- Quests
- Glossary
- Learning Resources
- Puppet Docs

Quests

- Welcome
- Power of Puppet
- Resources
- Manifests
- Variables
- Conditional Statements
- Ordering
- Classes
- Modules
- Puppet Module Tool
- Afterword
- Glossary

Glossary of Puppet Vocabulary

An accurate, shared vocabulary goes a long way to ensure the success of a project. To that end, this glossary defines the most common terms Puppet users rely on.

attribute

Attributes are used to specify the state desired for a given configuration resource. Each resource type has a slightly different set of possible attributes, and each attribute has its own set of possible values. For example, a package resource (like `vim`) would have an `ensure` attribute, whose value could be `present`, `latest`, `absent`, or a version number:

```
package {'vim':
  ensure   => present,
  provider => apt,
}
```

The value of an attribute is specified with the => operator; attribute/value pairs are separated by commas.

agent

(or **agent node**)

Puppet is usually deployed in a simple client-server arrangement, and the Puppet client daemon is known as the "agent." By association, a computer running puppet agent is usually referred to as an "agent node" (or simply "agent," or simply "node").

Puppet agent regularly pulls configuration catalogs from a puppet master server and applies them to the local system.

catalog

A catalog is a compilation of all the resources that will be applied to a given system and the relationships between those resources.

Catalogs are compiled from manifests by a puppet master server and served to agent nodes. Unlike the manifests they were compiled from, they don't contain any conditional logic or functions. They are unambiguous, are only relevant to one specific node, and are machine-generated rather than written by hand.

class

A collection of related resources, which, once defined, can be declared as a single unit. For example, a class could contain all of the elements (files, settings, modules, scripts, etc) needed to configure Apache on a host. Classes can also declare other classes.

Classes are singletons, and can only be applied once in a given configuration, although the `include` keyword allows you to declare a class multiple times while still only evaluating it once.

Note:

Being singletons, Puppet classes are not analogous to classes in object-oriented programming languages. OO classes are like templates that can be instantiated multiple times; Puppet's equivalent to this concept is defined types.

classify

(or **node classification**)

To assign classes to a node, as well as provide any data the classes require. Writing a class makes a set of configurations available; classifying a node determines what its actual configuration will be.

Nodes can be classified with node definitions in the site manifest, with an ENC, or with both.

declare

To direct Puppet to include a given class or resource in a given configuration. To declare resources, use the lowercase `file {'/tmp/bar':}` syntax. To declare classes, use the `include` keyword or the `class {'foo':}` syntax. (Note that Puppet will automatically declare any classes it receives from an external node classifier.)

You can configure a resource or class when you declare it by including attribute/value pairs.

Contrast with "define."

define

To specify the contents and behavior of a class or a defined resource type. Defining a class or type doesn't automatically include it in a configuration; it simply makes it available to be declared.

define (noun)

(or **definition**)

An older term for a defined resource type.

define (keyword)

The language keyword used to create a defined type.

defined resource type

(or **defined type**)

See "type (defined)."

ENC

See external node classifier.

environment

An arbitrary segment of your Puppet site, which can be served a different set of modules. For example, environments can be used to set up scratch nodes for testing before roll-out, or to divide a site by types of hardware.

expression

The Puppet language supports several types of expressions for comparison and evaluation purposes. Amongst others, Puppet supports boolean expressions, comparision expressions, and arithmetic expressions.

external node classifier

(or **ENC**)

An executable script, which, when called by the puppet master, returns information about which classes to apply to a node.

ENCs provide an alternate method to using the main site manifest (`site.pp`) to classify nodes. An ENC can be written in any language, and can use information from any pre-existing data source (such as an LDAP db) when classifying nodes.

An ENC is called with the name of the node to be classified as an argument, and should return a YAML document describing the node.

fact

A piece of information about a node, such as its operating system, hostname, or IP address.

Facts are read from the system by Facter, and are made available to Puppet as global variables.

Facter can also be extended with custom facts, which can expose site-specific details of your systems to your Puppet manifests.

Facter

Facter is Puppet's system inventory tool. Facter reads facts about a node (such as its hostname, IP address, operating system, etc.) and makes them available to Puppet.

Facter includes a large number of built-in facts; you can view their names and values for the local system by running `facter` at the command line.

In agent/master Puppet arrangements, agent nodes send their facts to the master.

filebucket

A repository in which Puppet stores file backups when it has to replace files. A filebucket can be either local (and owned by the node being managed) or site-global (and owned by the puppet master). Typically, a single filebucket is defined for a whole network and is used as the default backup location.

function

A statement in a manifest which returns a value or makes a change to the catalog.

Since they run during compilation, functions happen on the puppet master in an agent/master arrangement. The only agent-specific information they have access to are the facts the agent submitted.

Common functions include `template`, `notice`, and `include`.

global scope

See scope.

host

Any computer (physical or virtual) attached to a network.

In the Puppet docs, this usually means an instance of an operating system with the Puppet agent installed. See also "Agent Node".

host (resource type)

An entry in a system's `hosts` file, used for name resolution.

idempotent

Able to be applied multiple times with the same outcome. Puppet resources are idempotent, since they describe a desired final state rather than a series of steps to follow.

