Hiera:

Is a key value look up too used in puppet.

1) make module for ssh in the following location /etc/puppet/modules/sshdconfig/manifests/

vim init.pp

|  |
| --- |
| class sshdconfig ( $serviceName = hiera("sshservicename")){  file { "/etc/ssh/sshd\_config":  owner => 'root',  group => 'root',  mode => '0644',  source => "puppet:///modules/sshdconfig/sshd\_config",  notify => Service[$serviceName],  }  service { $serviceName:  ensure => 'running',  enable => 'true',  }  } |

2) make directory hieradata and create three different ".yaml" file in it

mkdir /etc/puppet/hieradata

cd hieradata and vim Redhat.yaml Debian.yaml common.yaml

Redhat.yaml

|  |  |  |
| --- | --- | --- |
| Redhat.yaml | Debian.yaml | Common.yaml |
| ---  sshservicename: sshd | ---  sshservicename: ssh | ---  sshservicename: zoo |

3) make hiera.yaml file in /etc/puppet/

|  |
| --- |
| hiera.yaml |
| ---  :backends:  - yaml  :hierarchy:  - %{osfamily}  - common  :yaml:  :datadir: /etc/puppet/hieradata/ |

4) To check and confirm that hiera is returning the correct data depending upon the osfamily value provided. This can be tested from the following command

hiera -c hiera.yaml sshservicename osfamily=Redhat # this command should give output whatever the sshservicename of the RedHat ( which is in the RedHat.yam file)

Puppet is an infrastructure automation tool with its own programming syntax, used to automate most of the task's done by a system administrator. If you are new to puppet, I would recommend reading the below articles before you go ahead.

**Read:** [What is Puppet](http://www.slashroot.in/puppet-tutorial-introduction-puppet-configuration-management-tool)

**Read:**[How does Puppet Work](http://www.slashroot.in/puppet-tutorial-how-does-puppet-work)

**Read:**[Installing and configuring Puppet Master](http://www.slashroot.in/puppet-tutorial-configuring-puppet-master)

**Read:**[Getting Started With Puppet Manifests](http://www.slashroot.in/puppet-tutorial-getting-started-puppet-manifests)

To understand why it is important to separate data from the code in puppet, let's create a simple module called sshconfig. This module will deploy ssh configuration file and also ensure that the ssh service is running.

Let me show you my sshconfig module directory structure.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3 | root@puppet:/etc/puppet/modules# cd sshdconfig/  root@puppet:/etc/puppet/modules/sshdconfig# ls  manifests  README.md  templates |

As you know all configuration and puppet module language is defined inside the .pp files in *<module-directory>/manifests/\*.pp*. Let's see what's inside my /etc/puppet/modules/sshdconfig/manifests/init.pp file.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | root@puppet:/etc/puppet/modules/sshdconfig/manifests# cat init.pp  class sshdconfig {      case $::osfamily {          Debian: {              $serviceName = 'ssh'          }          RedHat: {              $serviceName = 'sshd'          }      }        file { "/etc/ssh/sshd\_config":          owner   => 'root',          group   => 'root',          mode    => '0644',          content => template("$module\_name/sshd\_config.erb"),          notify  => Service[$serviceName],      }        service { $serviceName:          ensure => 'running',          enable => 'true',      }  }  root@puppet:/etc/puppet/modules/sshdconfig/manifests# |

The module only does two things as mentioned before.

* It populates the content of ssh configuration file (from a template)
* It ensures that the service is running or not.

Although it only does this two things, we have a case statement in the beginning of the init.pp file. This case statement does modify the value of the servicename variable depending upon on the osfamily reported by the puppet client.

*If you are new to puppet -- You can define case statements on the puppet master server inside the manifests, that will provide different sets of data based on the facter variable. Facter is nothing but a tool installed on all puppet agent's (puppet client's), that collects all inventory details of that server. These details contains information like Ip address, Hostname, Operating system, Osfamily, and much more...*

An example output of facter with the variable's we used is shown below.

