FREE eBook

LEARNING puppet

Free unaffiliated eBook created from
Stack Overflow contributors.

#puppet

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: puppet

It is an unofficial and free puppet ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official puppet.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with puppet

Remarks

This section provides an overview of what puppet is, and why a developer might want to use it.

It should also mention any large subjects within puppet, and link out to the related topics. Since the Documentation for puppet is new, you may need to create initial versions of those related topics.

Versions

Puppet Open Source

Version	Release Date
4.7.0	2015-09-22
4.6.0	2015-08-10
4.5.0	2015-05-17
4.4.0	2016-03-16
4.3.0	2015-11-17
4.2.0	2015-06-24
4.1.0	2015-05-19
4.0.0	2015-04-15
3.8.0	2015-04-28
3.7.0	2014-09-04
3.6.0	2014-05-15
3.5.0	2014-04-03
3.4.0	2013-12-19
3.3.0	2013-09-12
3.2.0	2013-05-14
3.1.0	2013-02-04

Version	Release Date
3.0.0	2012-09-28
2.7.0	2011-06-17
2.6.0	2010-07-20
0.25.0	2009-09-05
0.24.0	2007-12-13
0.23.0	2007-01-20
0.22.0	2007-01-06
0.20.0	2006-10-18
0.19.0	2006-09-07
0.18.0	2006-06-14
0.17.0	2006-05-16
0.16.0	2006-04-21
0.15.0	2006-03-13
0.14.0	2006-03-06
0.13.0	2006-02-09
0.12.0	2006-01-26
0.11.0	2006-01-17
0.10.0	2006-01-09
0.9.3	2006-01-03
0.9.2	2005-11-22

Examples

What is puppet and why should I care?

Puppet is a configuration management solution. Users describe the desired state of a server or software and configuration management achieves this state. This brings following advantages:

- Configurations can be reproduced exactly the same every time, as many times as necessary

- Configurations for all software and servers are stored in a central location. This makes backup and version control of configurations easily achievable
- Changes to all servers propagate through the entire infrastructure within a couple of minutes, without having to log in to any machine directly
- Everything is described in the same language, making it easy to configure new software
- Modules are similar to libraries and allow configurations to be consolidated. Modules for all major software packages already exist, making installing them extremely easy
- Servers can share information between each other, influencing the configuration of other servers. For example a new server can automatically register itself with the load balancer and monitoring solution

Puppet uses Ruby to describe the desired state of a server, called a **node**. It does so with the use of primitives called **resource types**. By default, every 30 minutes, the puppet **agent** authenticates itself against the puppet **server**. It then sends a list of properties of itself called **facts**. The server looks at the facts and the configuration files called **manifests** and compiles the desired state for the node. It then sends that **configuration** back to the node, where the agent applies it.

To give an idea of how powerful this can be, here are a couple examples of increasing complexity showcasing what puppet can do for you.

Example: User

This example creates the user *username* on node *myserver* and adds it to the group *wheel*.

```
node 'myserver' {
    user { 'username':
      ensure => 'present',
      groups => ['wheel'],
    }
}
```

This file that would be stored on the puppet *server* is the *manifest*. The *resource type* in this example is *user*. Every resource type has optional and required properties. In this example, *ensure* is required and *groups* is optional. This specific configuration would only be applied to *myserver*. You can apply configurations to all nodes by placing it outside of a node definition.

It is possible to take a couple of resource definitions and store them as **modules**. A module is similar to a library. These modules can be shared online, and you usually find one for every major software package. The official way to share modules is through the puppet forge: https://forge.puppet.com/

Example: Postgres

This example installs a postgres server on node *myserver*, and creates a database *db*, owned by *username*, identified by *password*. It does so using the postgresql module.

```
node 'myserver' {
    class { 'postgresql::server': }

    postgresql::server::db { 'db':
```

```
        user     => 'username',
        password => 'password',
    }
}
```

In this case *postgresql* is a module. The module itself takes care of identifying the operating system, downloading and installing the program, and then configuring it according to the manifest. This is a basic example but the module allows a great deal of customization.

Note that it is not necessary to know SQL or how to actually install a postgres server to do so. Official modules are well maintained and provide a sane and secure base configuration.

It is also possible to use *facts* in manifests. Facts act like variables.

Example: Conditions using facts

This example uses the rsyslog module to configure rsyslog on all non-windows machines.

```
if $osfamily != 'windows' {
  class { 'rsyslog::client': }
}
```

$osfamily is a fact. These facts are gathered every time the puppet agent runs. Note that because this definition is outside of a node definition, it gets applied to all nodes. However, *rsyslog::client* will only be executed on nodes that do not run windows.

Since puppet uses ruby, programmatic elements like control flows and variables can be used in manifests.

With the addition of **PuppetDB** you can share information between multiple nodes. This allows one node to influence configuration on a different node. Classic examples include load balancers or monitoring solutions.

Example: Registering a host with monitoring using exported resources

This example creates an **exported resource** on a node, and then imports that resource on the monitoring server, adding the host to the monitoring. It is using the Icinga2 puppet module.

```
@@icinga2::object::host { $::fqdn:
  display_name     => $::fqdn,
  ipv4_address     => $::ipaddress_eth0,
}

node 'icinga2' {
    Icinga2::Object::Host <<| |>> { }
}
```

@@icinga2::object::host creates a host definition object. This gets created by every node that executes this code. The @@ marks it as an *exported resource*. Usually, nodes do not share information in puppet. Exported resources allow to do that.

