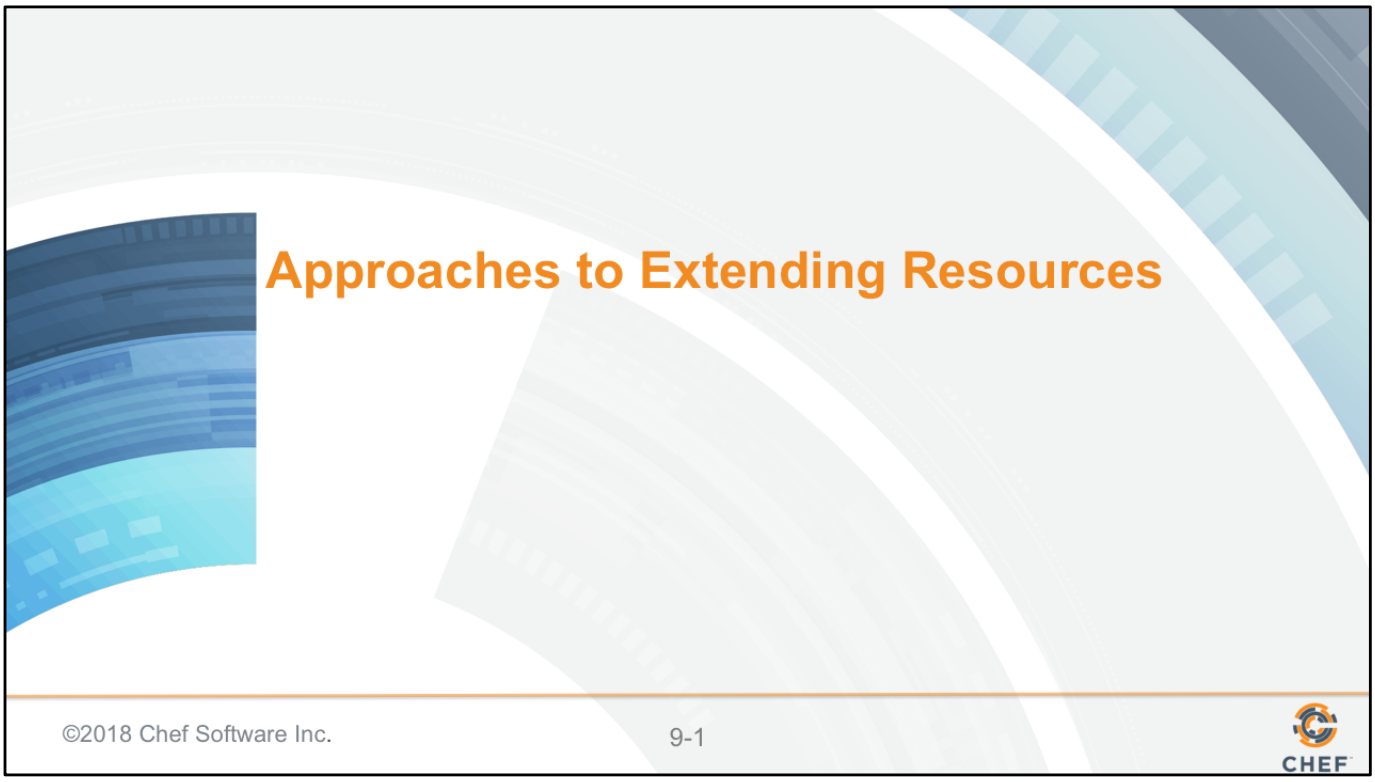


You express the state of your infrastructure with resources, defined in recipes, encapsulated in cookbooks. Chef provides a core set of resources (dependent on your version of Chef and your platform). These core resources allow you to express the desired state of your infrastructure in a majority of 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.

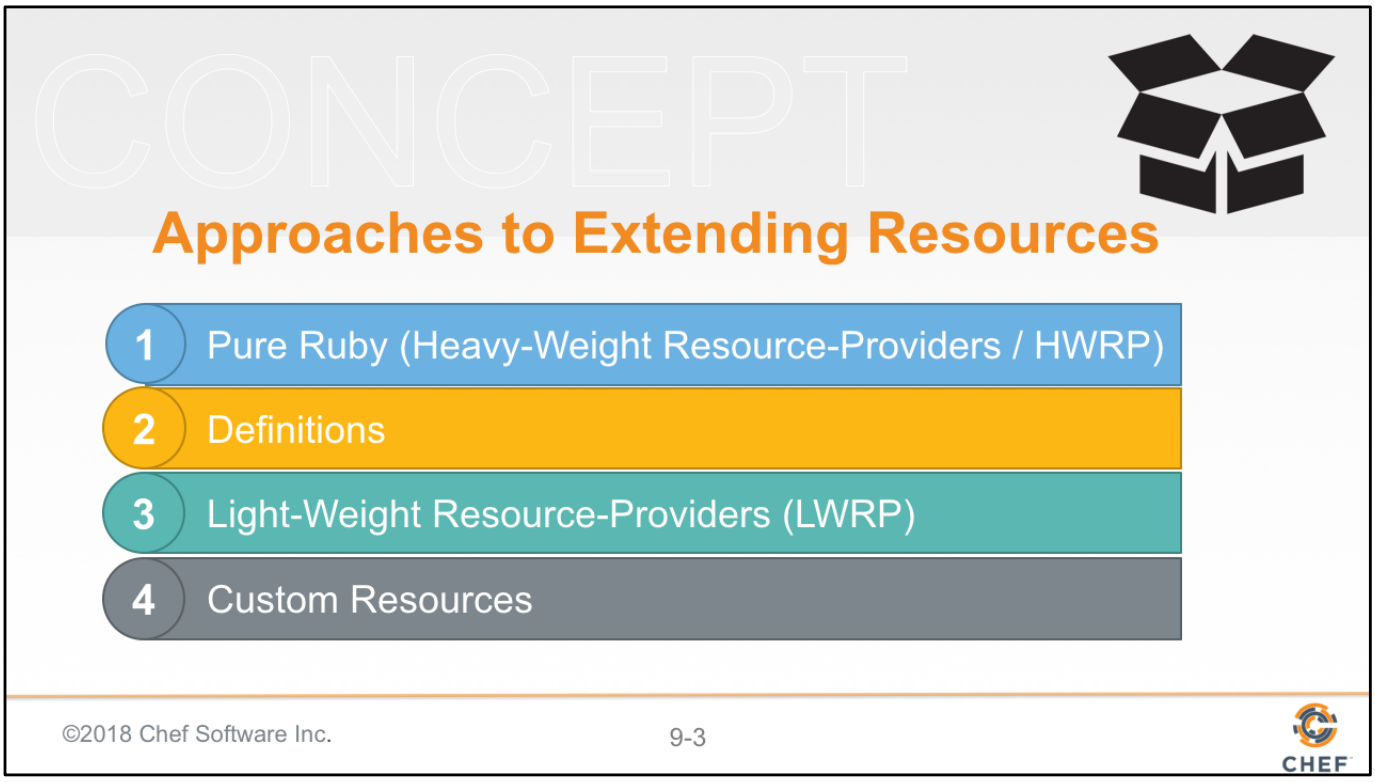
When a necessary resource does not exist or when you want to express a group of resources a single resource, Chef provides a few ways to accomplish this.

