DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Puppet Forge, Hiera, and Best Practices

Learning Objectives

By the end of this lesson, you will be able to:

- Install and uninstall Puppet modules from Puppet Forge
- Publish custom modules to Puppet Forge
- Configure Puppet Hiera
- Utilize Puppet Hiera to store and look up data
- Configure Puppet Facter

Introduction to Puppet Forge

What Is Puppet Forge?

Puppet Forge is the official online repository hosted by Puppet Labs. It contains more than 6000 Puppet modules written by Puppet developers and open source community. Users can download, install, publish, and modify the existing Puppet Forge modules.

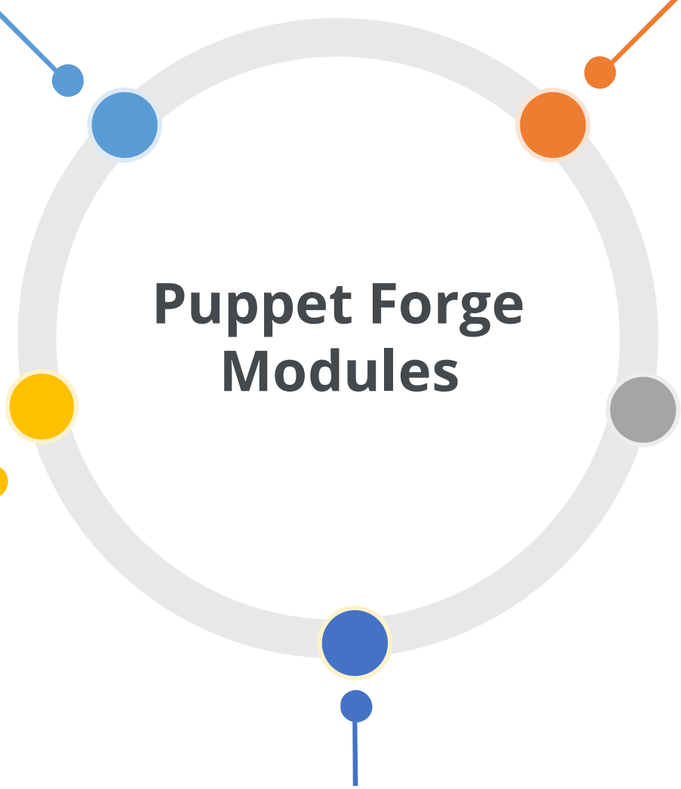
Types of Modules in Puppet Forge

Supported

Core modules that are rigorously tested with Puppet Enterprise and supported by Puppet, Inc

Approved

Modules that are actively maintained and meet the Puppet Labs' standards of being reliable and up to date



Partner

Modules that are rigorously tested with Puppet Enterprise and supported by a partner organization

