

Approaches to Extending Resources

©2018 Chef Software Inc.

9-1



You express the state of your infrastructure with resources, defined in recipes, encapsulated in cookbooks. Chef provides a core set of resources (dependent on your version of Chef and your platform). These core resources allow you to express the desired state of your infrastructure in a majority of situations. They can also be combined together to express the desired state where these individual resources fall short.

Early on when working with Chef these core resources and their ability to be combined will handle a majority of the configuration management issues that you face. After awhile you will come across more specific resource needs that have not yet been created or perhaps help describe a common set of resources you continue to use together.

When a necessary resource does not exist or when you want to express a group of resources a single resource, Chef provides a few ways to accomplish this.

Objectives

After completing this module, you should be able to:

- Describe the difference between:
 - Custom Resources
 - Definitions
 - Heavy-Weight Resource-Providers
 - Light-Weight Resource-Providers

After completing this module you will be able to describe the differences between Custom Resources, Definitions, Heavy-Weight Resource Providers and Light-Weight Resource Providers.

CONCEPT

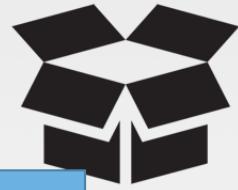


Approaches to Extending Resources

- 1 Pure Ruby (Heavy-Weight Resource-Providers / HWRP)
- 2 Definitions
- 3 Light-Weight Resource-Providers (LWRP)
- 4 Custom Resources

Having reached the limit of the core set of resources presents a new set of challenges before you. Fortunately these challenges are not insurmountable because of some of the design choices Chef has made to make it possible to extend its functionality. Chef is a maturing product that continues to evolve to bring joy to its users. While we are going to focus on Custom Resources it is important that have a basic understanding of these other implementations.

