DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Puppet Resources, Classes, and Modules

Learning Objectives

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

- Declare and modify Puppet resources
- Define and write Puppet classes
- Define and write Puppet modules
- Install Puppet module from an external repository
- Execute a module against a Puppet master with an agent

Puppet Resources

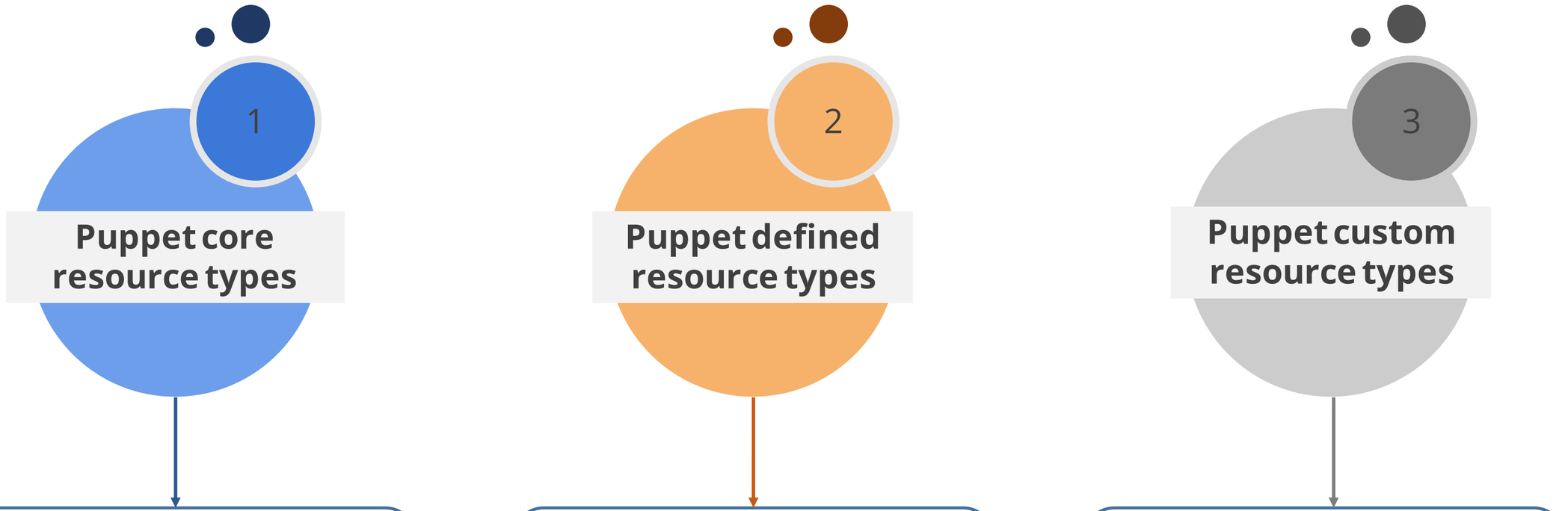
What are Puppet Resources?

A resource may represent any component in a system like a package, file, user, or service. They are the basic building blocks of any system and are used to perform configuration management operations on Puppet agents.

Types of Puppet Resources

Types of Puppet Resources

There are three types of Puppet resources. Puppet uses resources and resource types in order to describe the configurations for a system.



Characteristics of Puppet Resources

1

Declarative

Resources describe what Puppet should manage or modify in a system. It does not give any information related to the process used to manage the system.

2

Idempotent

Same resources can be used to configure a system multiple times. This gives the same result or output every time.