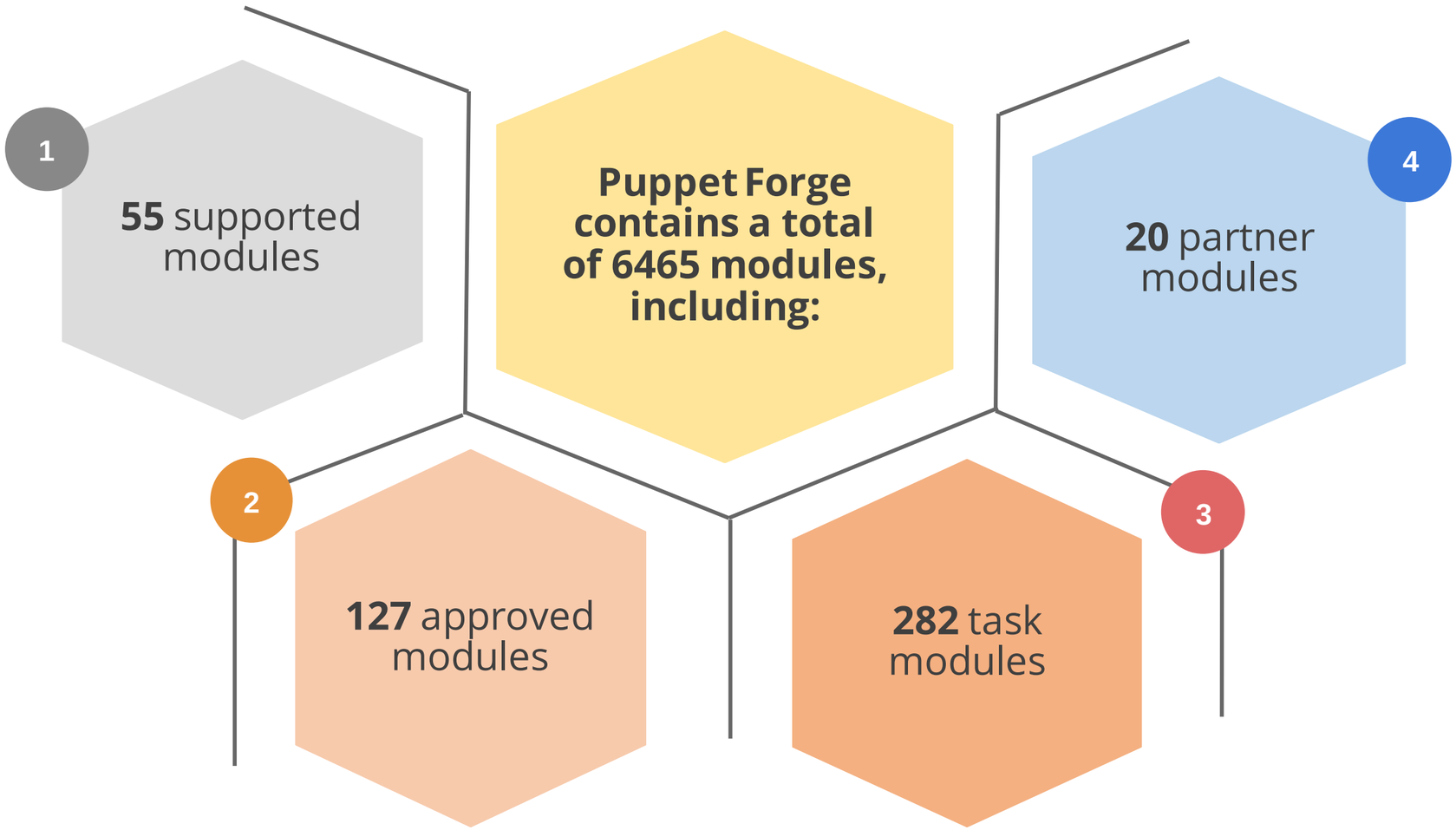
Tasks

Modules which contain tasks that act outside of a desired state managed by Puppet

PDK

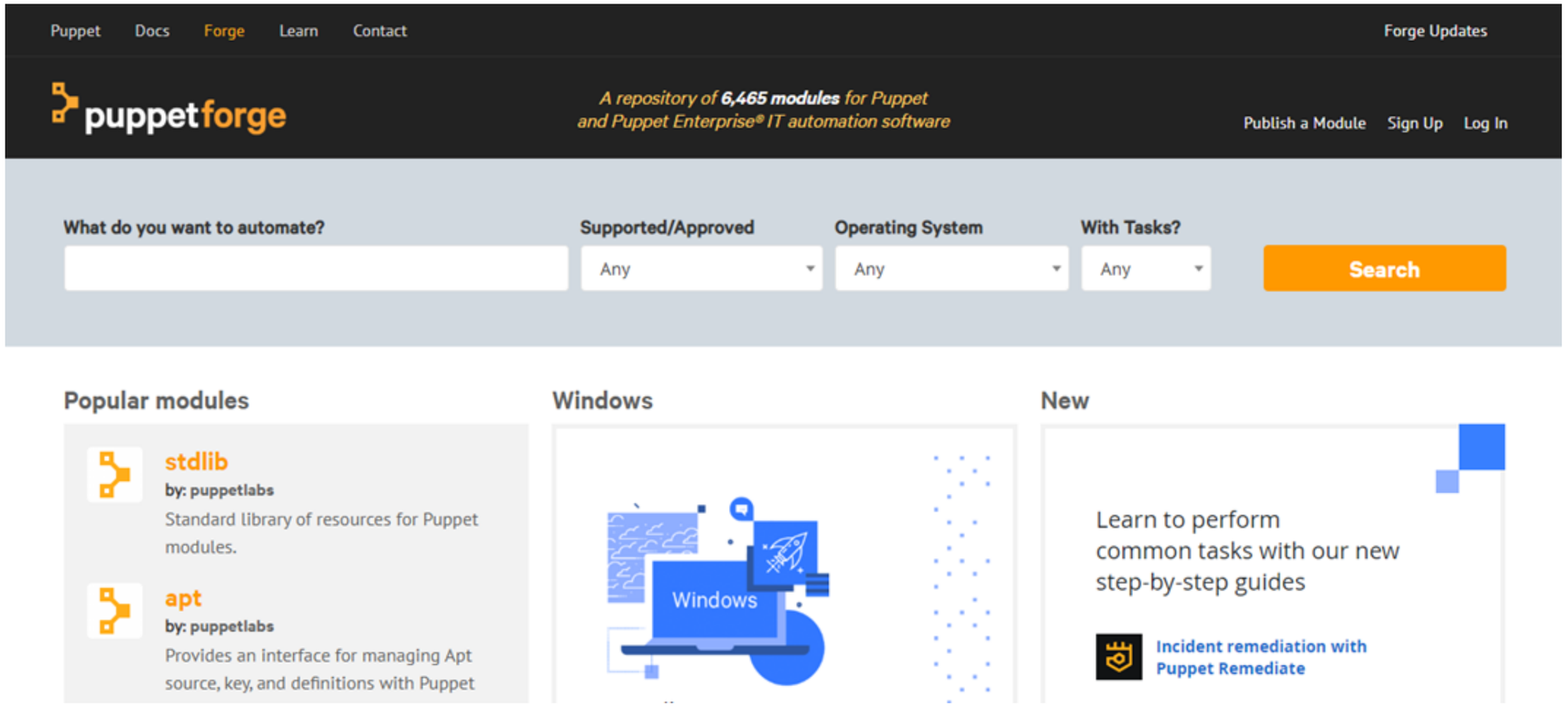
Modules that are compatible with Puppet Development Kit for validation and testing