Objectives

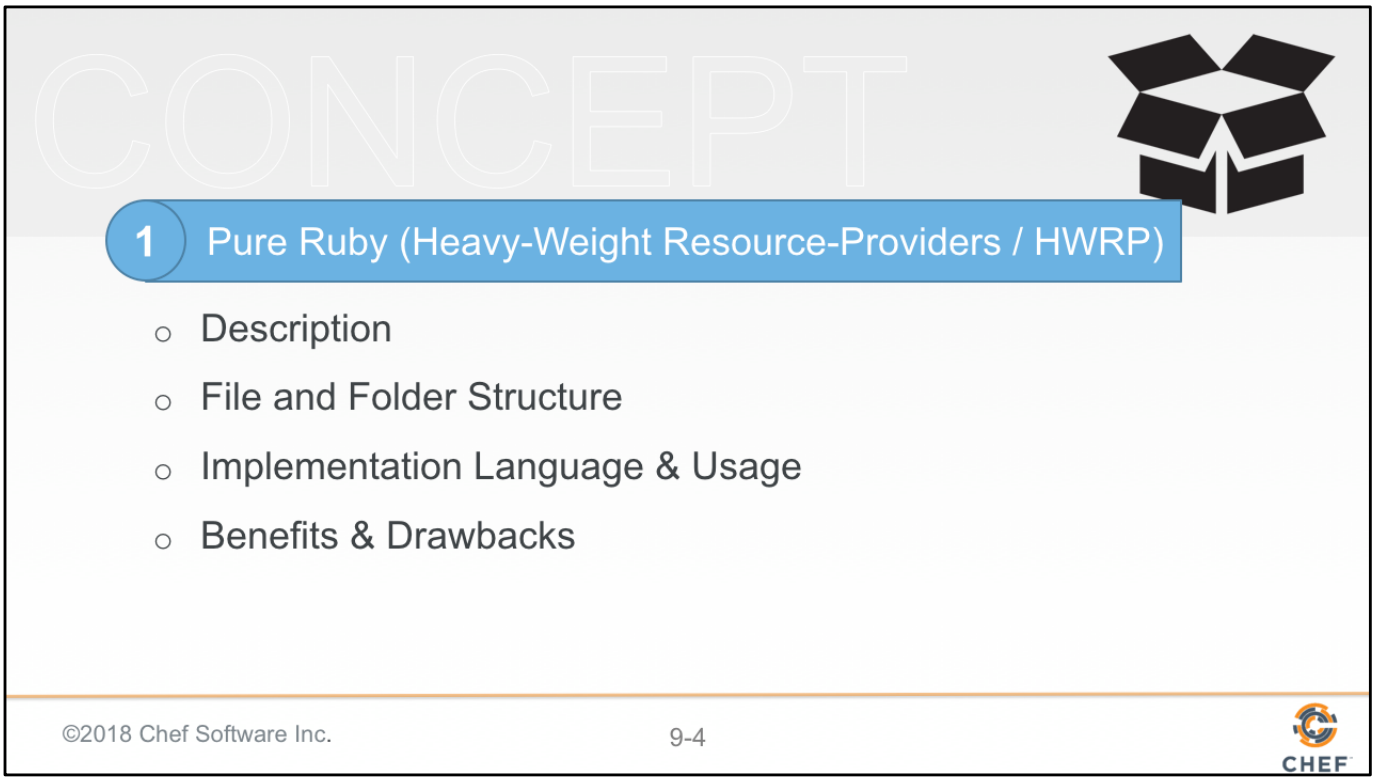
After completing this module, you should be able to:

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

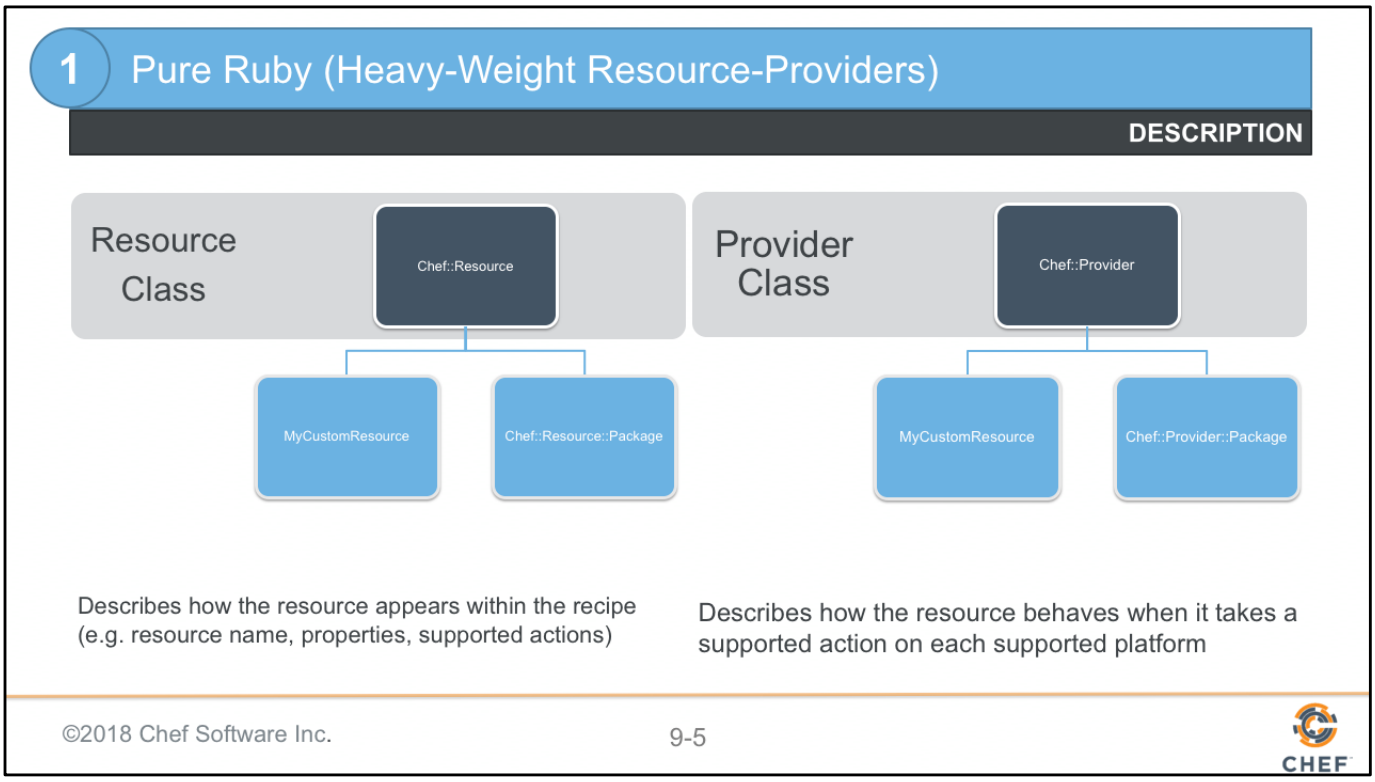
After completing this module you will be able to describe the differences between Custom Resources, Definitions, Heavy-Weight Resource Providers and Light-Weight Resource Providers.



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.