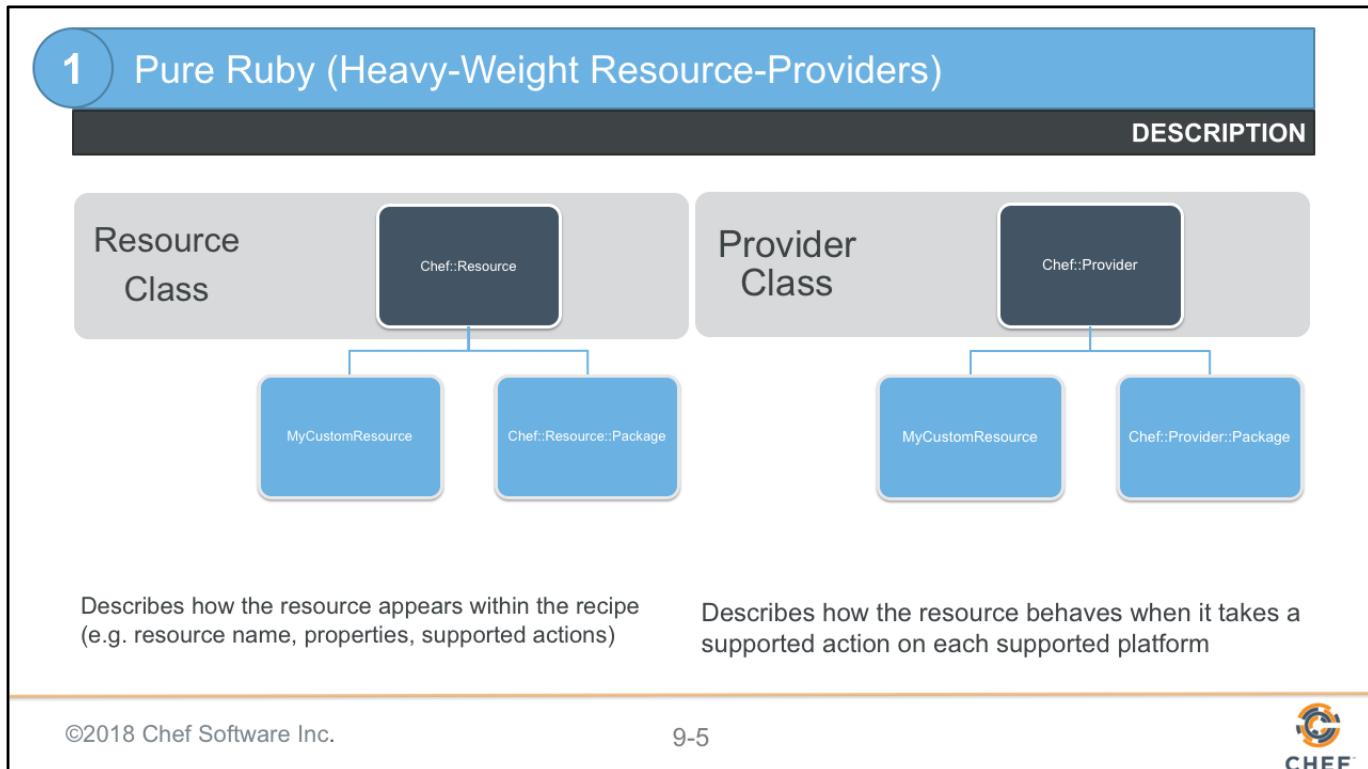
CONCEPT



1 Pure Ruby (Heavy-Weight Resource-Providers / HWRP)

- Description
- File and Folder Structure
- Implementation Language & Usage
- Benefits & Drawbacks

I will provide a description of each, explain the files and folder structure, take a quick look at how each is implemented, and then talk about any requirements or limitations when pursuing this implementation choice.



Chef's core resources are written in Ruby. The first approach to creating your own resources is to create your own with Ruby classes. These pure Ruby implementations of Resources is often referred to as Heavy-Weight Resource-Provider, or HWRP. Each resource defined in Chef is defined in two classes which sub-class the core Chef Resource and Chef Provider class. Sub-classing is an object-oriented programming term that means to inherit characteristics (e.g. methods and variables) from the parent class. Within the subclass you are required to override specific methods for the class to behave as a resource within the system.

The Chef::Resource class describes how the resource appears within the recipe; the interface. The Chef::Provider class describes how the resource will act when it takes one of the supported action on each supported platform.

1 Pure Ruby (Heavy-Weight Resource-Providers)

STRUCTURE

my_cookbook

```
libraries/
• [my_custom_resource]_resource.rb
• [my_custom_resource]_provider.rb
```

They are stored within the libraries folder in separate files for the resource and the provider. The file names are snake case representations of the class name stored within the file.

An HWRP, as pure Ruby, is stored in within the 'libraries' directory. Each class, one for the resource and the provider, are stored in separate files. The name of the file matches the class name except it has been snake-cased. Snake-casing lower cases the class name and places underscores between letters where capital letters used to exist. This is a common Ruby practice and one enforced by Rubocop. All the Ruby files within that directory are evaluated after the cookbook is synchronized and loaded.

1 Pure Ruby (Heavy-Weight Resource-Providers)

IMPLEMENTATION LANGUAGE - RESOURCE

libraries/apache_vhost_resource.rb

```
class Chef
  class Resource
    class ApacheVhost < Chef::Resource
      def initialize(name, run_context=nil)
        super
        @resource_name = :apache_vhost          # Defining the resource name
        @provider = Chef::Provider::ApacheVhost # Specifying which Provider to use
        @action = :create                      # Setting the default action
        @allowed_actions = [:create, :remove]   # Setting the list of actions
        # ... SETUP ANY DEFAULT VALUES HERE ...
      end

      def site_name(arg=nil)
        set_or_return(:site_name, arg, :kind_of => String)
      end
    end
  end
end
```

When defining the resource for a Heavy-Weight Resource-Provider you sub-class the Chef Resource class. The initialize method is overridden to specify new default values and allows us to configure the class as necessary when the resource is created in memory. Each potential attribute is defined as a method which uses a helper to setup the default values, value types it supports, etc.