Puppet Forge Facts



Puppet Forge Walk-through

Puppet Forge repository can be accessed by using the URL forge.puppet.com. Modules can be searched using module name, module category, or module use case.



Assisted Practice

Installing Puppet Modules from Puppet Forge

Problem Statement:

Install Puppet modules from Puppet Forge using the official Forge website and a command line.

Assisted Practice: Guidelines

Steps to perform:

1. Open the browser and navigate to forge.puppet.com
2. Search for the module you want to install using the search bar
3. Download the tarball of the module
4. Unzip the downloaded file
5. Open the command line, search for a module in forge and download it

Assisted Practice

Publishing Puppet Modules to Puppet Forge

Problem Statement:

Upload and publish a custom Puppet module in Puppet Forge.

Assisted Practice: Guidelines

Steps to perform:

1. Create a Forge account
2. Prepare the custom module
3. Add metadata.json file in the module
4. Build a module package using pdk
5. Upload the module package to the Puppet Forge

Assisted Practice

Deleting a Module from Puppet Forge

Problem Statement:

Delete a module release from Puppet Forge using the Puppet Forge interface.

Assisted Practice: Guidelines

Steps to perform:

1. Login to your Puppet Forge account
2. Navigate to your modules section
3. Select the module release from the dropdown
4. Click on delete
5. On confirmation page, click on submit

Introduction to Puppet Hiera

What Is Puppet Hiera?

Hiera is a built-in key-value configuration data lookup system for Puppet.

It is used for separating and storing data from Puppet code.

It follows "default, with overrides" patterns, so the users can specify common data and override it in situations where default pattern is not applicable.

Hiera uses Puppet facts to specify data sources.

Working of Hiera

Puppet Hiera is used to perform the following tasks:

1. Store the configuration data in key-value pairs
2. Look up the data that a module needs for a target node during catalog compilation

Hiera performs the above tasks using:

1. Automatic parameter lookup for classes included in the catalog
2. Explicit lookup calls

Why Use Puppet Hiera?

