmin* exists on a system:

```
user { "sysadmin":
ensure => present
}
```

Puppet example to ensure the *ypbind* package is installed and the most recent version, *ypbind* is correctly configured, and the *ypbind* service is running:

```
package { "ypbind":
ensure => latest
}

file { "/etc/yp.conf":
source => "puppet://$server/files/yp.conf",
notify => Service["ypbind"],
require => Package["ypbind"]
}

service { "ypbind":
enable => true,
ensure => running,
require => [
    File["/etc/yp.conf"],
    Package["ypbind"]
]
}
```

The above example is called a *manifest*, built out of *types* (package, file, service), which, once defined in a manifest, are referred to as *resources*. See also *Appendix A, Puppet Terminology*

Puppet Terminology

Terminology used in this documentation. See also *Appendix A, Puppet Terminology*

class

A class is a collection of resources applied to a node with a single include statement. It groups together a comprehensible set of resources. A class *ypclient* would manage the `File["/etc/nsswitch.conf"]`, `File["/etc/yp.conf"]`, `Package["ypbind"]`, and `Service["ypbind"]` resources.

fileserver

The fileserver is where the puppet pulls files from. It is normally integrated with the puppetmaster, but it can be an entirely different server, too.

The fileserver serves files to puppets that request them, but it also serves *templates*, which are parsed on the fileserver (puppetmaster), and passed on to the client as a whole new file.

manifest

The collection of classes, modules and resources that the *puppetmaster* uses to distribute the appropriate configuration to a *puppet*.

module

A module is a placeholder for files, manifests, plugins and templates. Creating a module has numerous advantages such as separate version control, separate staging from development through testing to production, and so forth.

See also: *Section 9.1, “Using Modules”*, *Section 9.2, “Using Plugins”*

node

The client, a node, is an operating system instance running the puppet client application. This can be a regular operating system running directly on top of actual hardware, a virtual guest as well as a virtual host.

puppet

The client, a node, runs the **puppetd** daemon or service, and is referred to as the *puppet*

puppetmaster

The puppetmaster is the node that runs the server-side application to a puppet setup.

resource

A resource is an instantiated *type*. It has been defined and it cannot be undefined. The puppetmaster sends all applicable resources the a puppet, which then applies them. Resources are fundamentally built from a *type*, a *title*, and a list of *attributes*, with each resource type having a specific list of supported attributes.

system resource

A system resource is a resource available on the node whether it is managed by puppet or not. Unlike what is otherwise understood by system resources, the puppet definition of system resources throughout this documentation does not so much refer to hardware resources like CPU or memory, but rather to manageable aspects of the operating system, like users, packages, services, files, cronjobs, and so forth.

type

Puppet uses *types* to abstract system resources. Types have parameters such as `ensure => present|absent` in case of a user, or `ensure => installed|absent|latest|`*`1.0-1.el5`*, indicating in which state the system resource should be. Each type has a title, which must be unique throughout the manifest, and a list of supported attributes. E.g., there is no `mode => 644` to the package type.

How Puppet Works

This is an overview of how puppet works -in a working setup.

1. **The puppet starts for the first time**

   It generates a certificate using the node's FQDN.

   Note

   Although not required, it is strongly recommended to have the client use a FQDN that is registered in DNS (forward as well as reverse).

2. **The puppet submits the certificate to the puppetmaster**

   The puppetmaster, also the Certificate Authority, or *puppetca*, needs to sign the certificate before the client can be considered authenticated.

3. **The puppet waits 300 seconds for a signed certificate**

   It this configurable timeout of 300 seconds[1] has passed, the puppet quits.

4. **The puppetmaster signs the certificate**

   To do so, you can either configure the puppetmaster to automatically sign certificates or sign manually. Automatically signing certificates is generally a very bad idea. To manually sign a certificate, use:

   ```
   # puppetca --sign <fqdn>
   ```

5. **The puppet receives the signed certificate**

   Immediately thereafter, the puppet starts a configuration run.

   Warning

   The time on both the puppetmaster and the puppet must be within 5 minutes of eachother as the certificate generated and signed has a validity period. If the difference in time of these two nodes is more then 5 minutes, you will get a "Certificates not trusted" type of error.

