**PUPPET AND HIERA DOCUMENTATION**

**Submitted by,**

**B Surabhi Suresh**

**Emp No. 60114103**

**PUPPET**

**What is Puppet?**

* A configuration management tool
* A framework for Systems Automation
* A Declarative Domain Specific Language (**DSL**)
* An OpenSource software in written Ruby
* Works on Linux, Unix (Solaris, AIX, \*BSD), MacOS, Windows ([Supported Platforms](http://docs.puppetlabs.com/guides/platforms.html))
* Developed by [Puppet Labs](http://puppetlabs.com/)

**Configuration management advantages**

* **Infrastructure as Code**: Track, Test, Deploy, Reproduce, Scale
* Code commits log shows the **history of change** on the infrastructure
* **Reproducible setups**: Do once, repeat forever
* **Scale** quickly: Done for one, use on many
* **Coherent** and consistent server setups
* Alternative to Puppet: [Chef](http://www.opscode.com/chef/), [CFEngine](http://cfengine.com/), [Salt](http://saltstack.com/), [Ansible](http://www.ansibleworks.com/)

**References and ecosystem**

* [Puppet Labs](http://puppetlabs.com/) - The Company behind Puppet
* [Puppet](http://puppetlabs.com/puppet/puppet-open-source/) - The OpenSource version
* [Puppet Enterprise](http://puppetlabs.com/puppet/puppet-enterprise/) - The commercial version
* [The Community](http://puppetlabs.com/community/overview/) - Active and vibrant
* [Puppet Documentation](http://docs.puppetlabs.com/) - Main and Official reference
* Puppet Modules on: [Module Forge](http://forge.puppetlabs.com/) and [GitHub](https://github.com/search?q=puppet)

**Installation**

* Take one virtual machine as master and any number of other VMs as agents of the master machine
* Edit

# vi /etc/hosts

* Add: ipaddress hostname //of master

ipaddress hostname //of agents

in master as well as all agent servers

* Take rpm package of puppet repository:

puppetlabs-release-e17.noarch

//By using this repo, puppet 3.8 will install using yum

Install rpm by using:

# rpm –ivh puppetlabs-release-e17.noarch

* Check if the puppet repo is created by going to:

# cd /etc/yum.repos.d/

# vi puppetlabs.repo

**Standalone puppet**

* If in master only the code need to be executed then give the following command:

# Puppet apply /etc/puppet/manifest/site.pp

**In Master**

* Install puppet-server using yum:

# Yum install puppet-server

* Go to puppet.conf and add dns names and certname:

# vi /etc/puppet/puppet.conf

add:

dns\_alt\_names = master,master.interns.hpe //master hostname

vcertname = master.interns.hpe

* Give :

# puppet master –verbose –no-daemonize

* Once you get “Notice:Starting puppet master version<VERSION>

Press ctrl+c (to kill the process)

**In Slave**

* Install puppet using yum:

# Yum install puppet

* Go to puppet.conf and add dns names and certname:

# vi /etc/puppet/puppet.conf

add:

[agent]

server = master.interns.hpe // master hostname

**In Master**

* Start the puppetmaster service

# service puppetmaster start

**In Slave**

* Start and enable the puppet service using systemctl so that when the server goes up, the puppet automatically starts

# systemctl start puppet.service

# systemctl enable puppet.service

**In Master**

* To list the certificates which is sent by the agents to the master to be signed:

# puppet cert list

# puppet cert sign “client.interns.hpe” // client hostname whose certificate need to be signed

**In Slave**

* To execute the puppet code:

# Puppet agent –t

**Troubleshooting**

* Check if firewall is running, if yes stop it:

# service firewalld status

# service firewalld stop

* While requesting for certificate by the clients, if the master not getting the certificate then
* clean the certficates by giving:

# puppet cert clean <certname>

then again try

* if error still present then last option is to delete the **/var/lib/puppet/ssl** files in both master and agent and try again
* We can check the syntax of the codes by giving:

puppet parser validate {filename}

**Puppet language**

* A **Declarative** Domain Specific Language (DSL)
* It defines **STATES** (Not procedures)
* Puppet code is written in **manifests** (files with **.pp** extension)
* In the code we declare **resources** that affect elements of the system (files, packages, services etc)
* Resources are grouped in **classes** which may expose parameters that affect their behavior.
* Classes and configuration files are organized in **modules**.