Holds all the data that has to be dynamically placed in a module

Store usernames, Passwords, DSN server details, and more

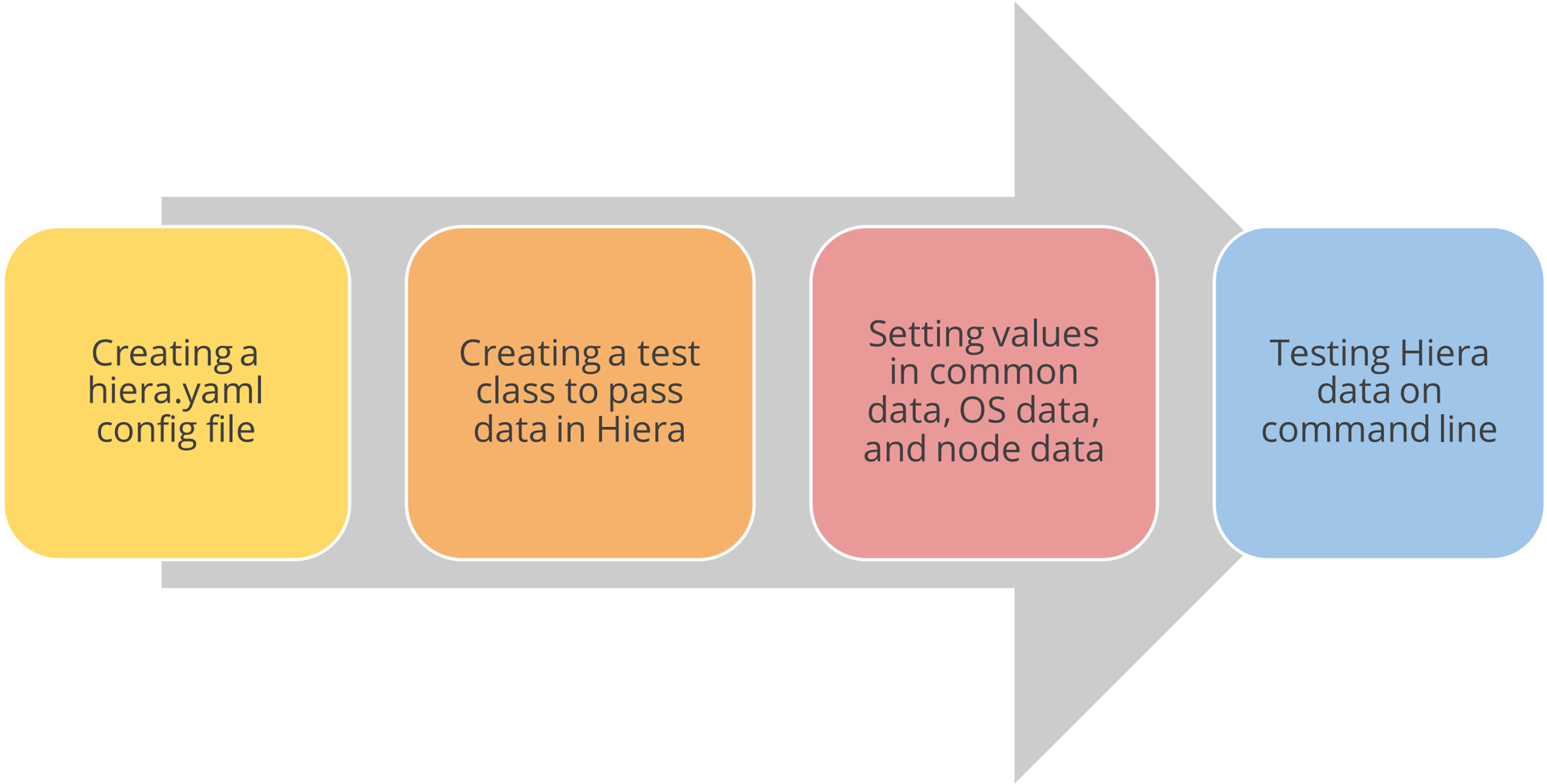
Encrypts the data stored in Hiera to maintain security

Is accessed from any master or agent system

Puppet Hiera Workflow Setup

Puppet Hiera Workflow Setup

Following are the basic and foundational steps required to be performed to set up Puppet Hiera:



Creating a hiera.yaml Config File

Each module or environment must have its own hiera.yaml file. For downloaded and installed modules, the user may need to touch up the existing hiera.yaml file to align the config file with their particular needs.

Screenshot of an ideal config file is given below:

```
---
:backends:
  - yaml
  - json
:yaml:
  :datadir:
/etc/puppetlabs/code/environments/%{::environment}/hieradata
:json
  :datadir:
/etc/puppetlabs/code/environments/%{::environment}/hieradata
:hierarchy:
  - "node/%{::fqdn}"
  - common
```

Components of Hiera Config File:

:backends	Represents all the backend program formats that Heira uses such as yaml, json, and Puppet class backends
:datadir	Represents the location where the Hiera data is stored and looked up from
:hierarchy	Represents the folder and file hierarchy inside the directory where the Hiera data is stored

Creating a Test Class

A test class will write the data it receives to a temporary file stored on the agent node while applying the catalog.

Following are the steps to create a test class:

1. Create a module named profile
2. Use Puppet Development Kit to create a class called hiera_test.pp in the profile module
3. Create a manifest that includes the class using the following syntax:

```
# site.pp
include profile::hiera_test
```

Creating a Test Class

4 Add the following code snippet in the hiera_test.pp file:

```
 /etc/puppetlabs/code/environments/production/modules/profile/manifests/hiera_test.pp
class profile::hiera_test (
  Boolean             $ssl,
  Boolean             $backups_enabled,
  Optional[String[1]] $site_alias = undef,
) {
  file { '/tmp/hiera_test.txt':
    ensure  => file,
    content => @("END"),
               Data from profile::hiera_test
               -----
               profile::hiera_test::ssl: ${ssl}
               profile::hiera_test::backups_enabled: ${backups_enabled}
               profile::hiera_test::site_alias: ${site_alias}
               |END
    owner   => root,
    mode    => '0644',
  }
}
```

Creating a Test Class

5 Set the following keys in the Hiera data:

Parameter	Hiera key
$ssl	profile::hiera_test::ssl
$backups_enabled	profile::hiera_test::backups_enabled
$site_alias	profile::hiera_test::site_alias

Setting Values in Common Data

Common data is the YAML backend data that is stored in YAML files.

