Puppet

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com


https://www.facebook.com/tutorialspointindia


https://twitter.com/tutorialspoint

About the Tutorial

Puppet is a configuration management technology to manage the infrastructure on physical or virtual machines. It is an open-source software configuration management tool developed using Ruby which helps in managing complex infrastructure on the fly.

This tutorial will help in understanding the building blocks of Puppet and how it works in an infrastructure environment. All the examples and code snippets used in this tutorial are tested. The working code snippets can be simply used in any Puppet setup by changing the current defined names and variables.

Audience

This tutorial has been prepared for those who want to understand the features and functionality of Puppet and how it can help in reducing the complexity of managing an infrastructure.

After completing this tutorial one would gain moderate level understanding of Puppet and its workflow. It will also give you a fair idea on how to configure Puppet in a preconfigured infrastructure and use it for automation.

Prerequisites

We assume anyone who wants to understand and learn Puppet should have an understanding of the system administration, infrastructure, and network protocol communication. To automate the infrastructure provisioning, one should have a command over basic Ruby script writing and the underlying system where one wants to use Puppet.

Copyright & Disclaimer

Table of Contents

Basic Puppet

1. Puppet – Overview

Puppet is a configuration management tool developed by Puppet Labs in order to automate infrastructure management and configuration. Puppet is a very powerful tool which helps in the concept of Infrastructure as code. This tool is written in Ruby DSL language that helps in converting a complete infrastructure in code format, which can be easily managed and configured.

Puppet follows client-server model, where one machine in any cluster acts as client known as puppet master and the other acts as server known as slave on nodes. Puppet has the capability to manage any system from scratch, starting from initial configuration till end-of-life of any particular machine.

Features of Puppet System

Following are the most important features of Puppet.

Idempotency

Puppet supports Idempotency which makes it unique. Similar to Chef, in Puppet, one can safely run the same set of configuration multiple times on the same machine. In this flow, Puppet checks for the current status of the target machine and will only make changes when there is any specific change in the configuration.

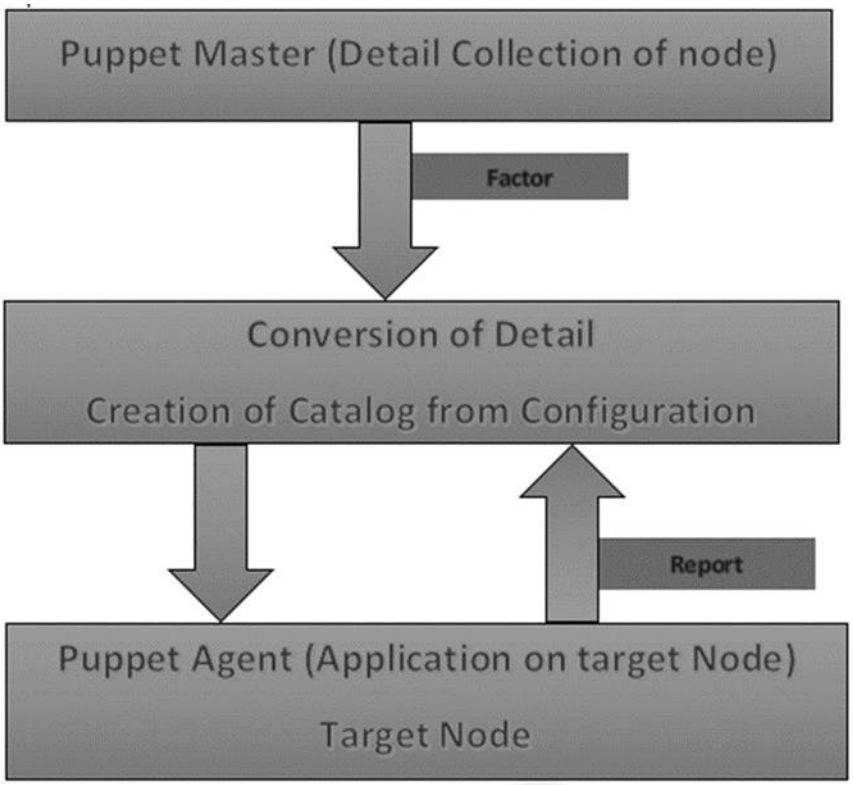
Idempotency helps in managing any particular machine throughout its lifecycle starting from the creation of machine, configurational changes in the machine, till the end-of-life. Puppet Idempotency feature is very helpful in keeping the machine updated for years rather than rebuilding the same machine multiple times, when there is any configurational change.