6. **The puppet generates all the facts**

   Most configurations rely on client information to make decisions. When the Puppet client starts, it loads the Facter Ruby library, collects all of the facts that it can, and passes those facts to the interpreter. When you use Puppet over a network, these facts are passed over the network to the server and the server uses them to compile the client's configuration.

7. **The puppetmaster parses it's manifests**

   The puppetmaster parses through all it's manifests, including the manifests not applicable to the puppet that is polling. It only sends out the manifest applicable to the puppet polling, however.

8. **The puppet receives the manifests**

   When the puppet receives the manifests, it may still contain variables such as `$hostname`, `$operatingsystem` and others, which the puppet fills out with the appropriate values.

9. **The puppet applies the manifest**

   While the puppet applies the manifest, it pulls files from the puppetmaster's *fileserver* after checking the local checksum against the remote checksum. When running with debug output, this will show as

   ```
   debug: Calling fileserver.list
   debug: //Node[node1.example.com]/File[/tmp/foo]/checksum: Initializing
    checksum hash
   debug: //Node[node1.example.com]/File[/tmp/foo]: Creating checksum
    {md5}85e53dc9439253a1ec9ca87aeffd9b0b
   debug: Calling fileserver.describe
   ```

10. **Files that are replaced are backed up**

    The puppet sends a copy of the files it replaces back to the puppetmaster.

11. **The puppet reports to the puppetmaster**

    A detailed report of what the puppet has done with the manifests is sent back to the puppetmaster.

12. **The puppet waits for 30 minutes**

    The next run the puppet performs/polls for is after a configurable timeperiod, which defaults to 30 minutes.

A puppet setup is comprised out of the following parts:

The Puppetmaster

The puppetmaster of course is the core element in a puppet setup. Not only is it responsible for the handing over the manifest to the client, it also takes care of serving the files needed by the manifest, as well as

Puppet Features

paragraph

Troubleshooting Puppet

This section is about troubleshooting the puppetmaster and puppet

Setting Up Puppet

In this section, we are going to set up a puppetmaster, and a puppet client. The puppetmaster is going to run the *mongrel* server-type, for setting up a puppetmaster for larger environments.

8.1. Installation

The default server type for the puppetmaster is called *webrick*, a single-threaded webserver. The webserver handles the puppets' requests for manifests, certificate exchanges, as well requests for files and templates. Being single-threaded, the webrick webserver can only handle one client at a time. While the puppets poll the puppetmaster with a default interval of 30 minutes, and configuration runs can take longer then 60 seconds, putting more then 25 clients in front of a puppetmaster with a webrick webserver is a very, very bad idea.

There is a multi-threaded webserver in Ruby, called *mongrel*. This is a simple, multi-threaded, but not very feature-rich webserver. For one, it does not perform SSL. For scalability purposes though, the mongrel server type is an absolute must, and can better be chosen as the webserver to handle the puppets' requests, right from the beginning. This however requires a frontend that performs the SSL part of the communications between the puppetmaster and the puppets. We choose Apache's HTTPd for it's excellent performance, flexible configuration, excellent configuration syntax, and because it can be set up as a reverse proxy load balancer, allowing more then one puppetmaster behind the scenes if necessary.

Install the required packages for the puppetmaster:

Smaller organizations (< ~25 clients)

- The puppetmaster.

```
# yum install puppet-server
```

- (optional) A database server (one of MySQL, SQLite3 or Postgresql), and the appropriate Ruby library. During this workshop, we use MySQL.

```
# yum install mysql-server ruby-mysql
```

- (optional) The Ruby RRDtool library.

```
# yum install ruby-RRDtool
```

Larger organizations (> ~25 clients)

- A webserver capable of performing as a frontend SSL reverse proxy load balancer, such as the Apache HTTPd webserver.

```
# yum install httpd
```

- The Ruby mongrel library, for better scalability.

```
# yum install rubygem-mongrel
```

- The puppetmaster.

```
# yum install puppet-server
```

- (optional) A database server (one of MySQL, SQLite3 or Postgresql), and the appropriate Ruby library. During this workshop, we use MySQL.

```
# yum install mysql-server ruby-mysql
```

- (optional) The Ruby RRDtool library.

```
# yum install ruby-RRDtool
```

8.2. Configuration

In this section, we walk you through the initial configuration of a puppetmaster with the mongrel server type.