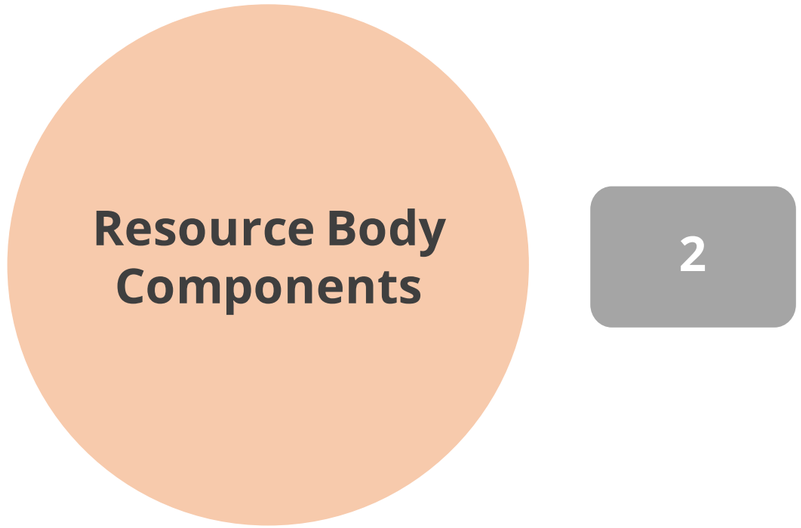
3

Unique

Resources in Puppet are unique. Two resources cannot be identical because each resource declares a desired end state of a system.

Resource Body Components

The three main resource body components are given below:



Title
The string that Puppet uses to identify a resource is called a title.

Attributes
The desired state of the resources are described by attributes. Each attribute handles some aspect of the resource.

Namevars
Namevar is the unique identity of the resource using which the target system identified the resource.

Resource Syntax

A basic resource syntax contains:

- The resource type without quotes
- A pair of curly braces
- Title string followed by a colon
- Any number of attributes and value pairs

Example:

```
file {
'/etc/passwd':
ensure => file,
owner => 'root',
group => 'root’,
mode => '0600', }
```

Viewing and Modifying Puppet Resources

Users can view and modify the state of a system's resource using the **puppet resource** command.

Viewing a resource state:

Syntax:

```
puppet resource
resource_type
```

Example:

```
puppet resource zone
output:
zone { 'global’:
  ensure   => 'running',
  brand    => 'solaris',
  iptype   => 'shared',
  zonepath => '/’, }
```

Modifying a resource state:

Syntax:

```
puppet resource
resoure_type state
```

Example:

```
puppet resource service
svc:/network/dns/client
:default enable=false
Notice:
/Service[svc:/network/d
ns/client:default]/enab
le: enable changed
'true' to 'false'
```

Assisted Practice

Viewing Puppet Resources Using Command Line

Problem Statement:

Use the command line to view the pre-defined Puppet resources

Assisted Practice: Guidelines

Steps to perform:

1. Use Puppet resource command to view the types of Puppet resources
2. Use Puppet description command to view a detailed description of the Puppet resources
3. Use Puppet resource command with an appropriate tag to list the resources with same resource types
4. Use Puppet resource command with an appropriate tag to view a particular resource

Assisted Practice

Creating and Removing Puppet Resources

Problem Statement:

Use the command line to create and remove Puppet resources

Assisted Practice: Guidelines

Steps to perform:

1. Use Puppet resource command to create a new user resource
2. Use command line to add attributes and values for the new user resource
3. Use command line to remove a user resource

Resource Relationships and Ordering

Resource Relationships and Ordering

By default, when not specified, Puppet applies resources in the order they are declared in the manifest.

For cases when a set of resources must always be managed in a particular sequence, Puppet offers the following ways to specify the sequence:

1. Relationship metaparameters
2. Chaining arrows
3. The require function

Relationship Metaparameters

There are four metaparameters that can be used to establish the order of resources. They can be used as attributes in any resource and value of metaparameter must be a reference to a resource.

Following are the four metaparameters:

before	Applies a resource **before** the target resource
require	Applies a resource **after** the target resource
notify	Applies a resource **before** the target resource and refreshes the target resource by notifying if there are any changes in the dependencies
subscribe	Applies a resource **after** the target resource and refreshes the resource if the target resources change

Relationship Metaparameters

Using before and require metaparameters create the same sequence. They can be used as shown in the following examples:

Before Metaparameter

```
package { 'openssh-server':
  ensure => present,
  before =>
File['/etc/ssh/sshd_config'],
}
```

Require Metaparameter

```
file { '/etc/ssh/sshd_config':
  ensure  => file,
  mode    => '0600',
  source  =>
'puppet:///modules/sshd/sshd_c
onfig',
  require => Package['openssh-
server'],
}
```

Both of the above resource declarations specify that the package ['openssh-server'] resource must be applied before file['/etc/ssh/sshd_config'] resource.