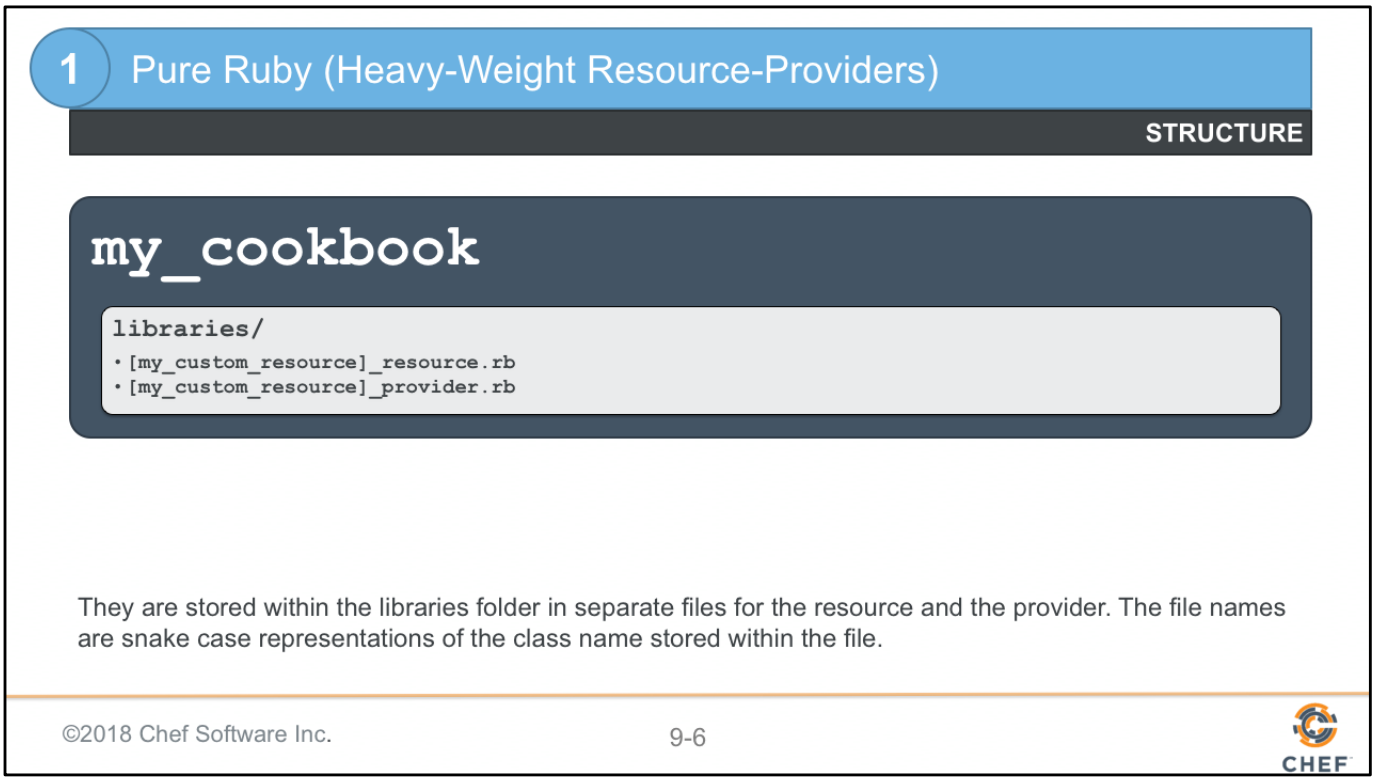


I will provide a description of each, explain the files and folder structure, take a quick look at how each is implemented, and then talk about any requirements or limitations when pursing this implementation choice.



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.



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

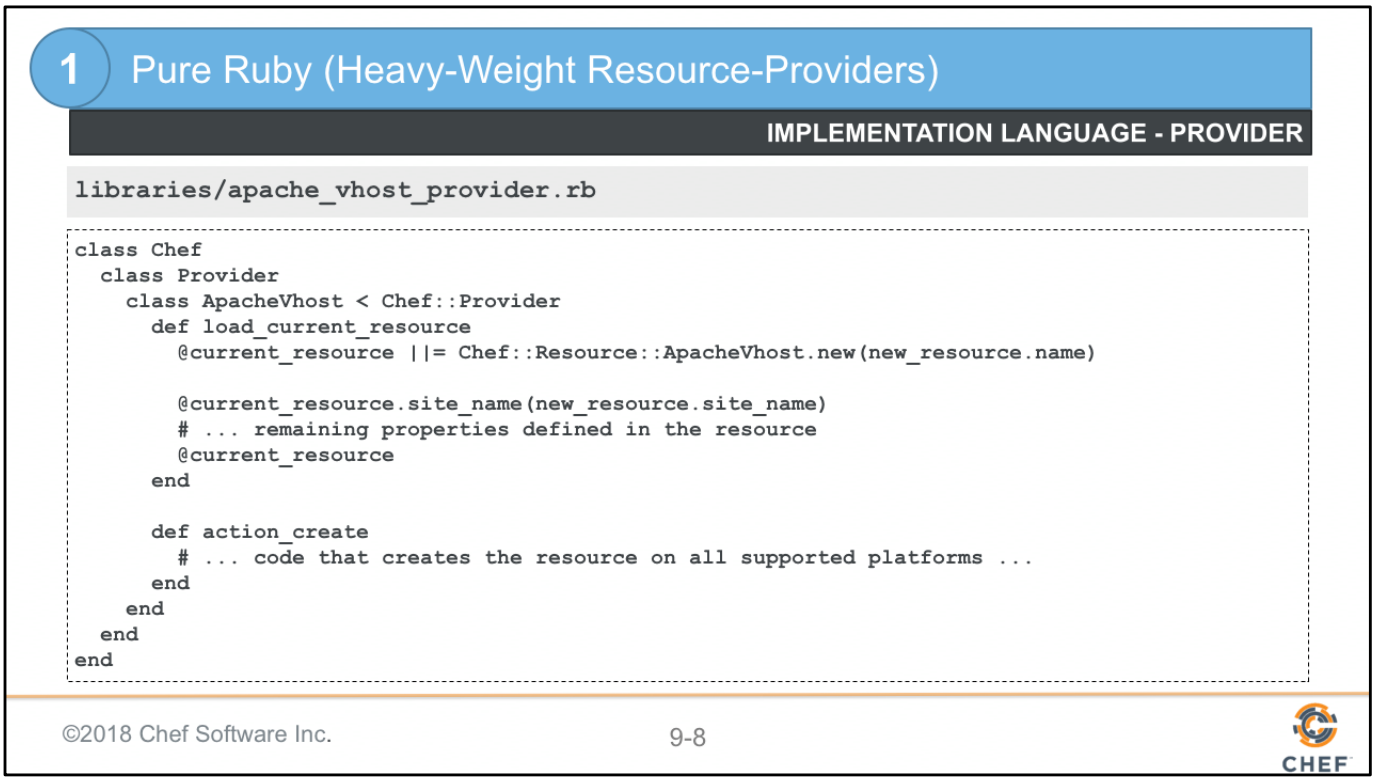
libraries/apache_vhost_resource.rb

```
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
```