1 Pure Ruby (Heavy-Weight Resource-Providers)

IMPLEMENTATION LANGUAGE - PROVIDER

libraries/apache_vhost_provider.rb

```
class Chef
  class Provider
    class ApacheVhost < Chef::Provider
      def load_current_resource
        @current_resource ||= Chef::Resource::ApacheVhost.new(new_resource.name)

        @current_resource.site_name(new_resource.site_name)
        # ... remaining properties defined in the resource
        @current_resource
      end

      def action_create
        # ... code that creates the resource on all supported platforms ...
      end
    end
  end
end
```

When defining the provider for a Heavy-Weight Resource-Provider you sub-class the Chef Provider class. The initialize method does not have to be overridden. The load_current_resource method must be overridden and is where the configuration from the resource is created and configured for use in each of the supported actions. The actions here are defined as methods with the prefix 'action_' and within them you would define the code necessary to perform the operations for this resource.

Chef provides additional helpers to allow you to shell out to perform operations on the system. You also have the entire Ruby language and any gems that might be packaged with the Chef DK (or you have added to Chef DK) at your disposal.

1 Pure Ruby (Heavy-Weight Resource-Providers)

USAGE

```
recipes/default.rb
```

```
apache_vhost 'welcome' do
  action :delete
end

apache_vhost 'users'

apache_vhost 'admins' do
  site_port 8080
end
```

The resource would now be available within any recipe defined in this cookbook or any cookbook that adds this cookbook as a dependency. Here in this example recipe the resources delete and creates apache sites. All three of the resources rely on the site name attribute being tied to the name provided to the resource. The first deletes the welcome site. The next two both rely on the default action of create. The second resource assumes the default site port value.

1 Pure Ruby (Heavy-Weight Resource-Providers)

BENEFITS & DRAWBACKS

- Available in some of the earliest versions of Chef
- Allows for extremely flexible and powerful resource implementations
- Requires knowledge of Ruby
- Requires knowledge of Object-Oriented Programming techniques

HWRP are incredibly useful when you need the full power of Ruby to implement your own resource. However, they come at the cost of understanding a number of Object-Oriented Programming techniques and the Ruby language. When exploring community cookbooks you may find examples of these resources in use.

2 Definitions

DESCRIPTION

```
recipes/admins_site.rb
```

```
apache_vhost 'admins' do
  site_name 'admins'
end
```

```
recipes/users_site.rb
```

```
apache_vhost 'users' do
  site_name 'users'
end
```

```
recipes/dogs_site.rb
```

```
...
```

```
definitions/apache_vhost.rb
```

```
define :apache_vhost site_name: 'default' do
  directory ...
  template ...
  file ...
end
```

Definitions behaves like a compile time macro that is reusable across recipes. Macros allow you to write a small amount of code that expands out into the contents of the definition wherever it is found within the recipes. With a definition you give it a name, provide parameters, and specify a block of code.

2 Definitions

STRUCTURE

my_cookbook

```
definitions/
• [my_definition_name].rb
```

They are stored within the definitions folder and often the name of the definition defines of the file.

The code that defines the definition is stored within a definitions directory in a Ruby file that is processed with the definition Domain Specific Language.

2 Definitions

IMPLEMENTATION LANGUAGE

```
definitions/apache_vhost.rb
```

```
define :apache_vhost site_name: 'default', site_port: 80 do
  directory "/srv/apache/#{params[:site_name]}/html" do
    recursive true
    mode '0755'
  end

  templates "/srv/apache/#{params[:site_name]}/html" do
    source 'conf.erb'
    mode '0644'
    variables(document_root: "/srv/apache/#{params[:site_name]}/html", port: params[:site_port])
    mode '0755'
    notifies :restart, 'service[httpd]'
  end

  # ... remaining resources ...
end
```