YAML file can be stored in any of the following locations:

1. Main environment's directory
2. Data directory
3. File path specified by the hierarchy level

Setting Values in Common Data

To set the common data, open the YAML file and paste the following code snippet in it:

```
# /etc/puppetlabs/code/environments/production/data/common.yaml
---
profile::hiera_test::ssl: false
profile::hiera_test::backups_enabled: true
```

Setting Values in OS Data

Operating System (OS) data is created using different data files depending on the operating system of the agent node.

Following are the steps to set values in operating system data:

1. Locate the data file and replace **%{facts.os.family}** with **/etc/puppetlabs/code/environments/production/data/ + os/ + Darwin + .yaml**

2. Add the following code snippet in the file:

```
# /etc/puppetlabs/code/environments/production/data/nodes/jenkins-prod-
03.example.com.yaml
---
profile::hiera_test::ssl: true
profile::hiera_test::site_alias: ci.example.com
```

Setting Values in Node Data

Node data is set individually for every node and is used at the highest level of the hierarchy.

Following are the steps to set node data:

1. Locate the data file and replace **%{trusted.certname}** with the node name you're targeting
2. Add the following code snippet in the file:

```
# /etc/puppetlabs/code/environments/production/data/nodes/jenkins-prod-
03.example.com.yaml
---
profile::hiera_test::ssl: true
profile::hiera_test::site_alias: ci.example.com
```

Testing Hiera Data on Command Line

The Puppet lookup command is used for testing Hiera data on command line. It is a command line interface for Puppet's lookup function.

Syntax of the Puppet lookup command:

```
puppet lookup <KEY> --node <NAME> --environment <ENV> --explain
```

Examples:

1

```
# to look up key_name using node's facts
$ puppet lookup key_name
```

2

```
# to lookup key_name with agent
puppet lookup --node agent.local --default 0
key_name
```

3

```
# to look up key_name with agent.local's
facts
$ puppet lookup --node agent.local key_name
```

4

```
# to see an explanation of how the value for
key_name is found, using agent
puppet lookup --node agent.local --explain
key_name
```

Puppet LookUp Command Options

The Puppet look up command has the following options:

--help	Prints usage message
--explain	Prints the details of how the lookup was performed
--node	Specifies the data node
--facts	Specifies a JSON or YAML file that contains key-value mapping for lookups
--environment	Specifies in which environment to store the Hiera data
--merge	Overrides any merge behavior from data lookup options

Assisted Practice

Testing Hiera with Puppet Agent

Problem Statement:

Test Hiera using Puppet agent server

Assisted Practice: Guidelines

Steps to perform:

1. Add data in the Puppet agent manifest file
2. Verify the returned data from Puppet master
3. Use Puppet agent command to pull node information from Puppet agent to Puppet master

Hiera Hierarchies

Hiera Hierarchies

Hiera looks up data by following a hierarchy. Hierarchies are specified and configured in a hiera.yaml configuration file.

Steps to perform a good hierarchy:

Create a short hierarchy

Include both built-in as well as custom facts

Give each environment or target node its own hierarchy level

Hiera Configuration Layers

Hiera Configuration Layers

Hiera uses three independent layers of configuration and each layer has its own hierarchy.

1

Global

Global layer is at the highest level of the Hiera hierarchy. It is mostly used for temporary overrides in data values.

2

Environment

Environment layer is at the second level of the Hiera hierarchy. It is where most of the Hiera data hierarchy definitions occur.

3

Module

Module layer is at the third level of the Hiera hierarchy. The module layer sets default values and merges behavior for a module's class parameters.

Location of hiera.yaml Files

Location of hiera.yaml Files

There are several hiera.yaml files in a Puppet deployment. Puppet Hiera uses three layers of configurations and each layer has its own hiera.yaml file.

The configuration file locations for each configuration layer is given below:

Layer	Location
Global	$confdir/hiera.yaml
Environment	<ENVIRONMENT>/hiera.yaml
Module	<MODULE>/hiera.yaml

Assisted Practice

Looking up Data in the Puppet Hiera

Problem Statement:

Use Puppet lookup command to look up data in Puppet Hiera from command line.

Assisted Practice: Guidelines

Steps to perform:

1. Configure hiera.yaml file
2. Add data to hiera.yaml file
3. Use lookup command in command line to look up the added data in Puppet Hiera

Introduction to Puppet Facter

What Is Puppet Facter?