Relationship Metaparameters

Using notify and subscribe metaparameters create the same sequence. They can be used as shown in the following examples:

Notify Metaparameter

```
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => '0600',
  source =>
'puppet:///modules/sshd/sshd_c
onfig',
  notify => Service['sshd'],
}
```

Subscribe Metaparameter

```
service { 'sshd':
  ensure    => running,
  enable    => true,
  subscribe =>
File['/etc/ssh/sshd_config'],
}
```

Both of the above resource declarations specify that the Service['sshd'] resource should be refreshed after File['/etc/ssh/sshd_config'] resource is applied and vice-versa.

Chaining Arrows

Puppet lets users define relationships on the basis of the sequence of two or more resources using the chaining arrow operators.

Types of chaining arrow operators:

Ordering arrow (->)	Applies the resource on the left before the one on the right
Notifying arrow (~>)	Applies the resource on the left before the one on right. If the left-hand resource changes, the right-hand resource gets refreshed

Operands to be used on either side of the operators include:

1. Resource declarations
2. Resource definitions

Chaining Arrows

An example of using chaining arrows with **resource declarations** in the Puppet manifests is given below:

```
Package['ntp'] -> File['/etc/ntp.conf'] ~> Service['ntpd']
```

NOTE

An operand can be shared between more than two resource declarations, linking the resources together in a timeline.

Chaining Arrows

An example of using chaining arrows with **resource definitions** in the Puppet manifests is given below:

```
package { 'openssh-server':
  ensure => present,
} # and then:
-> file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => '0600',
  source => 'puppet:///modules/sshd/sshd_config',
} # and then:
~> service { 'sshd':
  ensure => running,
  enable => true,
}
```

The require Function

The require function is used to declare a resource in order to make it a dependency for other resources.