(The only major exception is the `exec` type; exec resources must still be idempotent, but it's up to the user to design each exec resource correctly.)

inheritance (class)

A Puppet class can be derived from one other class with the `inherits` keyword. The derived class will declare all of the same resources, but can override some of their attributes and add new resources.

> **Note:** Most users should avoid inheritance most of the time. Unlike object-oriented programming languages, inheritance isn't terribly important in Puppet; it is only useful for overriding attributes, which can be done equally well by using a single class with a few parameters.

inheritance (node)

Node statements can be derived from other node statements with the `inherits` keyword. This works identically to the way class inheritance works.

Note:

Node inheritance **should almost always be avoided.** Many new users attempt to use node inheritance to look up variables that have a common default value and a rare specific value on certain nodes; it is not suited to this task, and often yields the opposite of the expected result. If you have a lot of conditional per-node data, we recommend using the Hiera tool or assigning variables with an ENC instead.

master

In a standard Puppet client-server deployment, the server is known as the master. The puppet master serves configuration catalogs on demand to the puppet agent service that runs on the clients.

The puppet master uses an HTTP server to provide catalogs. It can run as a standalone daemon process with a built-in web server, or it can be managed by a production-grade web server that supports the rack API. The built-in web server is meant for testing, and is not suitable for use with more than ten nodes.

manifest

A file containing code written in the Puppet language, and named with the `.pp` file extension. The Puppet code in a manifest can:

- Declare resources and classes
- Set variables
- Evaluate functions
- Define classes, defined types, and nodes

Most manifests are contained in modules. Every manifest in a module should define a single class or defined type.

The puppet master service reads a single "site manifest," usually located at `/etc/puppet/manifests/site.pp`. This manifest usually defines nodes, so that each managed agent node will receive a unique catalog.

metaparameter

A resource attribute that can be specified for any type of resource. Metaparameters are part of Puppet's framework rather than part of a specific type, and usually affect the way resources relate to each other.

module

A collection of classes, resource types, files, and templates, organized around a particular purpose. For example, a module could be used to completely configure an Apache instance or to set-up a Rails application. There are many pre-built modules available for download in the Puppet Forge.

namevar

(or **name**)

The attribute that represents a resource's **unique identity** on the **target system.** For example: two different files cannot have the same `path`, and two different services cannot have the same `name`.

Every resource type has a designated namevar; usually it is simply `name`, but some types, like `file` or `exec`, have their own (e.g. `path` and `command`). If the namevar is something other than `name`, it will be called out in the type reference.

If you do not specify a value for a resource's namevar when you declare it, it will default to that resource's title.

node (definition)

(or **node statement**)

A collection of classes, resources, and variables in a manifest, which will only be applied to a certain agent node. Node definitions begin with the `node` keyword, and can match a node by full name or by regular expression.

When a managed node retrieves or compiles its catalog, it will receive the contents of a single matching node statement, as well as any classes or resources declared outside any node statement. The classes in every *other* node statement will be hidden from that node.

node scope

The local variable scope created by a node definition. Variables declared in this scope will override top-scope variables. (Note that ENCs assign variables at top scope, and do not introduce node scopes.)

noop

Noop mode (short for "No Operations" mode) lets you simulate your configuration without making any actual changes. Basically, noop allows you to do a dry run with all logging working normally, but with no effect on any hosts. To run in noop mode, execute `puppet agent` or `puppet apply` with the `--noop` option.

notify

A notification relationship, set with the `notify` metaparameter or the wavy chaining arrow. (~>)

notification

A type of relationship that both declares an order for resources and causes refresh events to be sent.

ordering

Which resources should be managed before which others.

By default, the order of a manifest is not the order in which resources are managed. You must declare a relationship if a resource depends on other resources.

parameter

Generally speaking, a parameter is a chunk of information that a class or resource can accept.

pattern

A colloquial term, describing a collection of related manifests meant to solve an issue or manage a particular configuration item. (For example, an Apache pattern.) See also module.

plusignment operator

The +> operator, which allows you to add values to resource attributes using the ('plusignment') syntax. Useful when you want to override resource attributes without having to respecify already declared values.

provider

Providers implement resource types on a specific type of system, using the system's own tools. The division between types and providers allows a single resource type `package` to manage packages on many different systems (using, for example, `yum` on Red Hat systems, `dpkg` and `apt` on Debian-based systems, and `ports` on BSD systems).

Typically, providers are simple Ruby wrappers around shell commands, so they are usually short and easy to create.

plugin

A custom type, function, or fact that extends Puppet's capabilities and is distributed via a module.

realize

To specify that a virtual resource should actually be applied to the current system. Once a virtual resource has been declared, there are two methods for realizing it:

1. Use the "spaceship" syntax <| |>

2. Use the `realize` function

A virtually declared resource will be present in the catalog, but will not be applied to a system until it is realized.

refresh

A resource gets **refreshed** when a resource it subscribes to (or which notifies it) is changed.