**Note**: The below command was run on one of my puppet client. The full output of facter command is too long to show here, so i did a grep for osfamily and operating system variables.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4 | root@puppet:~# facter | egrep -w 'osfamily|operatingsystem'  operatingsystem => Ubuntu  osfamily => Debian  root@puppet:~# |

In the previously shown sshdconfig module, we only had two case statement's (one for redhat os, and the other for ubuntu os.). Imagine if you have 7 to 8 different types of Linux distribution's running in your environment. In that case, your sshdconfig module's init.pp file should contain a case statement for each of those operating system's. Now the code for that module will become unnecessarily lengthy because we are including variable data inside the code(which is not a good practice)

*The situation will become even worse if you have a module, and that module has to give different configuration values on different hosts. Take an example of your iptable firewall module. You will have different set of rules in each host (some will have port 25 open, some will have 80 open, some hosts should allow ssh access to a certain ip address, but some should deny etc etc.). In that case hardcoding values as we did is not going to help, and is really a bad idea.*

Another disadvantage of hardcoding data inside a puppet module is that it becomes useless to other's. Which means it is not portable. Because the module will contain values which are meaningful to your environment and architecture, and will be useless in one of your new project (and you cannot even share that module which other's).

A good idea is to separate the data from the puppet module manifests, so that the module code is untouched, and only data values are modified depending upon the hostname, osfamily, environment, etc etc.

To solve this problem we can use something called as Hiera with puppet.

What is hiera?

*Hiera is nothing, but a key value look up tool, which can be ordered and organized nicely without meddling with the actual code. Just give hiera the data that your modules need, and you are ready to go.*

Let me tell that again.."Hiera makes your data separate from the module's, so that the module code remains untouched. This helps your module's to be reusable and clean, and save you from repetition". Before you go ahead, read the below line very carefully.

***Hiera requires requires Puppet 2.7.x or later***

Your configuration data inside hiera can be in two formats(mentioned below.)

* YAML
* JSON

*YAML stands for "YAML Ain't Markup Language". YAML is used for hierarchical data representation, where a user can specify his configuration data that a program can call and use with ease. All YAML configuration files will end with a .yaml extension. YAML follows a strict syntax. White space and tab's inside YAML files have special meaning and must be used with care.*

*JSON stands for JavaScript Object Notation. It is similar to JSON but with different syntax. It is also a key value store used for programs to fetch variable data from.*

Before we go ahead and install hiera with puppet, please make a note of the below things.

1. Latest versions of puppet ship with hiera installed (well puppet 3 and later to be more specific), so no need to install hiera separately.
2. Hiera is not supported with puppet versions earlier to 2.7 (although it will work, but not supported officially)
3. If you are using hiera with puppet version 2.7, you will need to install an additional package on the puppet master server. This package name is *hiera-puppet*
4. Hiera requires Ruby 1.8.5 or later versions.

Using Hiera with Puppet 3

As discussed earlier in the above points, if you are using hiera with puppet 3, then you do not need to install anything to get hiera working. But still keep the below points about its configuration in mind.

* Puppet will look for hiera configuration file in **/etc/puppet** directory. Hence the config file path becomes **/etc/puppet/hiera.yaml.**
* Also you can easily change this path, if you want to keep the hiera configuration file separate by adding hiera\_config parameter inside puppet.conf file.

Using Hiera with puppet 2.7

If you are using puppet version 2.7, then you need to install two things. One is the puppet hiera package and the second is the hiera gem. Let's see how to do this.

On ubuntu OS, you can easily install hiera-puppet package using the default package manager apt-get as shown below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1 | root@puppet:~# apt-get install hiera-puppet |

Similarly on a red-hat system, you can install hiera-puppet using the yum package manager, as shown below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1 | root@puppet:~# yum install hiera-puppet |

Please note the fact that, to install the above package, you need to have puppet lab's package repositories to be enabled(do not do anything if you already have a running puppet 2.7 master on your server). You can find a excellent guide to do this on the official puppet lab's tutorial.

**Read:**[Enabling Puppet Package Repositories](http://docs.puppetlabs.com/guides/puppetlabs_package_repositories.html)

Now you need to install hiera gem. This can be done, as shown below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | root@puppet:~# gem install hiera  Fetching: hiera-1.3.2.gem (100%)  Successfully installed hiera-1.3.2  1 gem installed  Installing ri documentation for hiera-1.3.2...  Installing RDoc documentation for hiera-1.3.2...  root@puppet:~# |

Now let's get inside the configuration part of hiera with puppet.