Cross-platform

In Puppet, with the help of Resource Abstraction Layer (RAL) which uses Puppet resources, one can target the specified configuration of system without worrying about the implementation details and how the configuration command will work inside the system, which are defined in the underlying configuration file.

Puppet – Workflow

Puppet uses the following workflow to apply configuration on the system.



- In Puppet, the first thing what the Puppet master does is to collect the details of the target machine. Using the factor which is present on all Puppet nodes (similar to Ohai in Chef) it gets all the machine level configuration details. These details are collected and sent back to the Puppet master.

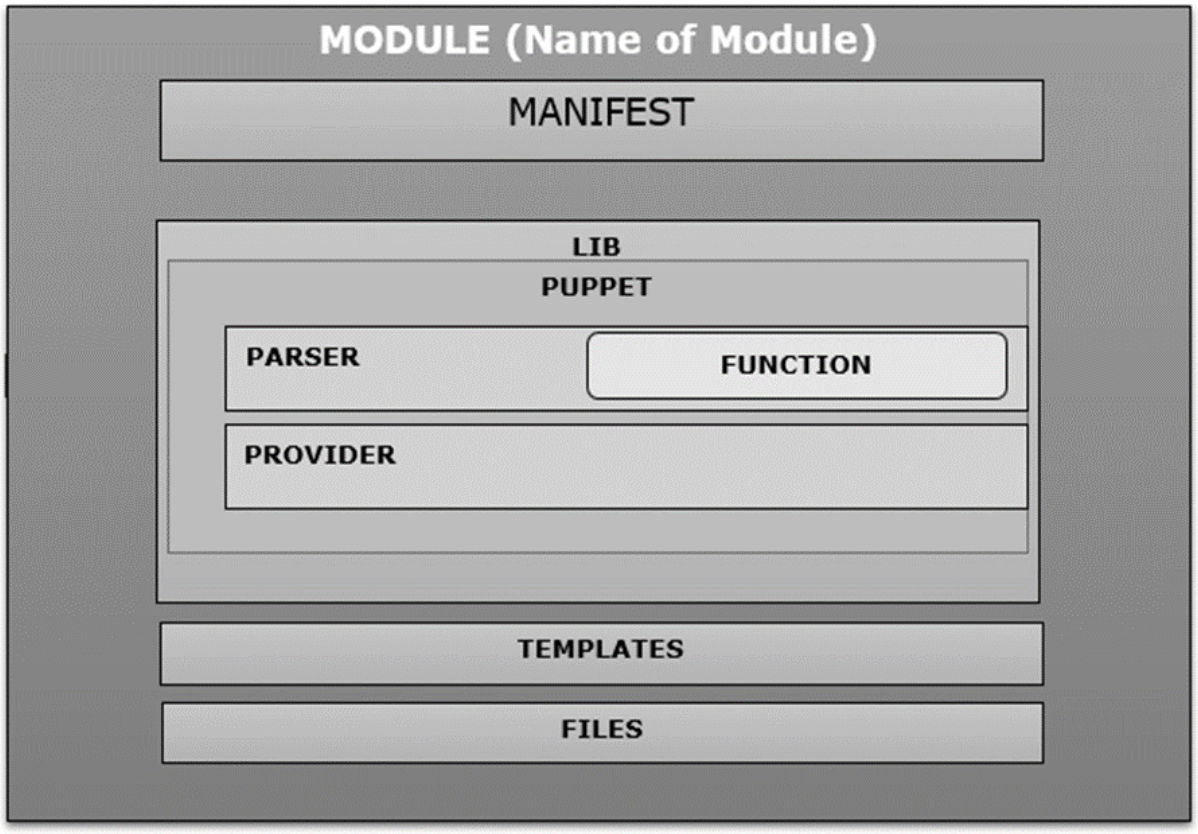
- Then the puppet master compares the retrieved configuration with defined configuration details, and with the defined configuration it creates a catalog and sends it to the targeted Puppet agents.