Different resource types do different things when they get refreshed. (Services restart; mount points unmount and remount; execs usually do nothing, but will fire if the `refreshonly` attribute is set.)

relationship

A rule stating that one resource should be managed before another.

resource

A unit of configuration, whose state can be managed by Puppet. Every resource has a type (such as `file`, `service`, or `user`), a title, and one or more attributes with specified values (for example, an `ensure` attribute with a value of `present`).

Resources can be large or small, simple or complex, and they do not always directly map to simple details on the client -- they might sometimes involve spreading information across multiple files, or even involve modifying devices. For example, a `service` resource only models a single service, but may involve executing an init script, running an external command to check its status, and modifying the system's run level configuration.

resource declaration

A fragment of Puppet code that details the desired state of a resource and instructs Puppet to manage it. This term helps to differentiate between the literal resource on disk and the specification for how to manage that resource. However, most often, these are just referred to as "resources."

scope

The area of code where a variable has a given value.

Class definitions and type definitions create local scopes. Variables declared in a local scope are available by their short name (e.g. `$my_variable`) inside the scope, but are hidden from other scopes unless you refer to them by their fully qualified name (e.g. `$my_class::my_variable`).

Variables outside any definition (or set by an ENC) exist at a special "top scope;" they are available everywhere by their short names (e.g. `$my_variable`), but can be overridden in a local scope if that scope has a variable of the same name.

Node definitions create a special "node scope." Variables in this scope are also available everywhere by their short names, and can override top-scope variables.

Note:

Previously, Puppet used dynamic scope, which would search for short-named variables through a long chain of parent scopes. This was deprecated in version 2.7 and will be removed in the next version.

site

An entire IT ecosystem being managed by Puppet. That is, a site includes all puppet master servers, all agent nodes, and all independent masterless Puppet nodes within an organization.

site manifest

The main "point of entry" manifest used by the puppet master when compiling a catalog. The location of this manifest is set with the `manifest` setting in puppet.conf. Its default value is usually `/etc/puppet/manifests/site.pp` or `/etc/puppetlabs/puppet/manifests/site.pp`.

The site manifest usually contains node definitions. When an ENC is being used, the site manifest may be nearly empty, depending on whether the ENC was designed to have complete or partial node information.

site module

A common idiom in which one or more modules contain classes specific to a given Puppet site. These classes usually describe complete configurations for a specific system or a given group of systems. For example, the `site::db_slave` class might describe the entire configuration of a database server, and a new database server could be configured simply by applying that class to it.

subclass

A class that inherits from another class. See inheritance.

subscribe

A notification relationship, set with the `subscribe` metaparameter or the wavy chaining arrow. (`~>`)

template

A partial document which is filled in with data from variables. Puppet can use Ruby ERB templates to generate configuration files tailored to an individual system.

title

The unique identifier (in a given Puppet catalog) of a resource or class.

- In a class, the title is simply the name of the class.
- In a resource declaration, the title is the part after the first curly brace and before the colon; in the example below, the title is `/etc/passwd`:

```
file  { '/etc/passwd':
  owner => 'root',
  group => 'root',
}
```

- In native resource types, the name or namevar will use the title as its default value if you don't explicitly specify a name.
- In a defined resource type or a class, the title is available for use throughout the definition as the `$title` variable.

Unlike the name or namevar, a resource's title need not map to any actual attribute of the target system; it is only a referent. This means you can give a resource a single title even if its name has to vary across different kinds of system, like a configuration file whose location differs on Solaris.

top scope

See scope.

type

A kind of resource that Puppet is able to manage; for example, `file`, `cron`, and `service` are all resource types. A type specifies the set of attributes that a resource of that type may use, and models the behavior of that kind of resource on the target system. You can declare many resources of a given type.

type (defined)

(or **defined resource type;** sometimes called a **define** or **definition**)

A resource type implemented as a group of other resources, written in the Puppet language and saved in a manifest. (For example, a defined type could use a combination of `file` and `exec` resources to set up and populate a Git repository.) Once a type is defined, new resources of that type can be declared just like any native or custom resource.

Since defined types are written in the Puppet language instead of as Ruby plugins, they are analogous to macros in other languages. Contrast with native types.

type (native)

A resource type written in Ruby. Puppet ships with a large set of built-in native types, and custom native types can be distributed as plugins in modules. See the type reference for a complete list of built-in types.

Native types have lower-level access to the target system than defined types, and can directly use the system's own tools to make changes. Most native types have one or more providers, so that they can implement the same resources on different kinds of systems.

variable

A named placeholder in a manifest that represents a value. Variables in Puppet are similar to variables in other programming languages, and are indicated with a dollar sign (e.g. `$operatingsystem`) and assigned with the equals sign (e.g. `$myvariable = "something"`). Once assigned, variables cannot be reassigned within the same scope; however, other local scopes can assign their own value to any variable name.

Facts from agent nodes are represented as variables within Puppet manifests, and are automatically pre-assigned before compilation begins.

variable scoping

See scope above.

virtual resource

A resource that is declared in the catalog but will not be applied to a system unless it is explicitly realized.