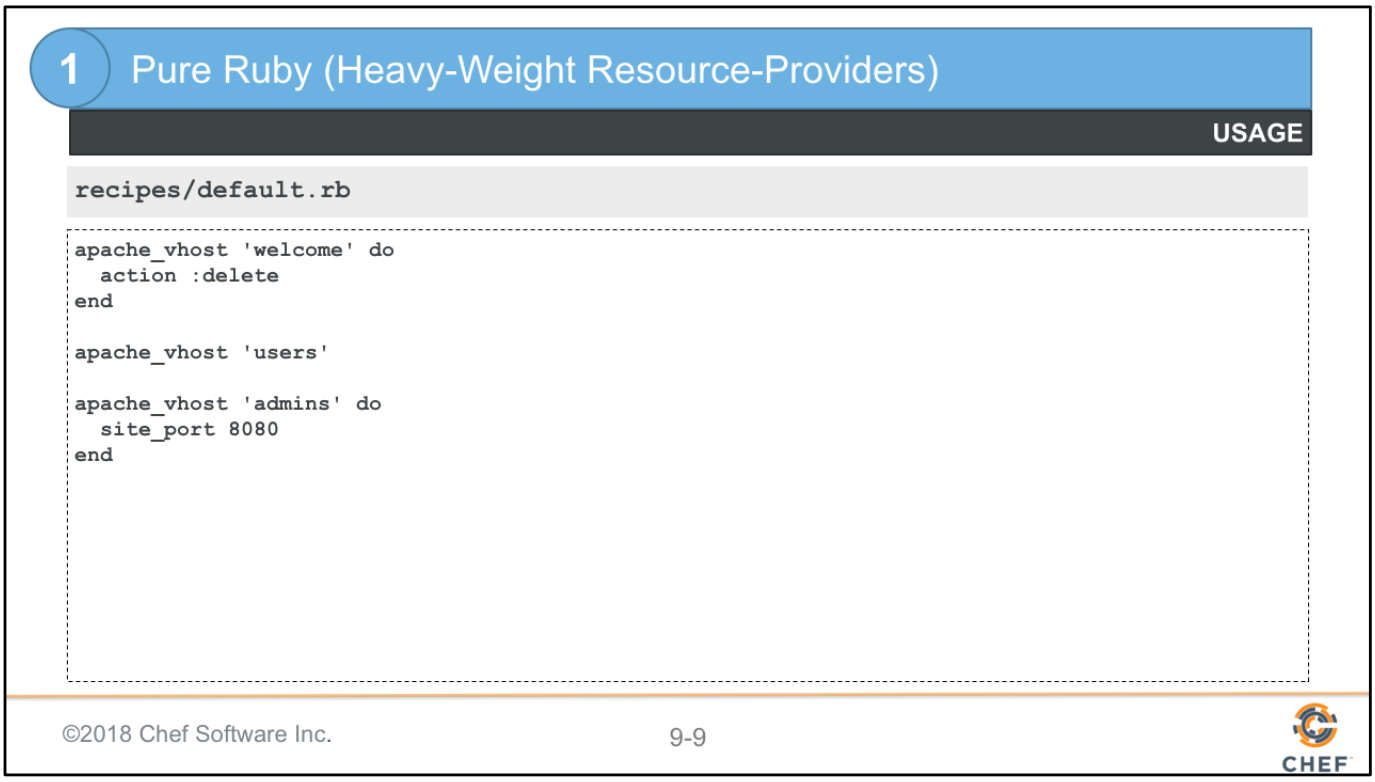

CHEF

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.

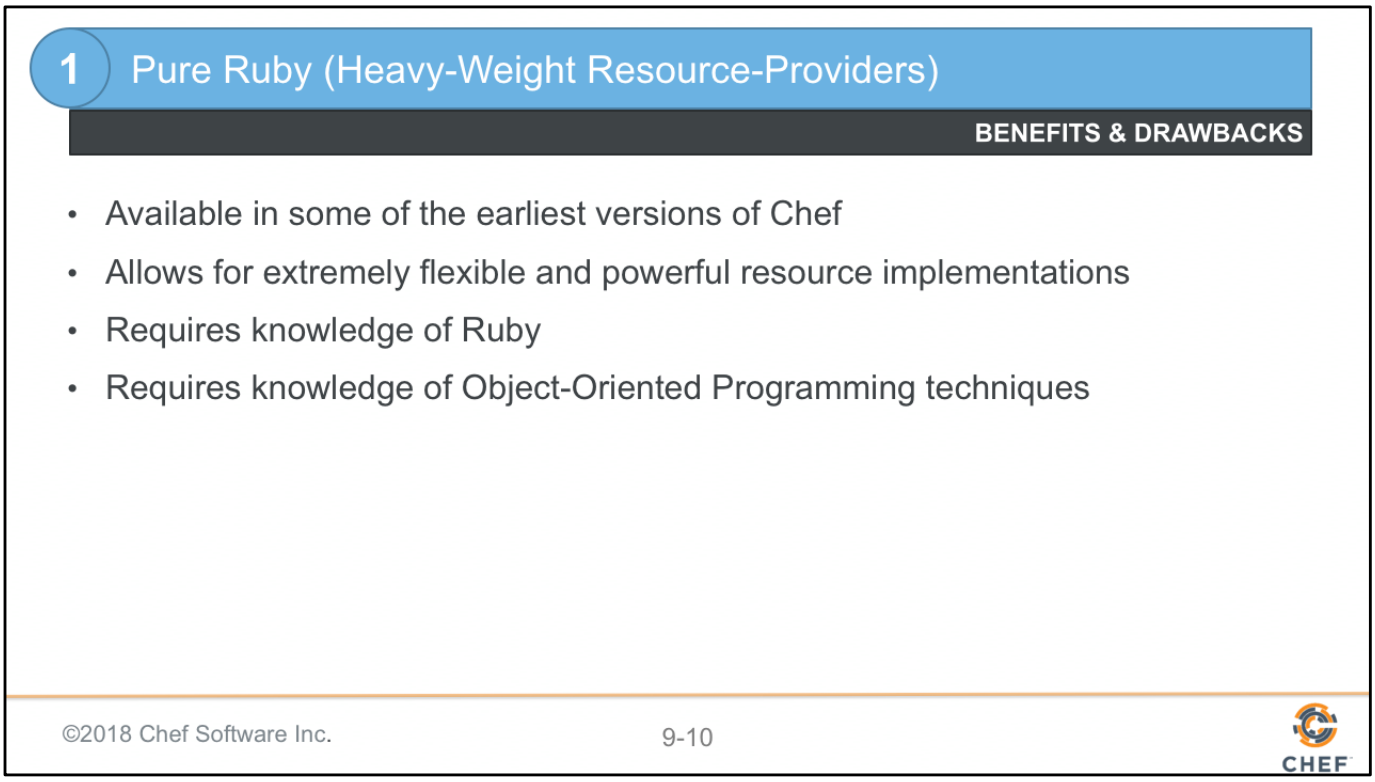


When defining the provider for a Heavy-Weight Resource-Provider you sub-class the Chef Provider class. The initialize method does not have to be overridden. The load_current_resource method must be overridden and is where the configuration from the resource is created and configured for use in each of 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.

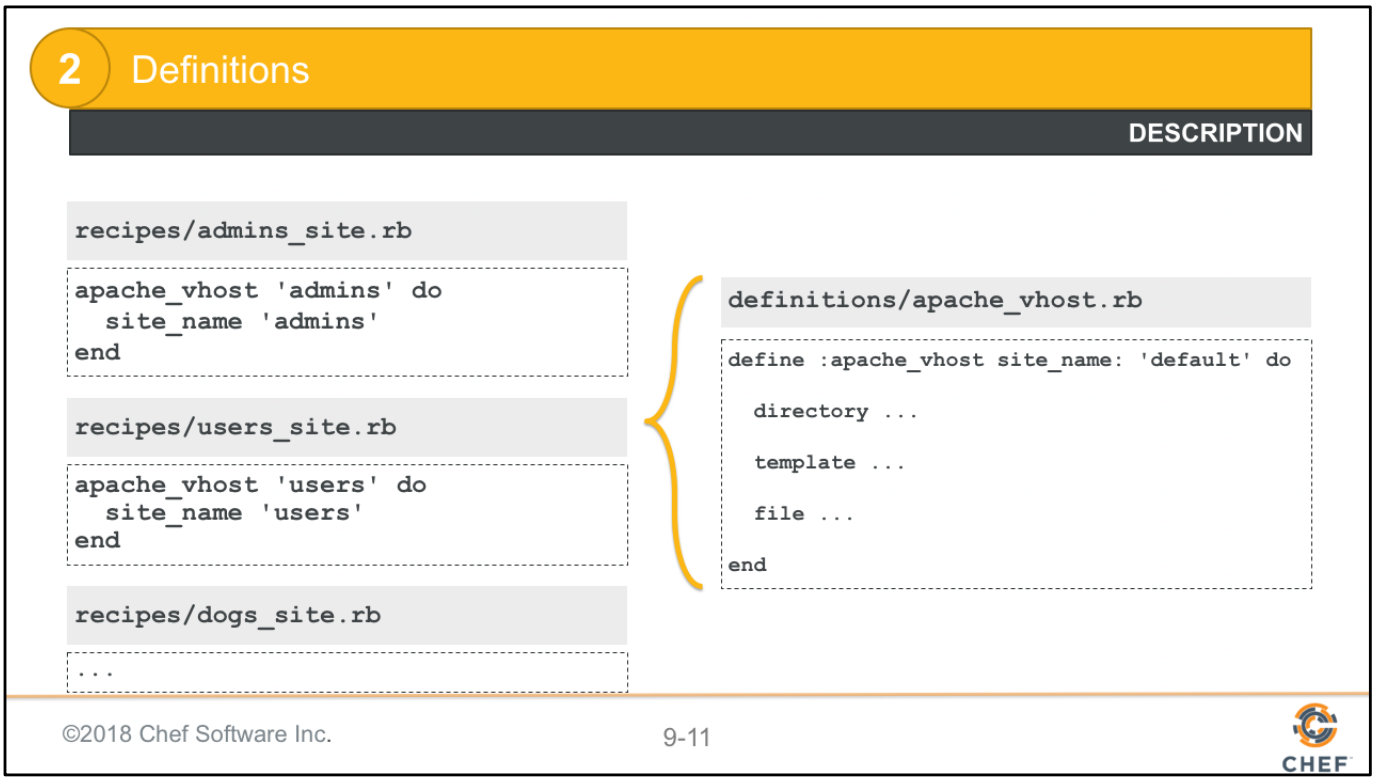
Chef provides additional helpers to allow you to shell out to perform operations on the system. You also have the entire Ruby language and any gems that might be packaged with the Chef DK (or you have added to Chef DK) at your disposal.