**Nodes configuration**

* When clients connect, the Puppet Master generates a **catalog** with the list of of the resources that clients have to apply locally.
* The Puppet Master has to classify nodes and define for each of them:
* The **classes** to include
* The **parameters** to pass
* The Puppet **environment** to use
* The **catalog** is generated by the Master according to the logic of our Puppet code and data.
* In our code we can define our **variables** and use other ones that may come from different sources:
* **facts** generated directly by the client
* **parameters** obtained from node's classification
* Puppet **internal** variables

**Resource type**

* Resource Types are single **units of configuration** composed by:
* A **type** (package, service, file, user, mount, exec ...)
* A **title** (how is called and referred)
* Zero or more **arguments**

type { 'title':

  argument  => value,

  other\_arg => value,

}

Example for a **file** resource type:

file { 'motd':

  path    => '/etc/motd',

  content => 'Tomorrow is another day',

}

Installation of OpenSSH package

package { 'openssh':

  ensure => present,

}

Start of httpd service

service { 'httpd':

  ensure => running,

  enable => true,

}

**Class definition**

class mysql (

  root\_password => 'default\_value',

  port          => '3306',

) {

  package { 'mysql-server':

    ensure => present,

  }

  service { 'mysql':

    ensure    => running,

  }

  [...]

}

**Useful paths**

**/var/log/puppet** contains logs (but also on normal syslog files, with facility daemon), both for agents and master

**/var/lib/puppet** contains Puppet operational data (catalog, certs, backup of files...)

**/var/lib/puppet/ssl** contains SSL certificate

**/var/lib/puppet/clientbucket** contains backup copies of the files changed by Puppet

**/etc/puppet/manifests/site.pp** (On Master) The first manifest that the master parses when a client connects in order to produce the configuration to apply to it (Default on Puppet < 3.6 where are used config-file environments)

**/etc/puppet/environments/production/manifests/site.pp** (On Master) The first manifest that the master parses when using directory environments (recommended from Puppet 3.6 and default on Puppt >= 4)

**/etc/puppet/modules** and **/usr/share/puppet/modules** (On Master) The default directories where modules are searched

**/etc/puppet/environments/production/modules** (On Master) An extra place where modules are looked for when using directory environments

**Class inheritance**

* In Puppet classes are just containers of resources and have nothing to do with OOP classes. Therefore the meaning of class inheritance is somehow limited to few specific cases.
* When using class inheritance, the main class ('puppet' in the sample below) is always evaluated first and all the variables and resource defaults it sets are available in the scope of the child class ('puppet::server').
* Moreover the child class can override the arguments of a resource defined in the main class. Note the syntax used when referring to the existing resource File['/etc/puppet/puppet.conf']:

class puppet {

  file { '/etc/puppet/puppet.conf':

    content => template('puppet/client/puppet.conf'),

  }

}

class puppet::server inherits puppet {

  File['/etc/puppet/puppet.conf'] {

    content => template('puppet/server/puppet.conf'),

  }

}

**Metaparameters**

Metaparameters are parameters available to any resource type, they can be used for different purposes:

Manage dependencies (**before**, **require**, **subscribe**, **notify**, **stage**)

Manage resources' application policies (**audit**, **noop**, **schedule**, **loglevel**)

Add information to a resource (**alias**, **tag**)

**Notify**

* This resource will be applied *before* the notified resource(s).
* If Puppet makes changes to this resource, it will cause all of the notified resources to *refresh.* (Refresh behavior varies by resource type: services will restart, mounts will unmount and re-mount, etc. Not all types can refresh.)

**Require**

* One or more resources that this resource depends on, expressed as [resource references](https://docs.puppetlabs.com/puppet/latest/reference/lang_data_resource_reference.html). Multiple resources can be specified as an array of references. When this attribute is present:
* The required resource(s) will be applied **before** this resource.