Example:

```
class wordpress {
  require apache
  require mysql
  ...
}
```

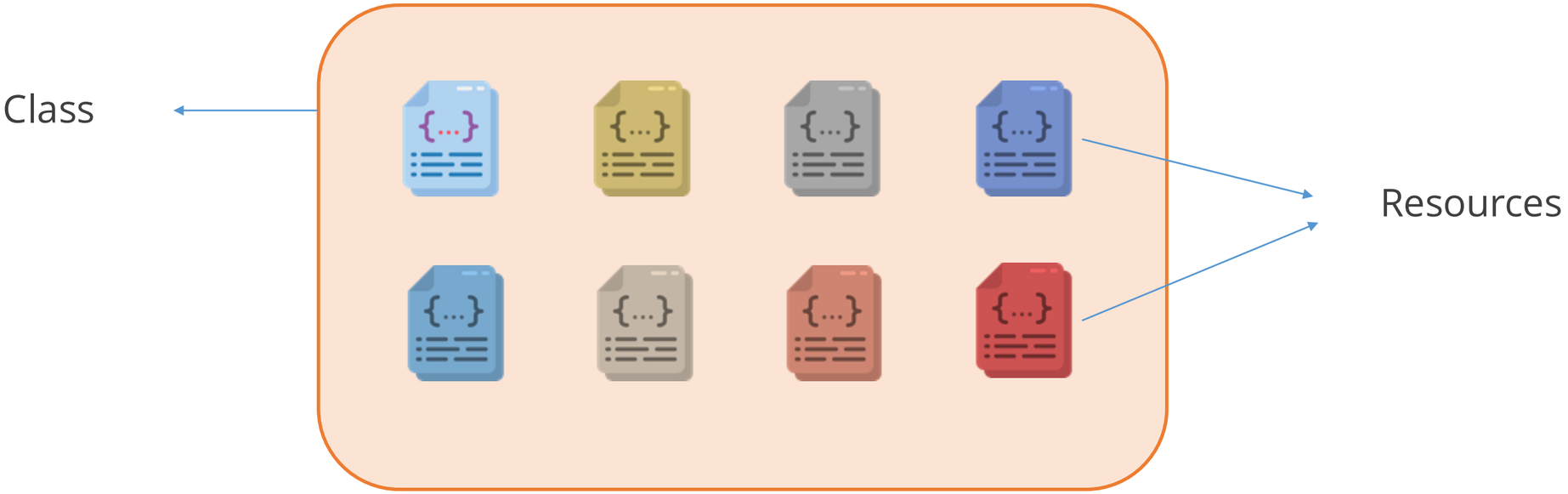
NOTE

Require function does not have a reciprocal form like relationship metaparameters and chaining arrows.

Puppet Classes

What are Puppet Classes?

A Puppet class is a collection of resources that are grouped together and given a name.
The grouped resources form a large, customized configuration to create a desired state in a system by invoking the class with its name.



Defining a Puppet Class

Defining a class includes:

The **class** keyword followed by the name of the class

A comma-separated parameter list enclosed in a pair of curly braces

The **Inherits** keyword followed by a single class name

A class body should contain one or more resource declarations

Syntax for Defining a Puppet Class

Syntax:

```
class class_name {

class body

}
```

Example:

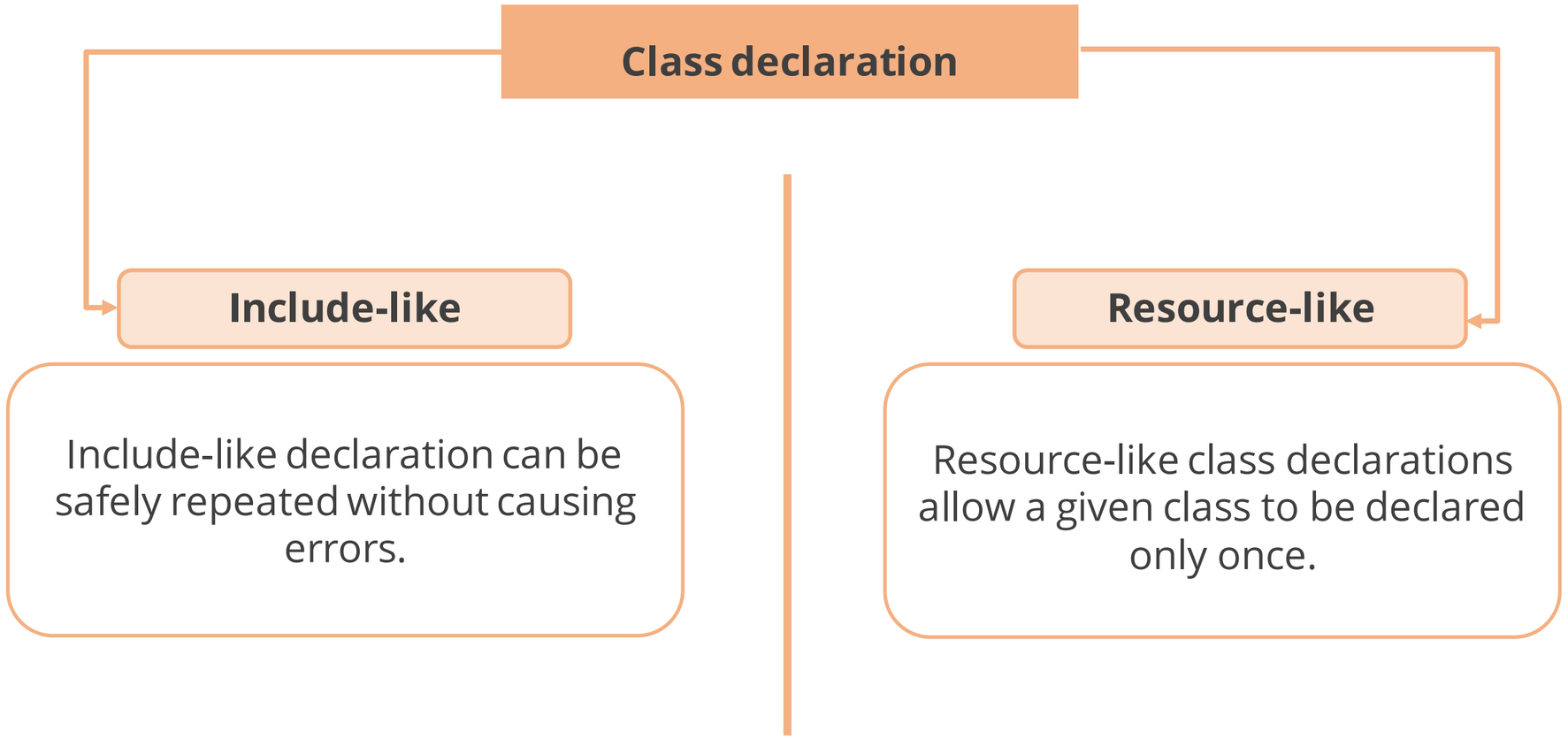
```
class base::linux {
  file { '/etc/passwd':
    owner => 'root',
    group => 'root',
    mode  => '0644',
  }
  file { '/etc/shadow':
    owner => 'root',
    group => 'root',
    mode  => '0440',
  }
```