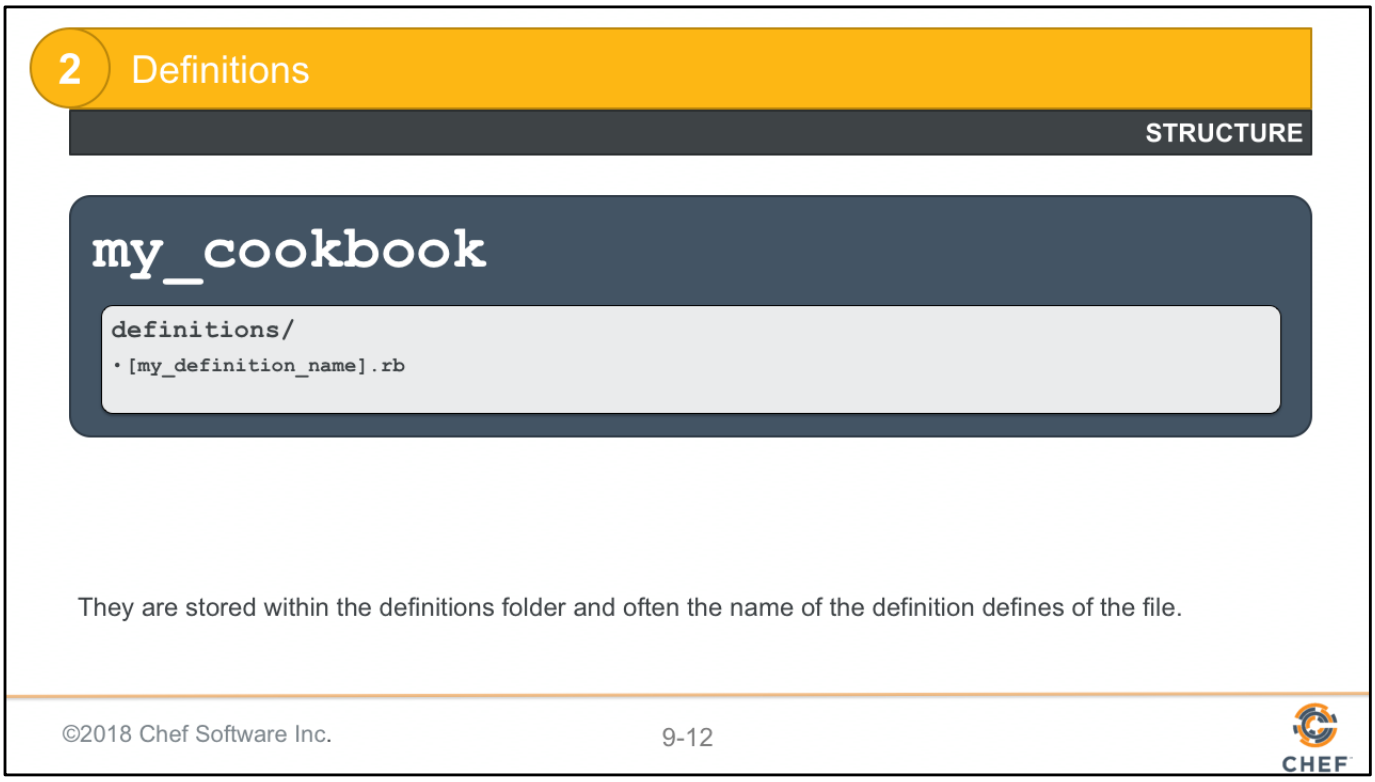
The resource would now be available within any recipe defined in this cookbook or any cookbook that adds this cookbook as a dependency. Here in this example recipe the resources delete and creates apache sites. All three of the resources rely on the site name attribute being tied to the name provided 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.



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.