**Subscribe**

* One or more resources that this resource depends on, expressed as [resource references](https://docs.puppetlabs.com/puppet/latest/reference/lang_data_resource_reference.html). Multiple resources can be specified as an array of references. When this attribute is present:
* The subscribed resource(s) will be applied before this resource.
* If Puppet makes changes to any of the subscribed resources, it will cause this resource to refresh.(Refresh behavior varies by resource  type: services will restart, mounts will unmount and re-mount, etc. Not all types can refresh.)

**Example of before, notify**

package { 'exim':

  before => File['exim.conf'],

}

file { 'exim.conf':

  notify => Service['exim'],

}

service { 'exim':

}

Above code is equivalent to

Package['exim'] -> File['exim.conf'] ~> Service['exim']

**Example of require, subscribe**

package { 'exim':

}

file { 'exim.conf':

  require => Package['exim'],

}

service { 'exim':

  subscribe => File['exim.conf'],

}

Above code is equivalent to

Service['exim'] <~ File['exim.conf'] <- Package['exim']

**INTRODUCTION TO HIERA**

* Hiera is the **key/value lookup tool** of reference where to store Puppet user data.
* It provides an highly customizable way to lookup for parameters values based on a custom hierarchy using many different backends for data storage.
* It provides a command line tool hiera that we can use to interrogate direclty the Hiera data and functions to be used inside Puppet manifests: hiera() , hiera\_array() , hiera\_hash() , hiera\_include()
* Hiera is installed by default with Puppet version 3 and is available as separated download on earlier version ([Installation instructions](http://docs.puppetlabs.com/hiera/1/installing.html)).
* We need Hiera only on the PuppetMaster (or on any node, if we have a masterless setup).

**Hiera configuration: hiera.yaml**

* Hiera's configuration file is in yaml format and is called **hiera.yaml** here we define the hierarchy we want to use and the backends where data is placed, with backend specific settings.
* Hiera's configuration file path is different according to how it's invoked:

From the Command Line and Ruby code

* Default path: /etc/hiera.yaml

### From Puppet

* Default path for Puppet OpenSource: /etc/puppet/hiera.yaml
* Default path for Puppet Enterprise: /etc/puppetlabs/puppet/hiera.yaml
* It's good practice the symlink these alternative configuration files in order to avoid inconsistencie when using Hiera from the shell line or within Puppet manifests:

ln -s /etc/hiera.yaml /etc/puppet/hiera.yaml

By default Hiera does not provide a configuration file. The default settings are equivalent to this:

---

:backends: yaml

:yaml:

  :datadir: /etc/puppet/hieradata

:hierarchy: common

:logger: console

**Hiera backends**

* One powerful feature of Hiera is that the actual key-value data can be retrieved from different backends.
* With the :backends global configuration we define which backends to use, then, for each used backend we can specify backend specific settings.

### **Build it backends:**

* **yaml** - Data is stored in yaml files (in the :datadir directory)
* **json** - Data is stored in json files (in the :datadir directory)
* **puppet** - Data is defined in Puppet (in the :datasouce class)

**Hierarchy**

* With the :hierarchy global setting we can define a string or an array of data sources which are checked in order, from top to bottom.
* When the same key is present on different data sources by default is chosen the top one. We can override this setting with the :merge\_behavior global configuration. Check [this page](http://docs.puppetlabs.com/hiera/1/lookup_types.html#deep-merging-in-hiera--120) for details.
* In hierarchies we can interpolate variables with the %{} notation (variables interpolation is possible also in other parts of hiera.yaml and in the same data sources).

This is an example Hierarchy:

---

    :hierarchy:

      - "nodes/%{::clientcert}"

      - "roles/%{::role}"

      - "%{::osfamily}"

      - "%{::environment}"

      - common

**Using hiera in puppet**

The data stored in Hiera can be retrieved by the PuppetMaster while compiling the catalog using the hiera() function.

In our manifests we can have something like this:

$my\_dns\_servers = hiera("dns\_servers")

which assigns to the variable $my\_dns\_servers (can have any name) the top value retrieved by Hiera for the key dns\_servers