Note that all the property values in the host definition are facts. This means they will be different for every node which executes it.

Finally, the exported resource gets *imported* by the *icinga2* node. The Icinga2 module is responsible for making sure that the correct configuration files are created and reloaded.

Is it for you?

If you do deployments, configure your applications on multiple servers and required to login to your servers and make some changes in infrastructure, applications, pre-requisits etc. then puppet can definitely help you.

Except all this if you handle a big infrastructure and want a centralized management you can also have a look.

Before you startup

Before you decide to work on puppet there are few things that you need to know.

1. puppet work in both client-server architecture (widely used) as well single machine (specially for testing purpose)
2. puppet master can only be configured on a linux machine (master machine/node), windows can be used only as client (managed machine/node)
3. if configuring master, you must be aware of using linux machine and basic commands
4. puppet provides it's own configuration language that looks like json

Official Documentation

Puppet provide official documention for both open-source and enterprise versions. you can find it here

Installation

System Requirements

However, the Puppet master service is fairly resource intensive, and should be installed on a robust dedicated server.

- At a minimum, your Puppet master server should have two processor cores and at least 1 GB of RAM.
- To comfortably serve at least 1,000 nodes, it should have 2-4 processor cores and at least 4 GB of RAM.

Check your network configuration:

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

- **Firewalls:** The Puppet master server must allow incoming connections on port 8140, and agent nodes must be able to connect to the master on that port.
- **Name resolution:** Every node must have a unique hostname. Forward and reverse DNS must both be configured correctly.

  Note: The default Puppet master hostname is puppet. Your agent nodes can be ready sooner if this hostname resolves to your Puppet master.

  The time must be set accurately on the Puppet master server that will be acting as the certificate authority. You should probably use NTP.

Installing Puppet Server

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

Red Hat Enterprise Linux

- Enterprise Linux 6
- Enterprise Linux 7

Debian

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

Ubuntu

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

Enable the Puppet package repositories

Enterprise Linux 7

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

For other versions look here

Installing puppet master

```
yum install puppetserver
```

or

```
apt-get install puppetserver
```

Puppet Server is configured to use 2 GB of RAM by default. To change look here

Start the Puppet Server service:

```
systemctl start puppetserver
```

or

```
service puppetserver start
```

Read Getting started with puppet online: https://riptutorial.com/puppet/topic/2871/getting-started-with-puppet

Chapter 2: Agent

Syntax

1. puppet agent [--certname NAME] [-D|--daemonize|--no-daemonize] [-d|--debug] [--detailed-exitcodes] [--digest DIGEST] [--disable [MESSAGE]] [--enable] [--fingerprint] [-h|--help] [-l|--logdest syslog|eventlog|FILE|console] [--masterport PORT] [--noop] [-o|--onetime] [-t|--test] [-v|--verbose] [-V|--version] [-w|--waitforcert SECONDS]

Examples

What is it?

The puppet agent is a service that runs on the servers. Once the service is started, The agent will be triggered on background every 30 min (by default).

The agent have 2 main usages:

- Send server`s facts to the puppet master
- Receive catalog from the puppet master ans apply it

Trigger

By default the agent is triggered every 30 minutes. This interval value can be changed from the `puppet.conf` file.

- Linux- `/etc/puppet/puppet.conf`
- Windows - `%PROGRAMDATA%\PuppetLabs\puppet\etc\puppet.conf`

Set the `runinterval` to the wanted interval.

```
runinterval=xxx
```

The agent can be triggered manually with the command:

```
puppet agent –t
```

Verbose output

Sometimes it is helpful to get more output on puppet agent run.

It is very useful for debugging.

Run puppet agent with `verbose` and `debug` parameters:

- `debug` - Enable full debugging.

- `verbose` - Turn on verbose reporting.

```
puppet agent -t --verbose --debug
```

Logging

Puppet agnet logs messages. You can view this logs here:

Linux - `/var/log/puppet/puppet.log`

Windows - view the `Event Viewer` (Control Panel → System and Security → Administrative Tools → Event Viewer)

Read Agent online: https://riptutorial.com/puppet/topic/7234/agent

Chapter 3: Handling NFS Mount

Introduction

NFS is the most common way to share disk between computers in linux. It allows user on a client computer to access files over a network much like local storage is accessed. Here we see how to configure Puppet to manage mounting and serving NFS drives.

Parameters

Parameter	Details
name	The path to local directory in which the remote drive should be mounted.
device	Remote server address and directory path on remote server, separated by :
atboot	Whether this drive should be mounted while booting. Enabling makes drives available sooner, but may cause delayed boot in case of network or mounting problem.
pass	Fsck order is to tell fsck what order to check the file systems, if set to "0" file system is ignored. Usually NFS drives need not be checked in clients, so "0" is a suitable option.

Remarks

- Mount target directory should exists on the client.
- Mount resource type documentation
- fstab description and option details

Examples

Mounting a remote NFS drive

```
mount { '/path/to/local/folder':
  ensure  => 'mounted',
  atboot  => false,
  device  => 'server-ip-or-domain:/path/to/server/folder',
  fstype  => 'nfs',
  options => 'defaults',
  pass    => 0,
}
```

Read Handling NFS Mount online: https://riptutorial.com/puppet/topic/8044/handling-nfs-mount

Credits

S. No	Chapters	Contributors
1	Getting started with puppet	Ankit Katiyar, Community, Matthieu FAURE, Mor Paz, mzhaase, Peter Souter, Quill
2	Agent	Oz Bar-Shalom
3	Handling NFS Mount	Amir Ali Akbari