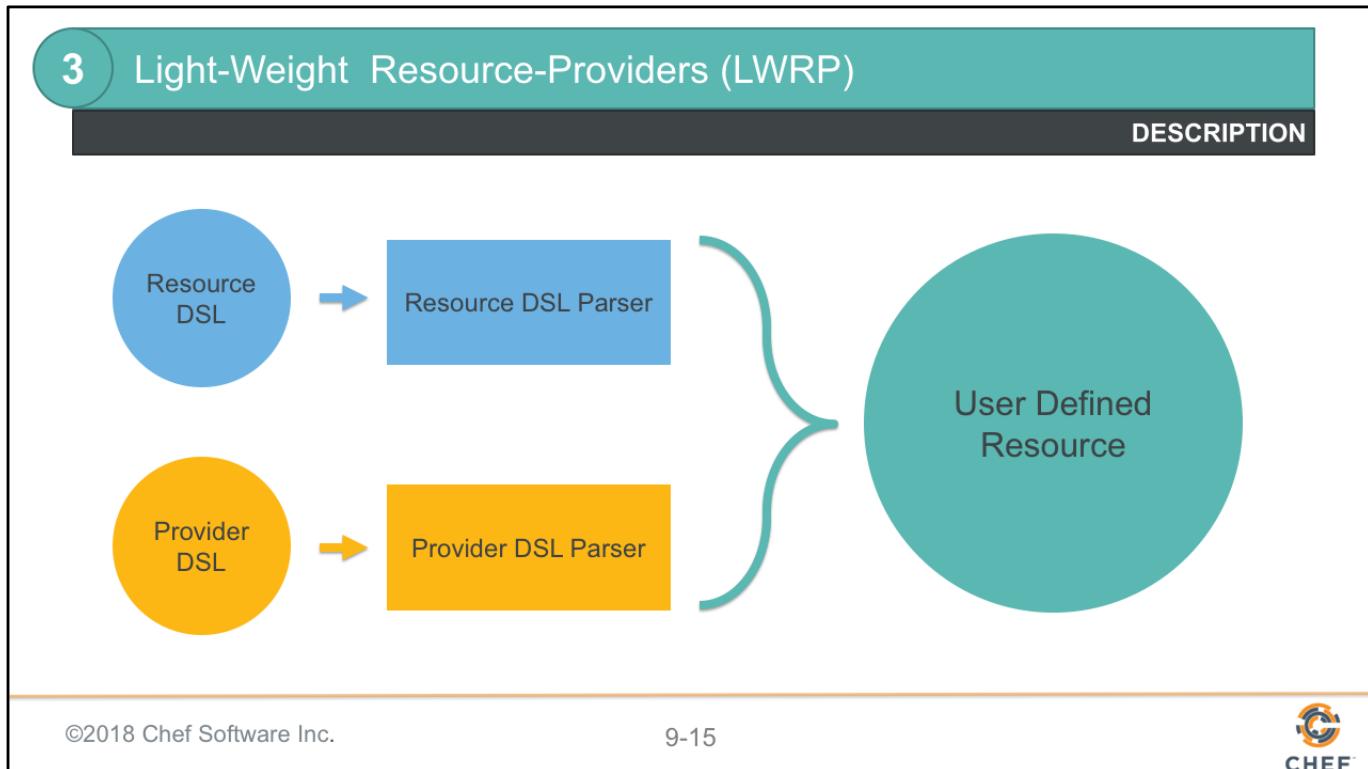
When creating a definition you specify a name and a hash of any parameters you wish to provide. Within the definition the parameters are retrievable from a hash named `params`. The use of the definition within a recipe looks similar to a resource but that is not the case. Definitions cannot notify other resources, subscribe to notifications from other resources, (i.e. `notifies` and `subscribes`) and cannot employ guards (i.e. `only_if` and `not_if`).