Class Parameters and Variables

- Parameters are entities that allow a class to request external data.
- Class parameters can be used as normal variables inside the class definition.
- Users can define parameters with or without datatypes.
- The variables **$title** and **$name** are set to the class name by default.
- Values of these variables are taken as inputs from users.

Declaring a Class

Declaring classes in a Puppet manifest will add all the resources included in the class to the catalog.



Include Function

Users can add classes in a catalog using the include function.
The include function can be used in the following ways:

A single class name

```
include base::linux
```

A single class reference

```
include
Class['base::linux']
```

A comma separated list of class names

```
include base::linux, apache
```

An array of class names

```
$my_classes =
['base::linux', 'apache']
include $my_classes
```

Assisted Practice

Defining and Declaring Puppet classes

Problem Statement:

Define and Declare a Puppet class in a Puppet manifest

Assisted Practice: Guidelines

Steps to perform:

1. Define a Puppet class with a package resource in it
2. Add a service resource in the previously defined class
3. Declare the class in the manifest using include-like function

Puppet Modules

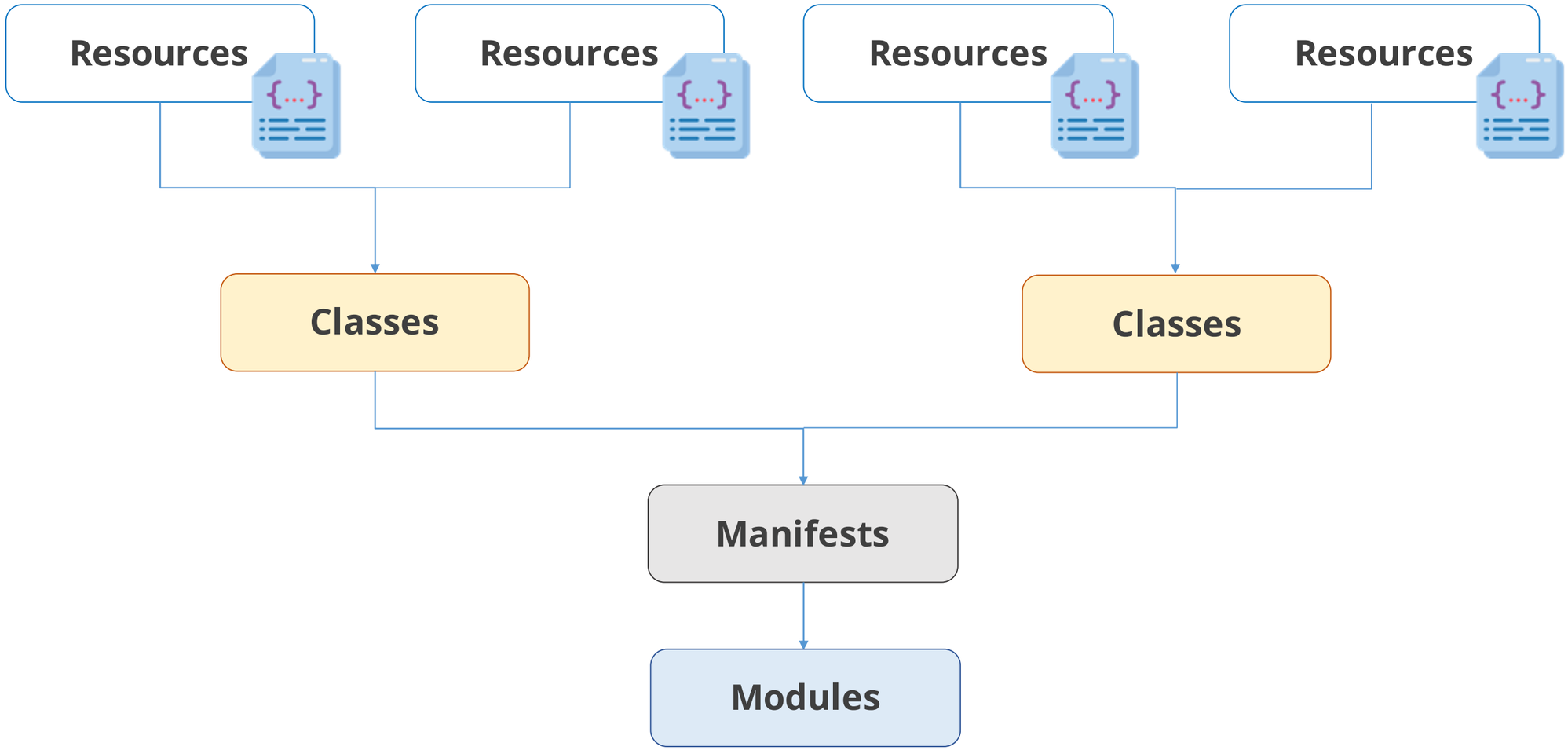
Puppet Modules

Puppet module is a tree directory that contains the files needed for a given configuration such as a manifest. It is a reusable and shareable entity in Puppet.

- Puppet modules can be custom written or installed from Puppet.
- Module names should contain only lowercase alphabets, numbers, and underscores.
- Module names should begin with a lowercase letter.

Puppet Modules

The following diagram shows the relationship between Puppet modules and other Puppet components:



Installing Puppet Modules

Puppet modules can be installed from module repositories that contain pre-built modules using command line.

Use the command given below for installation:

```
puppet module install --module_repository http://dev-forge.example.com
puppetlabs-apache
```

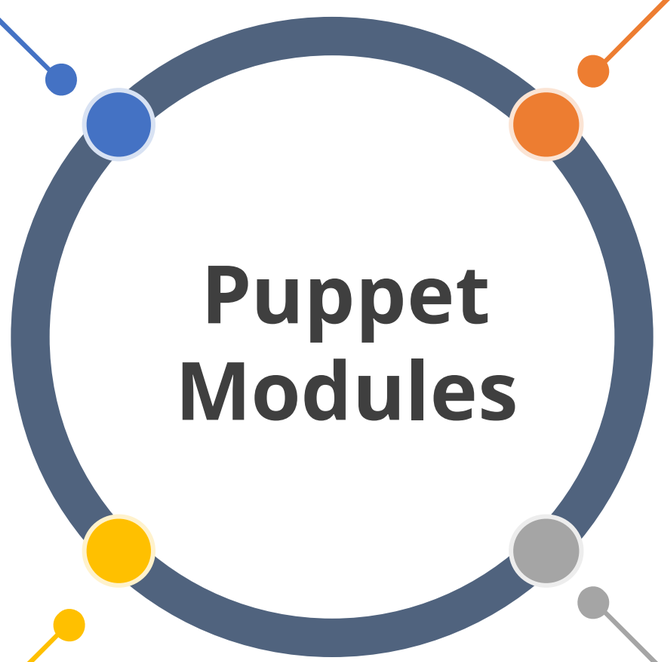
NOTE

To change the default module repository, users can edit the module_repository setting in the puppet.conf file.