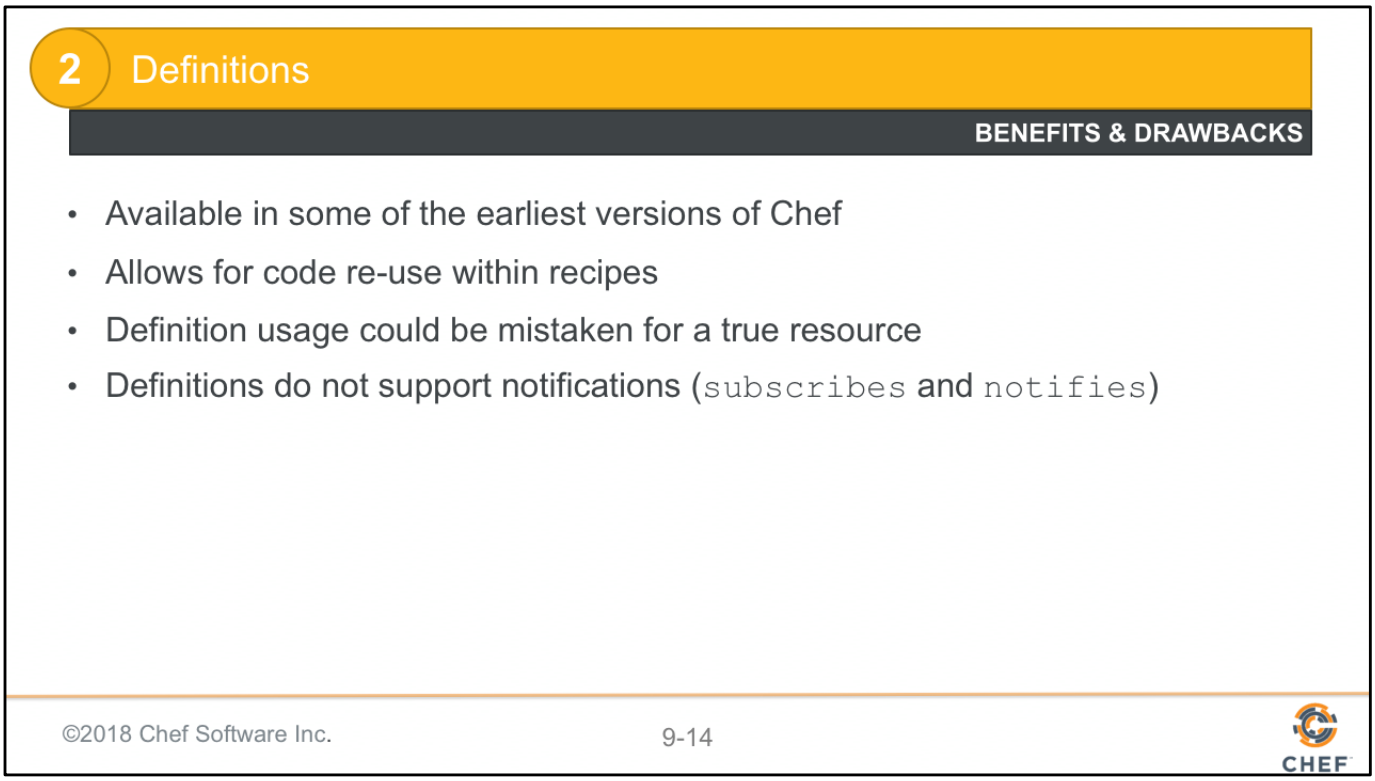


Definitions behaves like a compile time macro that is reusable across recipes. Macros all you to write a small amount of code that expands out into the contents of the definition wherever it is found within the recipes. With a definition you give it a name, provide parameters, and specify a block of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

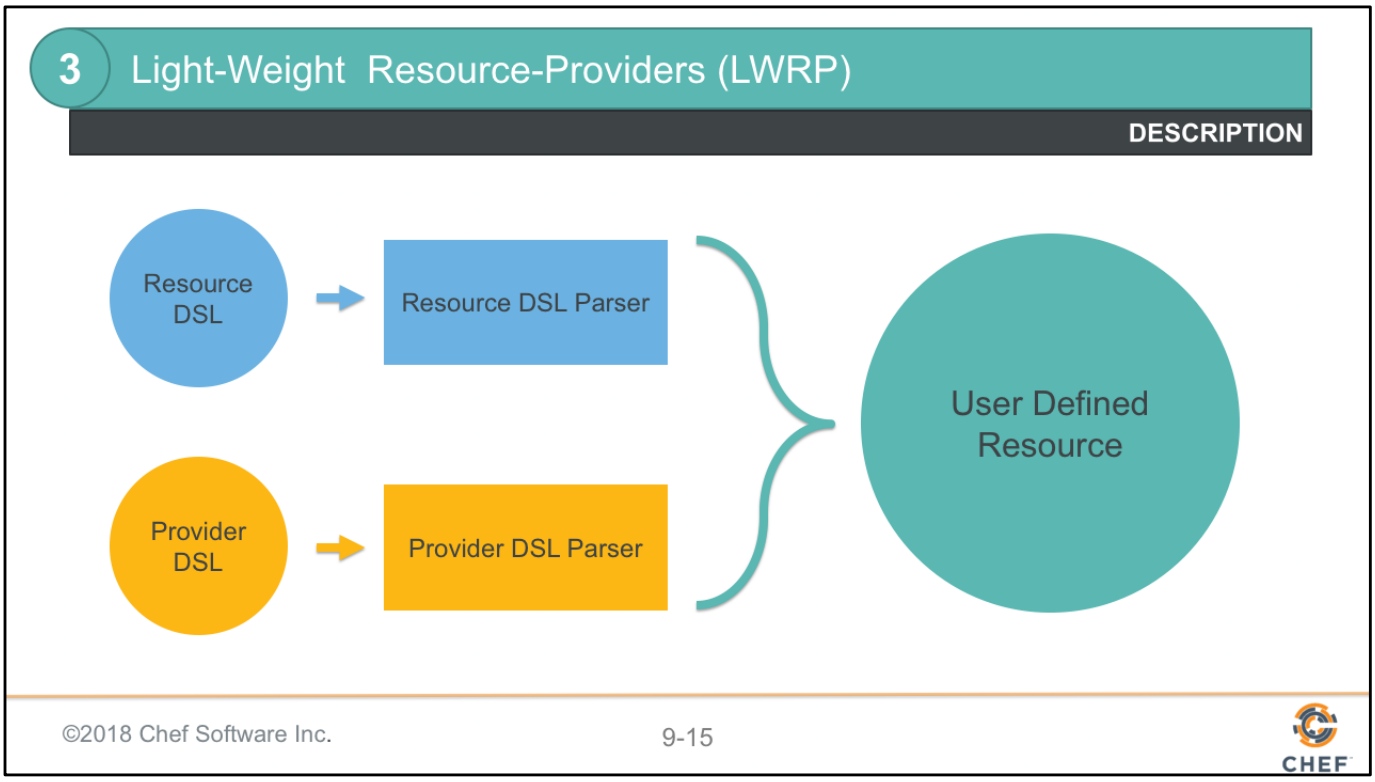


The code that defines the definition is stored within a definitions directory in a Ruby file that is processed with the definition Domain Specific Language.

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`).

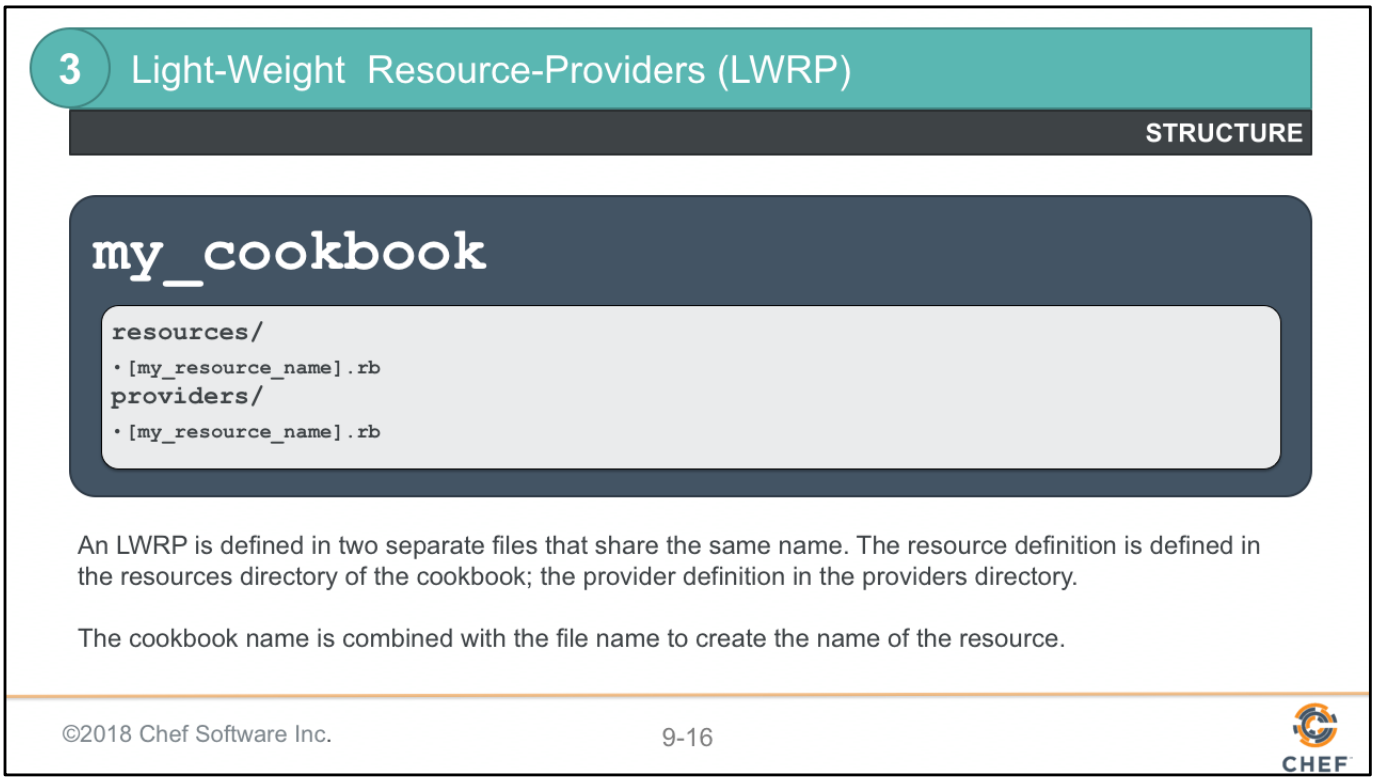


Definitions shipped in some of the earliest versions of Chef and are still supported today. However, as of Chef 12.5 it is strongly recommended that you choose a solution built with custom resources.