2 Definitions

BENEFITS & DRAWBACKS

- Available in some of the earliest versions of Chef
- Allows for code re-use within recipes
- Definition usage could be mistaken for a true resource
- Definitions do not support notifications (`subscribes` and `notifies`)

Definitions shipped in some of the earliest versions of Chef and are still supported today. However, as of Chef 12.5 it is strongly recommended that you choose a solution built with custom resources.



Light-Weight Resource-Provider, or LWRP, are Chef resources defined in two Domain Specific Languages (DSL) that allow you to create resources without having to understand the complexity presented by HWRP.

An LWRP is as much a resource as the core resources defined in Chef. The resource and the provider is parsed and converted into Ruby objects.

3

Light-Weight Resource-Providers (LWRP)

STRUCTURE

my_cookbook

```
resources/
  • [my_resource_name].rb
providers/
  • [my_resource_name].rb
```

An LWRP is defined in two separate files that share the same name. The resource definition is defined in the resources directory of the cookbook; the provider definition in the providers directory.

The cookbook name is combined with the file name to create the name of the resource.

A single LWRP definition is defined in two separate files. The file is named exactly the same but one file resides in the 'resources' directory; the other in the 'providers' directory. Both of these files are parsed after the cookbook is synchronized and loaded. Each file's DSL is then converted into Ruby class at runtime.

Within the file in the 'resources' directory you define the interface for the custom resource. There, within a resource DSL, you can specify a name of the resource, the list of available actions, the default action, and the properties that may be set for the resource. Within the file in the 'providers' directory you define the implementation for the custom resource. There, within a provider DSL, you specify what happens when an action is chosen.

3 Light-Weight Resource-Providers (LWRP)

IMPLEMENTATION LANGUAGE - RESOURCE

```
resources/vhost.rb

actions :create, :delete

default_action :create

attribute :site_name, String, name_attribute: true
attribute :site_port, Integer, default: 80
```

Within the resources file you specify the available actions, the default action, and the supported attributes that can be used when specifying the resource.

3 Light-Weight Resource-Providers (LWRP)

IMPLEMENTATION LANGUAGE - PROVIDER

```
providers/vhost.rb
```

```
action :create do
  directory "/srv/apache/#{new_resource.site_name}/html" do
    recursive true
    mode '0755'
  end

  templates "/srv/apache/#{new_resource.site_name}/html" do
    source 'conf.erb'
    mode '0644'
    variables(document_root: "/srv/apache/#{new_resource.site_name}/html",
              port: new_resource.site_port)
    mode '0755'
    notifies :restart, 'service[httpd]'
  end

  # ... remaining resources ...
end
```

Within the provider definition you specify action blocks for each of the actions defined in the resource file. Within the action you specify resources as if you are defining a small recipe. The attributes defined for the resource are available within the action through a local variable or method named 'new_resource'.

3 Light-Weight Resource-Providers (LWRP)

USAGE

recipes/default.rb

```
apache_vhost 'welcome' do
  action :delete
end

apache_vhost 'users'

apache_vhost 'admins' do
  site_port 8080
end
```

The name of the cookbook is combined with the name of the resource/provider file name with an underscore to create the user defined resource. This was explicitly defined in the HWRP but is automatically generated.

Otherwise this is the same results as the one defined by the HWRP.

3 Light-Weight Resource-Providers (LWRP)

BENEFITS & DRAWBACKS

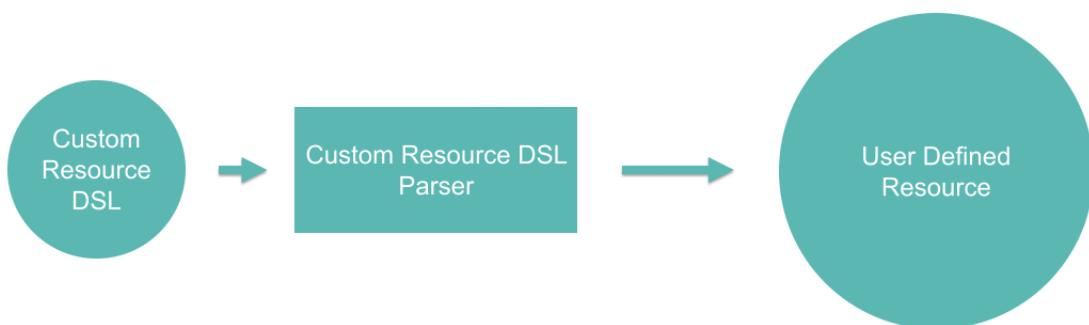
- Available in 0.7.12 version of Chef
- Allows for a real resource definition without understanding Ruby (vs. HWRP)
- Resource and provider implementation require learning a new DSL
- Complete resource definition is spread across two files