Puppet Facter is Puppet's cross-platform system profiling library. It is a stand-alone tool that holds all the environment-level variables. It is used to discover and report facts per node that are available in the Puppet manifest as variables.

Users can view all the facts and their values using the facter command as shown below:

```
facter
```

What Is Puppet Facter?

If the user wants to just view one variable's value, then the following command is used:

```
facter {variable-name}
```

Example:

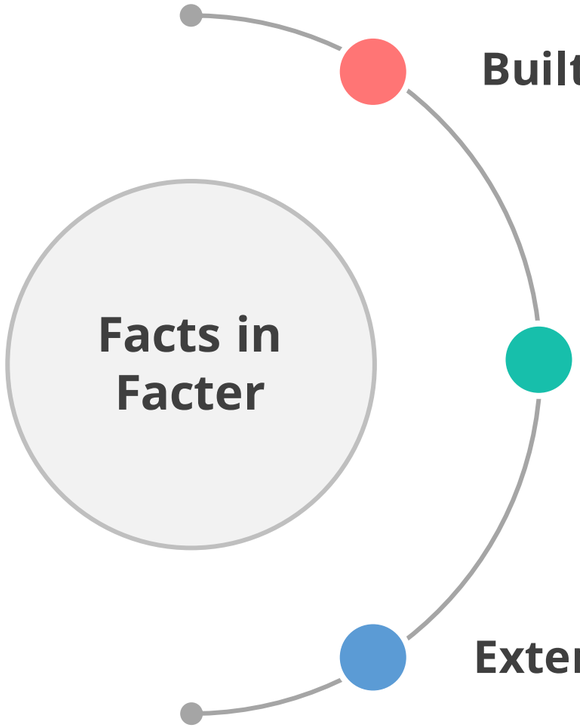
```
Facter virtual virtualbox
```

Parts of Facts in Facter

Most facts have the following two elements:

1	Call to Facter.add('fact_name')	This element determines the name of the fact.
2	Setcode statement	This element determines the fact's value.

Types of Facts in Facter



Built-in Facter facts: Shipped with Facter tool

Custom facts: Written by users in Puppet DSL and added to Facter

External facts: Written by users in other programming languages

Facts vs. Resolution

Facts

- A fact is a piece of information about a given agent node.
- Each fact must have at least one resolution.

Resolution

- A resolution is a way of obtaining the information about a given agent node.
- There can be multiple resolutions for a single fact.

Built-In Facter Facts

Built-In Facter Facts

Built-in facts are not available in the Puppet manifest but are shipped with Facter. Built-in facts include both legacy facts and newer structure facts.

Users can view all the facts and their values using the facter command as shown below:

```
puppet facts
```

Built-In Facter Facts

Examples of newly structured facts are given below:

Facts	Purpose
augeas	Returns information about augeas resources
cloud	Returns information about the cloud instance of the node if available
disks	Returns the information about disk devices attached to the node
dmi	Returns the information about system management information

Built-In Facter Facts

Examples of legacy facts are given below:

Facts	Purpose
architecture	Returns the information about the operating system's hardware architecture
blockdevices	Returns a comma-separated list of block devices
bios_version	Returns the version of the system BIOS
boardmanufacturer	Returns the system board manufacturer name

Custom Facter Facts

Writing Facts with Resolution

Resolution uses both the elements of a fact, that is fact name and setcode statement to obtain the information about an existing fact. These elements can also be used while writing custom facts.

Following syntax is used to write a fact with a single resolution:

```
Facter.add(:rubypath) do
  setcode 'which ruby'
end
```

NOTE

The above fact is not operating system dependent.

Writing Facts with Resolution

A single fact might have different or multiple resolutions depending on different operating systems.

Following syntax is used to write a fact with OS-specified resolution:

```
Facter.add(:rubypath) do
  setcode 'which ruby'
end

Facter.add(:rubypath) do
  confine :osfamily => "Windows"
  # Windows uses 'where' instead of 'which'
  setcode 'where ruby'
end
```

Writing Structured Facts

Structured facts are the facts that return values in hashes and arrays.

Following syntax is used to write a structured fact:

```
Facter.add(:interfaces_array) do
  setcode do
   interfaces = Facter.value(:interfaces)
   # the 'interfaces' fact returns a single comma-delimited
string, such as "lo0,eth0,eth1"
   # this splits the value into an array of interface names
   interfaces.split(',')
  end
end
```

Configuring Facts

Facts have a property called confine statement that can be configured to customize the way they are evaluated by Puppet Facter. The confine statement restricts the facts to only run on systems that match a given fact.