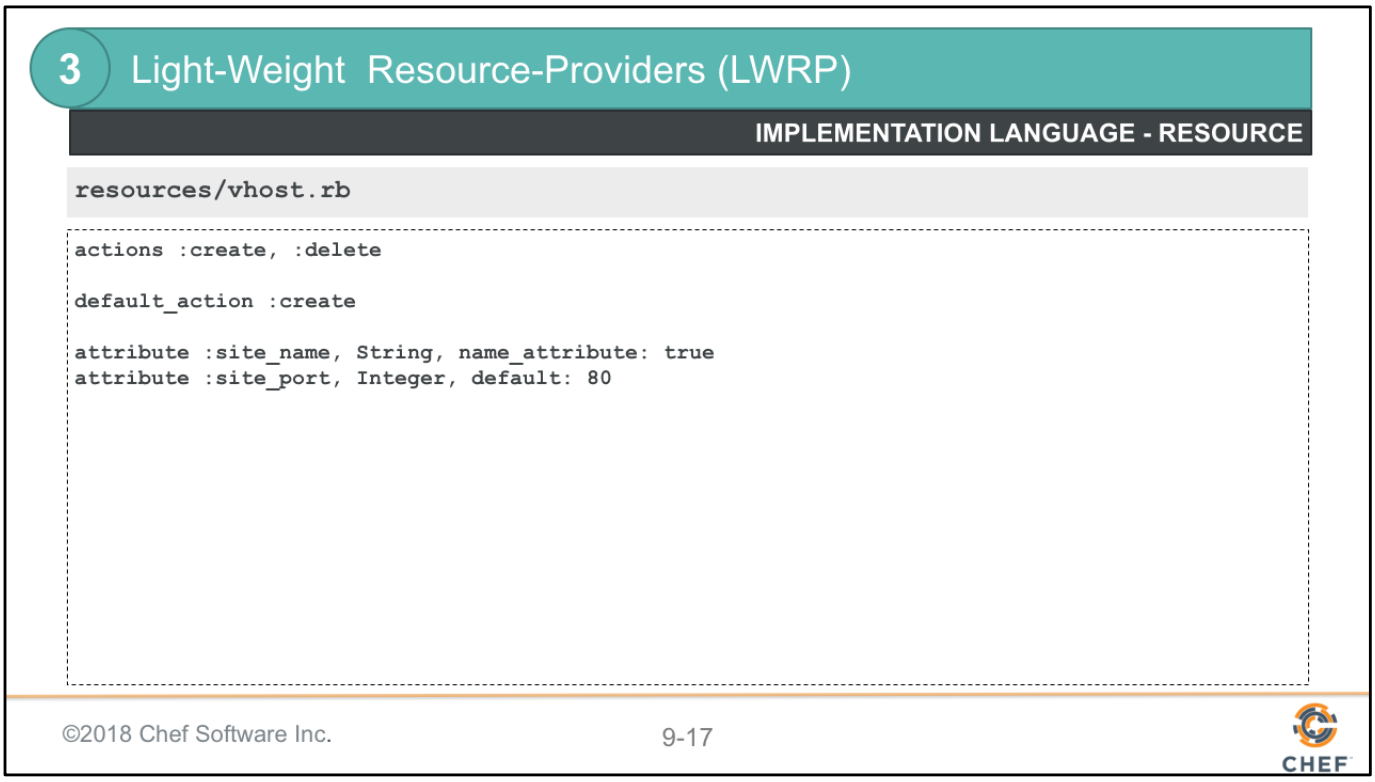
Light-Weight Resource-Provider, or LWRP, are Chef resources defined in two Domain Specific Languages (DSL) that allow you to create resources without having to understand the complexity presented by HWRP.

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.



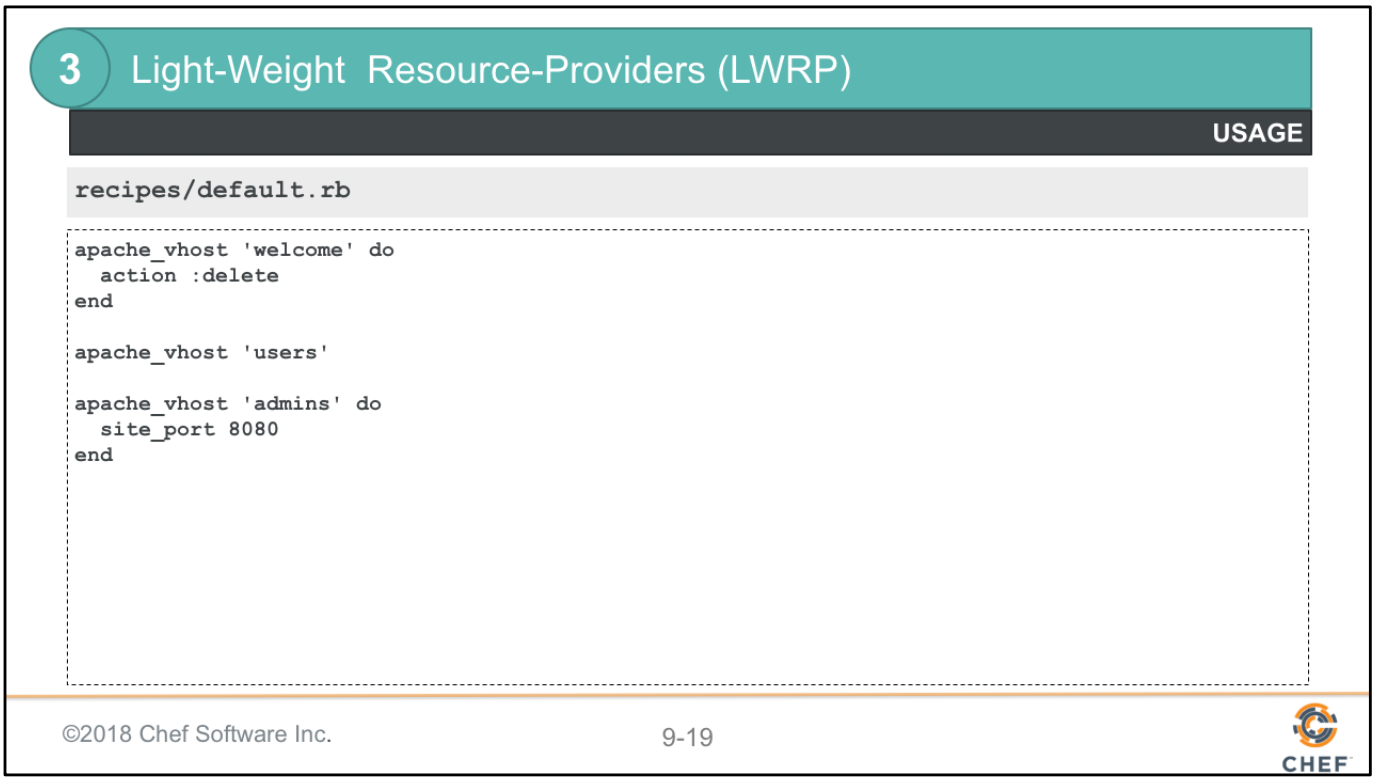
A single LWRP definition is defined in two separate files. The file is named exactly the same but one file resides in the 'resources' directory; the other in the 'providers' directory. Both of these files are parsed after the cookbook is synchronized and loaded. Each file's DSL is then converted into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