Implementing resources with LWRP is not the favored way to develop a resource in later versions of Chef (Chef 12.5). However, they are still in wide use within older cookbooks like those found within the Chef Supermarket.

4

Custom Resources

DESCRIPTION



Custom Resources are Chef resources defined in a Domain Specific Language (DSL) that allow you to create resources without having to understand the complexity presented by HWRP. At its core it is a simplification of the work done with LWRP.

An custom resource is as much a resource as the core resources defined in Chef. A custom resource definition is defined in a single file that resides in the 'resources' directory. This file is parsed after the cookbook is synchronized and loaded. The custom resource DSL is then converted into Ruby class at runtime.

4 Custom Resources

STRUCTURE

my_cookbook

```
resources/
• [my_resource_name].rb
```

A custom resource is defined in a single file within the resources directory.

Within the file in the 'resources' directory you define the interface and the implementation for the custom resource. This is written in a custom resource DSL where you can specify the name of the resource, the default action, the properties that may be set, and all the actions that the resource supports.

4 Custom Resources

IMPLEMENTATION LANGUAGE

```
resources/vhost.rb

resource_name :apache_vhost

property :site_name, String, name_attribute: true
property :site_port, Integer, default: 80

action :create do
  directory "/srv/apache/#{new_resource.site_name}/html" do
    recursive true
    mode '0755'
  end

  # ... remaining resources ...
end

# ... remaining actions ...
```

The custom resource implementation is similar to the LWRP except all of the details that describe the resource are combined into a single file. The custom resource DSL is similar to one defined for the LWRP resource and LWRP provider DSL. It is an evolution of the LWRP implementation with some minor changes. The attributes are instead called properties and when used within the action implementations they no longer require the 'new_resource' local variable or method. The default action is assumed to be the first action defined in this file: create.

4 Custom Resources

USAGE

recipes/default.rb

```
apache_vhost 'welcome' do
  action :delete
end

apache_vhost 'users'

apache_vhost 'admins' do
  site_port 8080
end
```

The result is the same here as the HWRP and LWRP.

The default action is determined by the first action listed in the custom resource definition.

4 Custom Resources

BENEFITS & DRAWBACKS

- Available in 12.5.0 version of Chef
- Allows for a real resource definition without understanding Ruby (vs. HWRP)
- Complete resource definition is defined in a single file (vs. LWRP)
- Custom resource implementation require learning a new DSL

Implementing resources with a custom resource is the current favored way to develop a resource for versions of Chef 12.5.X or greater. They are easier to implement than a pure Ruby implementation and are defined in a single file compared to the LWRP implementation.

CONCEPT



Approaches to Extending Resources

- 1 Pure Ruby (Heavy-Weight Resource-Providers / HWRP)
- 2 Definitions
- 3 Light-Weight Resource-Providers (LWRP)
- 4 Custom Resources

As you can see there are more than a few ways to extend Chef and create a resource or resource-like implementation within your recipes.

DISCUSSION



Discussion

Which approaches require you to define your solution in two separate files?

What are the limitations of choosing the Definitions approach?

What are some differences between LWRP and Custom Resources?

Given a Chef version prior to 12.5.0, which approach would you choose?

As a group, let's answer these questions.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

DISCUSSION



Q&A

What questions can we answer for you?

What questions can we answer for you?