- The Puppet agent then applies those configurations to get the system into a desired state.

- Finally, once one has the target node in a desired state, it sends a report back to the Puppet master, which helps the Puppet master in understanding where the current state of the system is, as defined in the catalog.

Puppet – Key Components

Following are the key components of Puppet.



Puppet Resources

Puppet resources are the key components for modeling any particular machine. These resources have their own implementation model. Puppet uses the same model to get any particular resource in the desired state.

Providers

Providers are basically fulfillers of any particular resource used in Puppet. For example, the package type 'apt-get' and 'yum' both are valid for package management. Sometimes, more than one provider would be available on a particular platform. Though each platform always have a default provider.

Manifest

Manifest is a collection of resources which are coupled inside the function or classes to configure any target system. They contain a set of Ruby code in order to configure a system.

Modules

Module is the key building block of Puppet, which can be defined as a collection of resources, files, templates, etc. They can be easily distributed among different kinds of OS being defined that they are of the same flavor. As they can be easily distributed, one module can be used multiple times with the same configuration.

Templates

Templates use Ruby expressions to define the customized content and variable input. They are used to develop custom content. Templates are defined in manifests and are copied to a location on the system. For example, if one wants to define httpd with a customizable port, then it can be done using the following expression.

```
Listen <%= @httpd_port %>
```

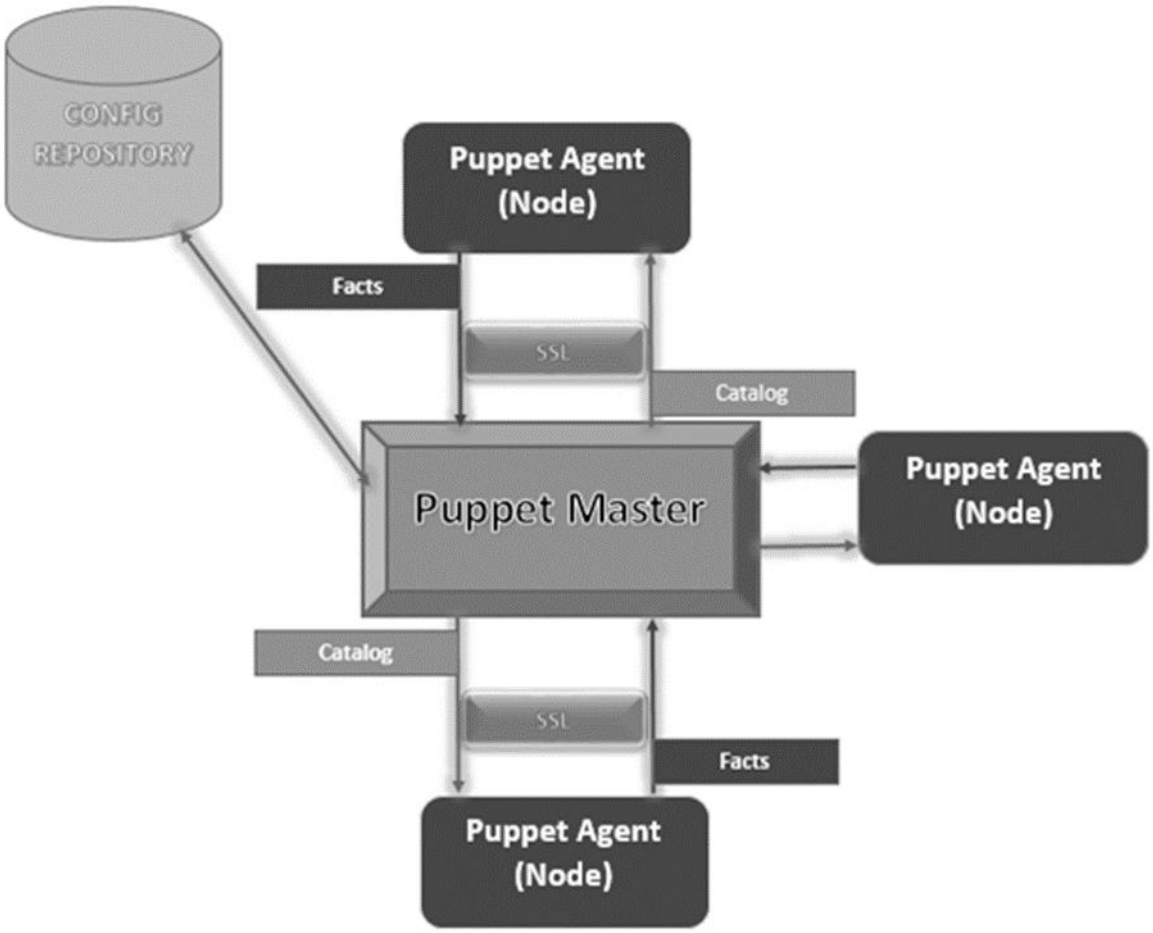
The httpd_port variable in this case is defined in the manifest that references this template.