Writing Puppet Modules

A module includes:

The <Module> class
The main class of the module that shares the same name as the module and is defined in init.pp file



The config class
Contains resources related to configuring the installed software

The install class
Contains all of the resources related to installing the software

The service class
Contains service resources as well as anything related to the running state of the software

Puppet Module Structure

Puppet Module Structure

Puppet modules have a specific directory structure where nodes can discover and load classes, resources, facts, manifests, and other Puppet components.

The structure of the directory is given below:

```
module-name/
|- manifets
 |- init.pp
|- files
|- templates
|- facts
|- files
 |- service.conf
|- functions
```

Puppet Module Structure

All the sub-directories in a module hold different entities required by Puppet master to apply configuration management to agent nodes.

The following are the most common subdirectories with their respective content:

manifests/	Contains all manifests in the module
files/	Contains static files that can be served to the agent nodes
data/	Contains data files that specify parameter default
templates/	Contains templates that can be used by the manifests to generate variable values

Puppet Module Structure

facts/	Contains external facts that are synced with agent nodes
functions/	Contains custom functions written by users in Puppet language
lib/	Contains plugins such as custom facts and custom resource types
puppet/	Contains custom functions and resource providers

Puppet Module Plugins

Puppet Module Plugins

Plugins in Puppet modules enable various features such as custom facts and functions to manage the agent nodes. Plugins can be in-built in the module as well as developed by users.

The following are the most common subdirectories with their respective content:

Plugin type	Description	Used by	Module Subdirectory
Custom facts	Facts provide information about a specific system state.	Puppet agents	lib/facter
External facts	External facts provide a way to set facts statically with structured data.	Puppet agents	Facts/
Puppet functions	Functions are written in Puppet language to return calculated values.	Only Puppet server	functions/

Puppet Module Plugins

The following are the most common subdirectories with their respective content:

Plugin type	Description	Used by	Module Subdirectory
Ruby functions	Functions are written in Ruby language to return calculated values.	Only Puppet server	lib/puppet/functions
Resource types	These are written in Puppet to add new resource types to Puppet.	Puppet agents and Puppet server	Facts/lib/puppet/type
Resource providers	These are written in Puppet to add new resource providers to Puppet.	Puppet agents and Puppet server	functions/lib/puppet/provider

Assisted Practice
Creating a Puppet Module

Problem Statement:

Create a custom module on the Puppet master

Assisted Practice: Guidelines

Steps to perform:

1. Create the module structure
2. Generate the metadata for the module
3. Explore the module directories

Assisted Practice

Installing Module from the Command Line

Problem Statement:

Install, upgrade, and uninstall modules using command line

Assisted Practice: Guidelines

Steps to perform:

1. Install Puppet module from Puppet Labs
2. Upgrade Puppet module to the latest version available
3. Uninstall Puppet module

Assisted Practice

Managing Modules from the Command line

Problem Statement:

List, search, and install modules from external repositories using the command line

Assisted Practice: Guidelines

Steps to perform:

1. List the modules installed in the specified module path
2. Search the Puppet labs module repositories for a module
3. Install module from another repository

Assisted Practice

Installing PHP from Puppet master to Puppet Agent

Problem Statement:

Set up a Puppet module to install PHP from Puppet master to Puppet agent

Assisted Practice: Guidelines

Steps to perform:

1. Install PHP modules in Puppet master
2. Pull configurations from Puppet master to Puppet agent
3. Verify PHP installation in Puppet agent

Key Takeaways

- A Puppet resource may represent any components of a system such as package, file, users, and services.
- By default, Puppet resources are applied in the order they are declared in the main manifest. To change the order dynamically, user can use relations metaparameters, chaining arrows, and require function.
- A Puppet class is a collection of resources that form a large and customized configuration to create a desired state in a system.
- A Puppet modules is a collection of manifests, classes, and resources organised in a specific directory structure.