8.2.1. Configuring the Puppetmaster

The configuration file for puppet and puppetmaster is **`/etc/puppet/puppet.conf`**. It is a file in INI-like format with sections, keys and values. There's 4 sections of interest,

[main]
: Primarily file locations, directory settings and other globals applicable to both the puppet as well as the puppetmaster.

[puppetca]
: Puppet Certificate Authority (puppetca) settings.

[puppetd]
: Puppet client daemon settings.

[puppetmasterd]
: Puppetmaster daemon settings.

8.2.1.1. Relevant Settings

Relevant Settings For The First Run

For the first run of the puppetmaster, the following settings require configuration:

[main]
: The locations where puppet seeks it's configuration and puts it's transitional data. The most important setting is **`vardir`**, which should be set to **`/var/lib/puppet/`**. Further settings include:
  - `logdir = /var/log/puppet/`

- `rundir = /var/run/puppet/`
- `ssldir = $vardir/ssl/`

> **Note**
> If you used a package to install puppet, the defaults should work, but may not comply with your backup strategy. It is the upstream puppet package that cannot cater to each and every distribution or operating system it is available for, and therefore has a set of defaults that will work, but will need to be changed on most platforms.

[puppetmasterd]

- **certname**

  The puppetmaster certificate's Common Name (CN), for which by default the system's hostname is used. The hostname of the system is a pretty reasonable value.

- **certdnsnames**

  A colon (`:`) seperated list of DNS names resolving to the puppetmaster. Include here:

  1. The short hostname of the system, using the output of:

     ```
     # hostname -s
     ```

  2. **`puppet`**

  3. **`puppet`**, followed by the DNS domain name of the system, using the output of

     ```
     # dnsdomainname
     ```

  4. Any other hostname or fully qualified domain name you want to use for the puppetmaster.

- Another setting to check is whether or not this puppetmaster is going to be the Certificate Authority

  ```
  [puppetmasterd]
      ca = true
  ```

  The default is often set to `true`.

- Whether or not to use autosigning of certificates, using

  ```
  [puppetca]
      autosign = false
  ```

  The default is to *not* use autosigning. Only applicable if `puppetca` is set to `true`.

Other Relevant Settings

The following settings require review before the puppetmaster is going in production.

- A list of environments using a comma seperated list, in

```
[puppetmasterd]
    environments = development,testing,production
```

See also: Section 9.3, “Environments”

- Whether or not to use reporting, and what reporting to use (tagmail, store, rrdgraph). To configure the types or reports that should be used by the puppetmaster, use a comma separated list without spaces, in:

```
[puppetmasterd]
    reports = tagmail,store,rrdgraph
```

  See also: Section 10.1, “Tweaking Reporting”

- The location of tagmail.conf, in order to map tags you give to resources to email addresses the reports should be sent to;

```
[main]
    tagmap = /path/to/tagmail.conf
```

  for reporting changes applied to puppets, via email.

  See also: Section 10.1, “Tweaking Reporting”

8.2.1.2. Minimal site.pp

Create a minimal `site.pp` in **`/etc/puppet/manifests/site.pp`** for the puppetmaster to parse on it's initial startup. Below is an example.

```
#
# site.pp for any domain
#

$server = "master.puppetmanaged.org"

# The default node

node default {
}
```

8.2.1.3. Service Configuration

On Red Hat based systems, use **`/etc/sysconfig/puppetmaster`** to configure the service. It has three variables set, of which `PUPPETMASTER_MANIFEST` needs to point to the default manifest to use.

8.2.2. Configuring the SSL Frontend Reverse Proxy Load Balancer

A webserver needs to be configured to handle the SSL XML-RPC requests from the puppets, because the mongrel server type is not capable of performing SSL.

The webserver is going to listen on port 8140, the default port for the puppetmaster to listen for clients. It is going to forward traffic (after being decrypted) to the puppetmaster on 127.0.0.1:8141.

8.2.3. Configuring the Database Server

para

8.2.3.1. SQLite3

para

8.2.3.2. MySQL

para

8.2.3.3. PostgreSQL

para

How To Use Puppet

This is a first section

9.1. Using Modules

About using modules

9.2. Using Plugins

About the use of plugins

9.3. Environments

paragraph

Other Things To Do With Puppet

This is a first section

10.1. Tweaking Reporting

paragraph

10.2. Writing Custom Types

paragraph

10.3. Writing Custom Facts

paragraph

10.4. Writing Custom Functions

paragraph

Chapter 11.