Static Files

Static files can be defined as a general file which are sometimes required to perform specific tasks. They can be simply copied from one location to another using Puppet. All static files are located inside the files directory of any module. Any manipulation of the file in a manifest is done using the file resource.

2. Puppet – Architecture

Following is the diagrammatic representation of Puppet architecture.



Puppet Master

Puppet Master is the key mechanism which handles all the configuration related stuff. It applies the configuration to nodes using the Puppet agent.

Puppet Agent

Puppet Agents are the actual working machines which are managed by the Puppet master. They have the Puppet agent daemon service running inside them.

Config Repository

This is the repo where all nodes and server-related configurations are saved and pulled when required.

Facts

Facts are the details related to the node or the master machine, which are basically used for analyzing the current status of any node. On the basis of facts, changes are done on any target machine. There are pre-defined and custom facts in Puppet.

Catalog

All the manifest files or configuration which are written in Puppet are first converted to a compiled format called catalog and later those catalogs are applied on the target machine.

3. Puppet – Installation

Puppet works on the client server architecture, wherein we call the server as the Puppet master and the client as the Puppet node. This setup is achieved by installing Puppet on both the client and well as on all the server machines.

For most of the platforms, Puppet can be installed via the package manager of choice. However, for few platforms it can be done by installing the **tarball** or **RubyGems**.

Prerequisites

Factor is the only pre-requisite that does not come along with the standard package edition of Puppet. This is similar to **Ohai** which is present in Chef.

Standard OS Library

We need to have standard set of library of any underlying OS. Remaining all the system comes along with Ruby 1.8.2 + versions. Following is the list of library items, which an OS should consist of.

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

Facter Installation

As discussed, the **facter** does not come along with the standard edition of Ruby. So, in order to get the facter in the target system one needs to install it manually from the source as the facter library is a pre-requisite of Puppet.

This package is available for multiple platforms however just to be on the safer side it can be installed using **tarball**, which helps in getting the latest version.

First, download the **tarball** from the official site of Puppet using the **wget** utility.

```
$ wget http://puppetlabs.com/downloads/facter/facter-latest.tgz  ------: 1
```

Next, un-tar the tar file. Get inside the untarred directory using the CD command. Finally, install the facter using **install.rb** file present inside the **facter** directory.

```
$ gzip -d -c facter-latest.tgz | tar xf - -----: 2
$ cd facter-* ------: 3
$ sudo ruby install.rb # or become root and run install.rb -----:4
```

Installing Puppet from the Source

First, install the Puppet tarball from the Puppet site using **wget**. Then, extract the tarball to a target location. Move inside the created directory using the **CD** command. Using **install.rb** file, install Puppet on the underlying server.

```
# get the latest tarball
$ wget http://puppetlabs.com/downloads/puppet/puppet-latest.tgz -----: 1

# untar and install it
$ gzip -d -c puppet-latest.tgz | tar xf - ----: 2
$ cd puppet-* ------: 3
$ sudo ruby install.rb # or become root and run install.rb -------: 4
```

Installing Puppet and Facter Using Ruby Gem

```
# Installing Facter
$ wget http://puppetlabs.com/downloads/gems/facter-1.5.7.gem
$ sudo gem install facter-1.5.7.gem

# Installing Puppet
$ wget http://puppetlabs.com/downloads/gems/puppet-0.25.1.gem
$ sudo gem install puppet-0.25.1.gem
```

4. Puppet – Configuration

Once we have Puppet installed on the system, the next step is to configure it to perform certain initial operations.