*Puppet will ask hiera, for a configuration data value. And its hiera's job to return the correct value depending upon the environment and hierarchy.*

**Step 1:**Create a configuration file called *hiera.yaml* inside /etc/puppet/ directory. Please note the fact that this is the default location where puppet will look for hiera config file. You can change this with hiera\_config setting inside puppet.conf. The file should look something like the below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | :hierarchy:      -"%{::osfamily}"      - common  :backends:      - yaml  :yaml:      :datadir: '/etc/puppet/hieradata/' |

You can also use an empty hiera.yaml file inside /etc/puppet for hiera to work. But if the file is empty, then it will take the default hiera configurations. The default settings, if you create an empty file will be something like the below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ---  :backends: yaml  :yaml:    :datadir: /var/lib/hiera  :hierarchy: common  :logger: console |

*The hiera.yaml file may contain any of the backends, yaml, hierarchy, logger settings. If you miss any of these, then the above default value will be considered.*

All configuration settings that you see inside hiera.yaml file is considered and looked up in the order you define. That is, if your **:backends:** settings has got two values (- yaml, and - json), then hiera will first search all yaml files first, and then search all .json files. Hierarchy is core concept behind hiera in puppet.

**:hierarchy**setting in hiera can contain a string or an array of strings as values. And all these strings you provide will be considered as a data source for hiera to lookup. Strings can be static or dynamic.

*A dynamic data source in hiera is the one that contains %{your data source variable}. An example of a dynamic data source is the one that we gave in our hiera.yaml file for osfamily ie: %{::osfamily}. A normal string like common in our example is static data source.*

**:datadir**setting in hiera can contain a string value, which defines the location of the data source files. These files will contain user defines values(the files inside this directory will be searched, when puppet asks for a configuration data in a module).

*Only keep two things in mind as of now. That is,****:hierarchy****setting is used to define your own hierarchy of data source(which files to look first and which to look last). And****:datadir****setting is used to specify the location of these data source files(basically you will be creating files inside that location with the name/variable you provided in****:hierarchy****setting, with .yaml extension if you are using backends as****:yaml****).*

As we saw earlier, hiera can have two backend types(JSON or YAML), we will be discussing only YAML back ends in this tutorial, as its quite simple. JSON format appears a little messy to me, being that said, you can achieve it with JSON with little effort by yourself. As i mentioned before, everything is hierarchy in hiera.yaml. Let's see what that means with an example.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | ---  :backends:    - yaml    - json  :hierarchy:    - first    - second    - third |

Please note the fact that i have omitted other settings from the above hiera.yaml file, just to make it simpler to only understand the hierarchy based lookup part.

If you have the above setting in your hiera.yaml, then hiera will start its lookup in the following order.

As the first backend is YAML, it will first look for all .yaml files in the data source folder(provided by the **:datadir** settings). It will first look for data in file named **first.yaml**, then it will move on to **second.yaml**, then it will finally look for data inside **third.yaml**. Please make a note of the fact that, hierarchy to hiera is the order in which you have given string values in **:hierarchy** setting.

Now once hiera has completed looking for all **.yaml** files, it will start looking for all json files. This is because we have given json as the second backend in the **:backends** setting. JSON files will also be searched in the same order as YAML files. First it will look for data inside **first.json**, then **second.json**, and then finally **third.json**.

Now as we have some basics about hiera at hand. Let's start configuring it for our sshdconfig module. First before using it inside our sshdconfig module, we will configure hiera data sources for different osfamily, then do a test lookup for data. And once we are sure, we will apply it inside our module.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | root@puppetmaster:/etc/puppet# cat hiera.yaml  ---  :backends: - yaml    :hierarchy:    - %{osfamily}    - common    :yaml:    :datadir: /etc/puppet/hieradata/ |

You can see that the **:hierarchy**setting in the above shown hiera.yaml file, will first look for a dynamic data source file (which means it will look for a variable called osfamily inside the datadir). Then it will look for data inside a file called as common.yaml.

Now we will create two different files based on osfamily dynamic hiera data source. These files will contain data which are specific to Debian OS family, and RedHat OS family. Let's see the contents of these files.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4 | root@puppetmaster:~# cat /etc/puppet/hieradata/Debian.yaml  ---  sshservicename: ssh  root@puppetmaster:~# |

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3 | root@puppetmaster:~# cat /etc/puppet/hieradata/RedHat.yaml  ---  sshservicename: sshd |

Now let's check, and confirm that hiera is returning the correct data depending upon the osfamily value provided. This can be tested with hiera command line tool as shown below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4 | root@puppetmaster:/etc/puppet# hiera -c hiera.yaml sshservicename osfamily=RedHat  sshd  root@puppetmaster:/etc/puppet# hiera -c hiera.yaml sshservicename osfamily=Debian  ssh |