Best Practices

This is a first section

Part I. Appendices

Appendix A. Puppet Terminology

class
: A class is a collection of resources applied to a node with a single include statement. It groups together a comprehensible set of resources. A class *ypclient* would manage the `File["/etc/nsswitch.conf"]`, `File["/etc/yp.conf"]`, `Package["ypbind"]`, and `Service["ypbind"]` resources.

fact
: A client-side generated aspect of the node the puppet client runs on. Example facts are the amount of available memory, the hostname, the fully qualified domain name, the operating system (version).

manifest
: The collection of classes, modules and resources that the *puppetmaster* uses to distribute the appropriate configuration to a *puppet*.

module
: module

node
: The client, a node, is an operating system instance running the puppet client application. This can be a regular operating system running directly on top of actual hardware, a virtual guest as well as a virtual host.

puppet
: The client, a node, runs the **puppetd** daemon or service, and is referred to as the *puppet*

puppetmaster
: The puppetmaster is the node that runs the server-side application to a puppet setup.

resource
: A resource is an instantiated *type*

system resource
: A system resource is a resource available on the node whether it is managed by puppet or not. Unlike what is otherwise understood by system resources, the puppet definition of system resources does not so much refer to resources like CPU or memory, but rather to whether or not a package is installed or what version of said package, or the $osversion, and so on and so forth.

type
: definition

Appendix B. Example SSL Frontend Reverse Proxy Load Balancer Configuration

```
<ifModule !mod_proxy.c>
    LoadModule proxy_module modules/mod_proxy.so
</IfModule>

<IfModule !mod_proxy_http.c>
    LoadModule proxy_http_module modules/mod_proxy_http.so
</IfModule>

<IfModule !mod_proxy_balancer.c>
    LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
</IfModule>

<IfModule !mod_headers.c>
    LoadModule headers_module modules/mod_headers.so
</IfModule>

<IfModule !mod_ssl.c>
    LoadModule ssl_module modules/mod_ssl.so
</IfModule>

<IfModule !mod_authz_host.c>
    LoadModule authz_host_module modules/mod_authz_host.so
</IfModule>

<IfModule !mod_log_config.c>
    LoadModule log_config_module modules/mod_log_config.so
</IfModule>

<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>

<Proxy balancer://master.puppetmanaged.org>
  BalancerMember http://127.0.0.1:8141 keepalive=on retry=30
</Proxy>

<VirtualHost *:8140>
    ServerName master.puppetmanaged.org    SSLEngine on
    SSLCipherSuite SSLv2:-LOW:-EXPORT:RC4+RSA
    SSLCertificateFile      /var/lib/puppet/ssl/
certs/master.puppetmanaged.org.pem
    SSLCertificateKeyFile   /var/lib/puppet/ssl/
private_keys/master.puppetmanaged.org.pem
    SSLCertificateChainFile /var/lib/puppet/ssl/ca/ca_crt.pem
    SSLCACertificateFile    /var/lib/puppet/ssl/ca/ca_crt.pem
    SSLVerifyClient optional
    SSLVerifyDepth  1
    SSLOptions +StdEnvVars

    # The following client headers allow the same configuration to work
 with Pound.
    RequestHeader set X-SSL-Subject %{SSL_CLIENT_S_DN}e
    RequestHeader set X-Client-DN %{SSL_CLIENT_S_DN}e
    RequestHeader set X-Client-Verify %{SSL_CLIENT_VERIFY}e

    <Location />
        SetHandler balancer-manager
        Order allow,deny
        Allow from all
    </Location>

    ProxyPass / balancer://master.puppetmanaged.org:8140/ timeout=180
    ProxyPassReverse / balancer://master.puppetmanaged.org:8140/
    ProxyPreserveHost on
    SetEnv force-proxy-request-1.0 1
    SetEnv proxy-nokeepalive 1

    ErrorLog  logs/master.puppetmanaged.org-balancer-error_log
    CustomLog logs/master.puppetmanaged.org-balancer-access_log combined
    CustomLog logs/master.puppetmanaged.org-balancer-ssl_request_log "%t
 %h \
                                %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r
\" %b"
</VirtualHost>
```

Appendix C. Revision History

Revision History
Revision 1.0