Open Firewall Ports on Machines

To make the Puppet server manage the client's server centrally, one needs to open a specified port on all the machines, i.e. **8140** can be used if it is not in use in any of the machines which we are trying to configure. We need to enable both TCP and UDP communication on all the machines.

Configuration File

The main configuration file for Puppet is **etc/puppet/puppet.conf**. All the configuration files get created in a package-based configuration of Puppet. Most of the configuration which is required to configure Puppet is kept in these files and once the Puppet run takes place, it picks up those configurations automatically. However, for some specific tasks such as configuring a web server or an external Certificate Authority (CA), Puppet has separate configuration for files and settings.

Server configuration files are located in **conf.d** directory which is also known as the Puppet master. These files are by default located under **/etc/puppetlabs/puppetserver/conf.d** path. These config files are in HOCON format, which keeps the basic structure of JSON but it is more readable. When the Puppet startup takes place it picks up all .cong files from conf.d directory and uses them for making any configurational changes. Any changes in these files only takes place when the server is restarted.

List File and Settings File

- global.conf
- webserver.conf
- web-routes.conf
- puppetserver.conf
- auth.conf
- master.conf (deprecated)
- ca.conf (deprecated)

There are different configuration files in Puppet which are specific to each component in Puppet.

Puppet.conf

Puppet.conf file is Puppet's main configuration file. Puppet uses the same configuration file to configure all the required Puppet command and services. All Puppet related settings such as the definition of Puppet master, Puppet agent, Puppet apply and certificates are defined in this file. Puppet can refer them as per requirement.

The config file resembles a standard ini file wherein the settings can go into the specific application section of the main section.

Main Config Section

```
[main]
certname = Test1.vipin.com
server = TestingSrv
environment = production
runinterval = 1h
```

Puppet Master Config File

```
[main]
certname = puppetmaster.vipin.com
server = MasterSrv
environment = production
runinterval = 1h
strict_variables = true

[master]
dns_alt_names = MasterSrv,brcleprod01.vipin.com,puppet,puppet.test.com
reports = puppetdb
storeconfigs_backend = puppetdb
storeconfigs = true
environment_timeout = unlimited
```

Detail Overview

In Puppet configuration, the file which is going to be used has multiple configuration sections wherein each section has different kinds of multiple number of settings.

Config Section

Puppet configuration file mainly consists of the following config sections.

- **Main**: This is known as the global section which is used by all the commands and services in Puppet. One defines the default values in the main section which can be overridden by any section present in puppet.conf file.
- **Master**: This section is referred by Puppet master service and Puppet cert command.
- **Agent**: This section is referred by Puppet agent service.
- **User**: It is mostly used by Puppet apply command as well as many of the less common commands.

```
[main]
certname =PuppetTestmaster1.example.com
```

Key Components of Config File

Following are the key components of Config file.

Comment Lines

In Puppet, any comment line starts with (**#**) sign. This may intend with any amount of space. We can have a partial comment as well within the same line.

```
# This is a comment.
Testing= true #this is also a comment in same line
```

Settings Lines

Settings line must consist of -

- Any amount of leading space (optional)
- Name of the settings
- An equals **=** to sign, which may be surrounded by any number of space
- A value for the setting

Setting Variables

In most of the cases, the value of settings will be a single word but in some special cases, there are few special values.

Paths

In configuration file settings, take a list of directories. While defining these directories, one should keep in mind that they should be separated by the system path separator character, which is (**:**) in *nix platforms and semicolons (**;**) on Windows.

```
# *nix version:

environmentpath = $codedir/special_environments:$codedir/environments

# Windows version:

environmentpath =
$codedir/environments;C:\ProgramData\PuppetLabs\code\environment
```

In the definition, the file directory which is listed first is scanned and then later moves to the other directory in the list, if it doesn't find one.

Files and Directories

All the settings that take a single file or directory can accept an optional hash of permissions. When the server is starting up, Puppet will enforce those files or directories in the list.

```
ssldir = $vardir/ssl {owner = service, mode = 0771}
```

In the above code, the allowed hash are owner, group, and mode. There are only two valid values of the owner and group keys.

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**