*hiera* command line tool can be used to test data returned. In the above command, i have provided two different values of osfamily each time. And it searched in the exact data source file to fetch the value. Hence now we can use it in puppet. Let's use this same hiera configuration inside our simple sshdconfig module as shown below(our previous sshconfig init.pp file with hiera data is shown below.)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | root@puppetmaster:/etc/puppet/modules/sshdconfig/manifests# cat init.pp  class sshdconfig ( $serviceName = hiera("sshservicename") ){        file { "/etc/ssh/sshd\_config":          owner   => 'root',          group   => 'root',          mode    => '0644',          source => "puppet:///modules/sshdconfig/sshd\_config",          notify  => Service[$serviceName],      }        service { $serviceName:          ensure => 'running',          enable => 'true',      }  } |

In the above shown init.pp file, we have used a function called *hiera* which is part of puppet 3 package, and hiera-puppet package in 2.7 version.

*When puppet asks for the hiera data, hiera provides back the data it finds through lookup. The lookup is performed with the variables provided by puppet to hiera. In our case its a facter variable named osfamily, as shown earlier. Using the above shown hiera function inside puppet modules, we can save many lines of unnecessary site specific data as well as organize it in much better way.*

You must be thinking that a key can be present in multiple data sources. That is, why does hiera stop at the first occurrence of a match? Actually it depends on the function that we use with puppet to fetch data from hiera.

*The hiera function that we used in the above sshdconfig module's init.pp file(hiera("sshservicename")), will stop its lookup at the first occurrence of the key and return the data. That is, it will start the lookup, in the order you specified in****:hierarchy****setting, and will return the first value it finds. This is fine, where you need one specific and unique data for a value inside your module. But there are cases, where you require all the occurrences to be returned. In other words, we need hiera to return all the data it finds in all the data source files. For such use cases we will use another puppet function called hiera\_include. hiera\_include is nothing but an array merge lookup function.*

Array merge lookup function will assemble all the values it finds for a key and will return it as a single large array.

I recommend my readers to read the excellent article from the official puppet documentation below. The below article will take you through deploying an NTP module with different servers for different hosts using hiera.

**Read:**[Hiera complete example from Puppetlabs](http://docs.puppetlabs.com/hiera/1/complete_example.html" \t "_blank)

Now let's try to do a little bit more complex example of using hiera with puppet. In this example, we will include classes to nodes using hiera config files. The complete list of classes that needs to be applied to a node will come from hiera.

So let's first create a hiera.yaml file, with a new hierarchy. And then create a node definition to fetch classes from hiera.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | root@puppetmaster:~# cat /etc/puppet/hiera.yaml  ---  :backends: - yaml    :hierarchy:    - %{fqdn}    - %{osfamily}    - common    :yaml:    :datadir: /etc/puppet/hieradata/ |

The above shown hierarchy setting will first look for any fqdn of your puppet clients with .yaml extension inside the data directory. Then it will look for osfamily (similar to our previous example, RedHat.yaml & Debian.yaml etc.). And then finally it will look for a common.yaml file which will contain data that will be applicable to all nodes.

so let's create three files inside /etc/puppet/hieradata folder. These files and its contents are shown below.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | root@puppetmaster:~# cat /etc/puppet/hieradata/common.yaml  ---  classes:    - security    - firewall  root@puppetmaster:~# cat /etc/puppet/hieradata/Debian.yaml  ---  classes:    - apt  root@puppetmaster:~# cat /etc/puppet/hieradata/puppetclient.example.com  ---  classes:    - mysql    - apache |

My puppet client (puppetclient.example.com, with the fqdn.yaml above) is a Debian os. Now let's see what all classes will get applied to that node if we call it from our site.pp file inside a node.

You can call classes from hiera using the below shown method, while defining a node.

[?](http://www.slashroot.in/puppet-hiera-tutorial-example-configuration)

|  |  |
| --- | --- |
| 1  2  3 | root@puppetmaster:/etc/puppet/manifests# cat site.pp  hiera\_include('classes', '')  node 'puppetclient.example.com' {} |

Wow isn't that too short a site.pp file. The *hiera\_include* function in the beginning of the site.pp file, will pull all the classes applicable to that all nodes defined inside site.pp

*As our node puppetclient.example.com is a Debian system, it will have all classes defined inside Debian.yaml, as its hostname and fqdn facter values are puppetclient.example.com, it will have all classes defined inside puppetclient.example.com.yaml, and also it will have all classes defined inside common.yaml(as its a static data source, applicable to all nodes). If our puppetclient.example.com was a RedHat system, then it will have classes defined in RedHat.yaml(if there is one), common.yaml, and puppetclient.example.com.yaml file.*