Following is an example of confine statement:

```
Facter.add(:powerstates) do
  confine :kernel => 'Linux'
  setcode do
    Facter::Core::Execution.execute('cat /sys/power/states')
  end
end
```

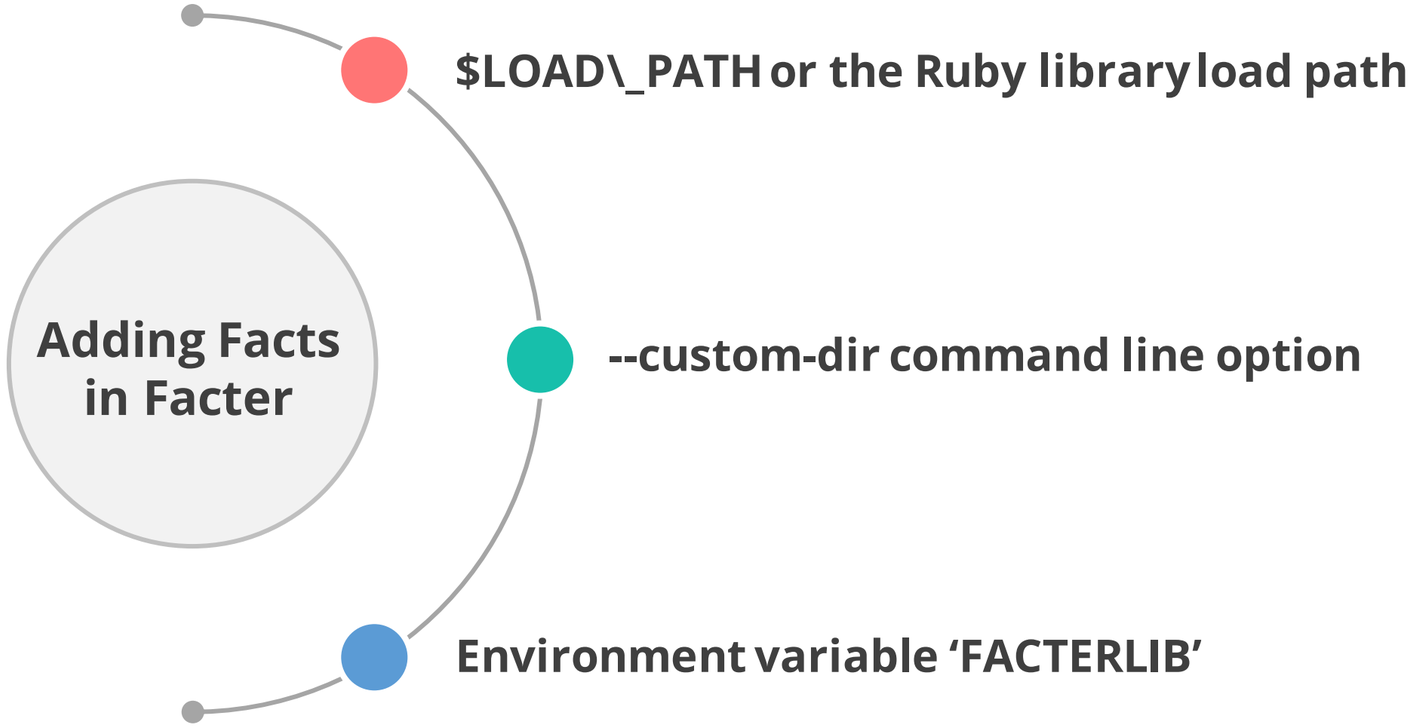
Facter.value attribute of confine statement is used to confine a structured fact like ['os']['family'], as shown in the following example:

```
confine Facter.value(:os)['family'] => 'Linux'
```

Adding Custom Facts to Facter

Adding Custom Facts to Facter

Puppet Facter offers the following three ways to add facts to Facter:



Using Ruby Load Path

Puppet Facter searches all directories in the Ruby $LOAD_PATH variable for sub-directories named factor and loads all ruby files in those sub-directories.

The screenshot of a load path variable directory structure is given below:

```
#~/lib/ruby
└── facter
    ├── rackspace.rb
    ├── system_load.rb
    └── users.rb
```

Using --custom-dir Command Line Option

The --custom-dir command specifies a single directory where the Puppet Facter searches for custom facts and adds them in the relevant directories.

Following example shows how to use --custom-dir command:

```
$ ls my_facts
system_load.rb
$ ls my_other_facts
users.rb
$ facter --custom-dir=./my_facts --custom-
dir=./my_other_facts system_load users
system_load => 0.25
users => thomas,pat
```

Using FACTERLIB Environment Variable

Puppet Facter checks the environment variable FACTORIB for a delimited set of directories and adds ruby files in those directories.

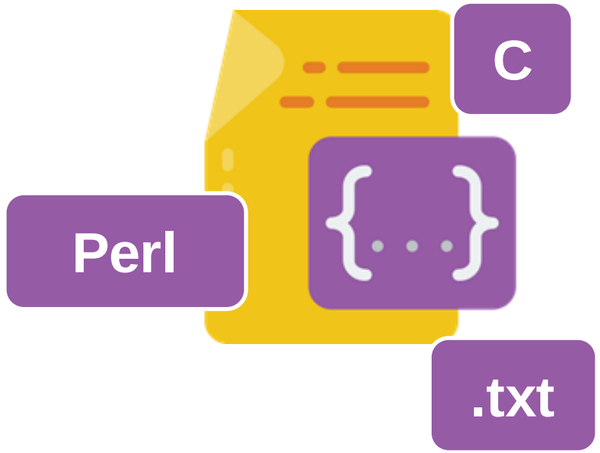
Following example shows how to use FACTERLIB environment variable:

```
$ ls my_facts
system_load.rb
$ ls my_other_facts
users.rb
$ export FACTERLIB="./my_facts:./my_other_facts"
$ facter system_load users
system_load => 0.25
users => thomas,pat
```

External Facts

External Facts

External facts are the arbitrary executables or scripts that can be used as facts or to set facts statically with structured data. These executables or scripts can be written in Perl, C, or in a one-line text file.



Location of External Facts

The external facts are distributed in the Puppet module with pluginsync in the directory **<MODULEPATH>/<MODULE>facts.d/.**

- Pluginsync is a Puppet tool that ensures that all the nodes have the most recent version of the custom resources and facts in a module.
- If the user is not using pluginsync, then the external facts are placed in a standard directory whose location depends on the operating system.
- When using facter from the command line, the external fact directory is specified with **--external-dir** option in the facter command.

External Fact Directories Based on OS

Following are the three directories on Unix/Linux/OS X operating systems:

```
/opt/puppetlabs/facter/facts.d/
/etc/puppetlabs/facter/facts.d/
/etc/facter/facts.d/
```

Following is the directory on Windows operating system:

```
C:\ProgramData\PuppetLabs\facter\facts.d\
```

Configuring Facter

Configuring Facter

The Facter config file is used to manage the interaction of Puppet Facter with the system be it on the node or master system. It is also known as facter.conf.

The config file is composed of three sections, namely:

01 Facts

02 Global

03 CLI

NOTE

Facter does not automatically create the config file. It has to be created manually by the user.

Configuring Facter

Example of facter.conf file is given below:

```
facts : {
    blocklist : [ "file system", "EC2" ],
    ttls : [
        { "timezone" : 30 days },
    ]
}
global : {
    external-dir     : [ "path1", "path2" ],
    custom-dir       : [ "custom/path" ],
    no-exernal-facts : false,
    no-custom-facts  : false,
    no-ruby          : false
}

cli : {
    debug     : false,
    trace     : true,
    verbose   : false,
    log-level : "warn"
}
```

Facts Section in Facter Config File

Facts section in the facter.conf file contains the settings that affect fact groups, that is, a set of individual facts that get resolved together as they rely on the same underlying system information.

Facts section contains the following settings:

Setting Name	Purpose
Blocklist	To prevent all facts within the listed groups from being resolved when Facter runs
ttls	To cache the key-value pairs of groups and their duration to be cached

Global Section in Facter Config File

The global section of facter.conf file contains settings to control how Facter interacts with its external elements.

Global section contains the following settings:

Setting Name	Purpose
external-dir	To search for external facts in a list of directories
custom-dir	To search for custom facts in a list of directories
no-external*	To prevent Facter from searching external facts if set to true
no-custom*	To prevent Facter from searching custom facts if set to true

CLI Section in Facter Config File

The CLI section of facter.conf contains settings that affect Facter's command line output.

CLI section contains the following settings:

Setting Name	Purpose
debug	To output debug messages if set to true
trace	To print stacktraces from errors in custom facts
verbose	To output most detailed messages if set to true
log-level	To set minimum level of message severity that gets logged

Key Takeaways

- Users can use Puppet Forge to download and install advanced Puppet modules written by Puppet developers and the open source community.
- Puppet Hiera is a built-in data lookup system for Puppet and is used for separating and storing data from Puppet code.
- Hiera can be configured using hiera.yaml config file to define the directories where the Hiera data is stored.
- Puppet Facter is Puppet's cross-platform system profiling tool that is used to discover and report facts per node that are available in Puppet manifests as variables.
- Puppet Facter can be configured to manage the way it interacts with the system using facter.conf config file.