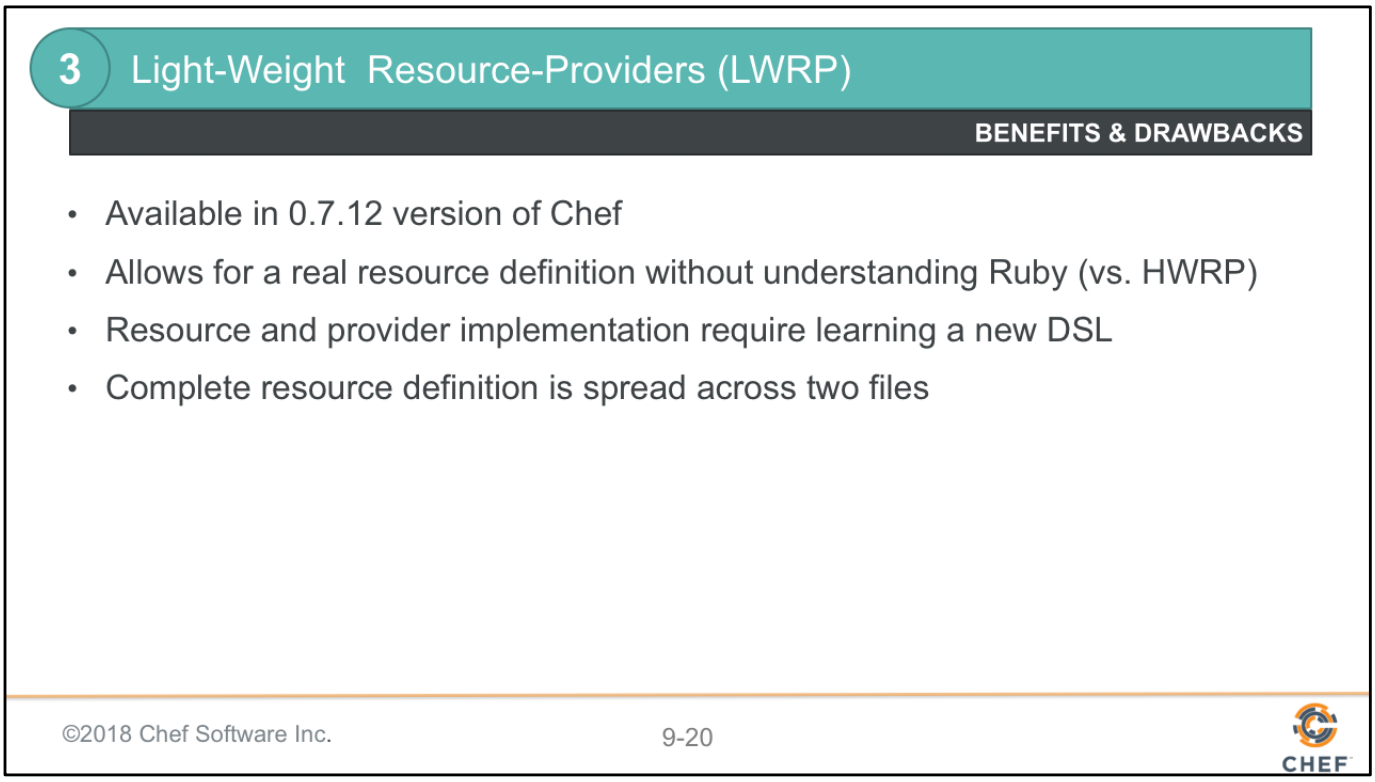
Within the resources file you specify the available actions, the default action, and the supported attributes that can be used when specifying the resource.

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'.

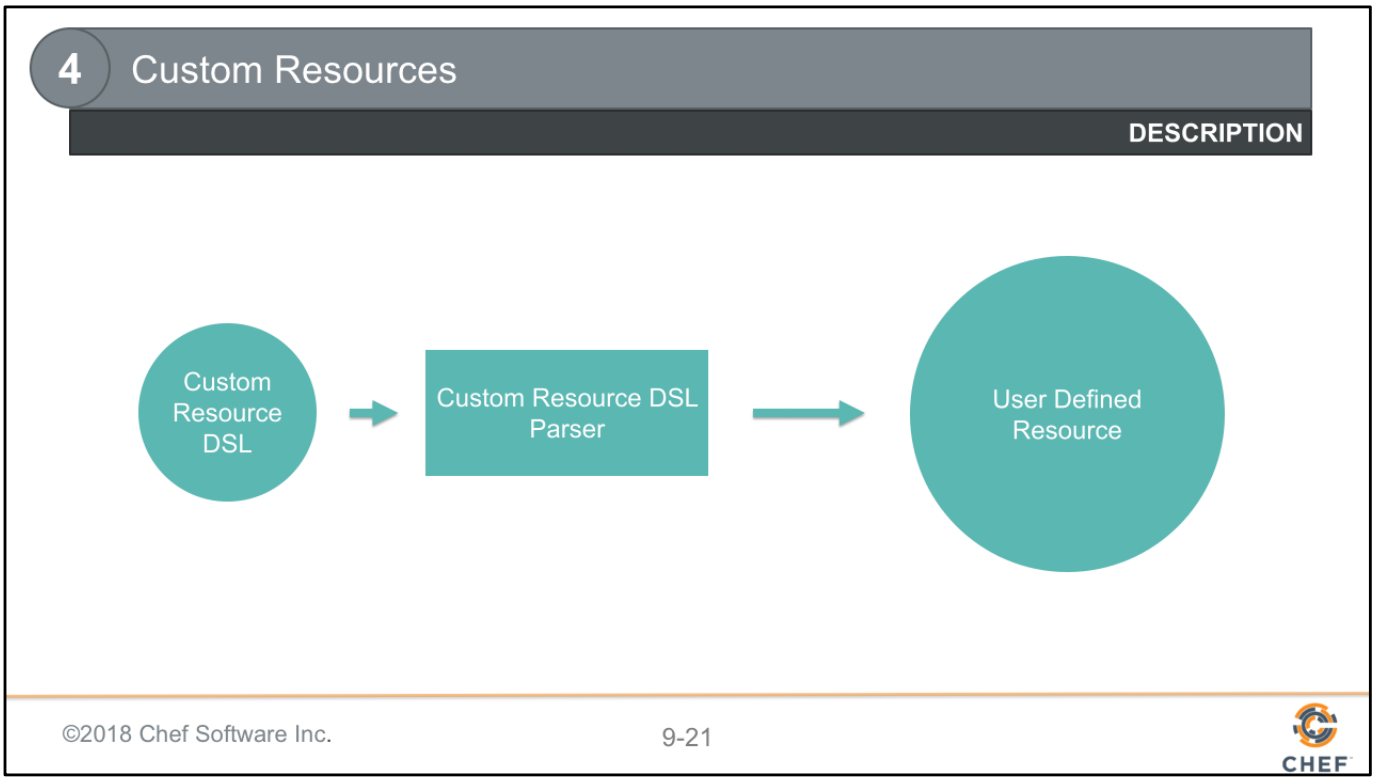


The name of the cookbook is combined with the name of the resource/provider file name with an underscore to create the user defined resource. This was explicitly defined in the HWRP but is automatically generated.

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

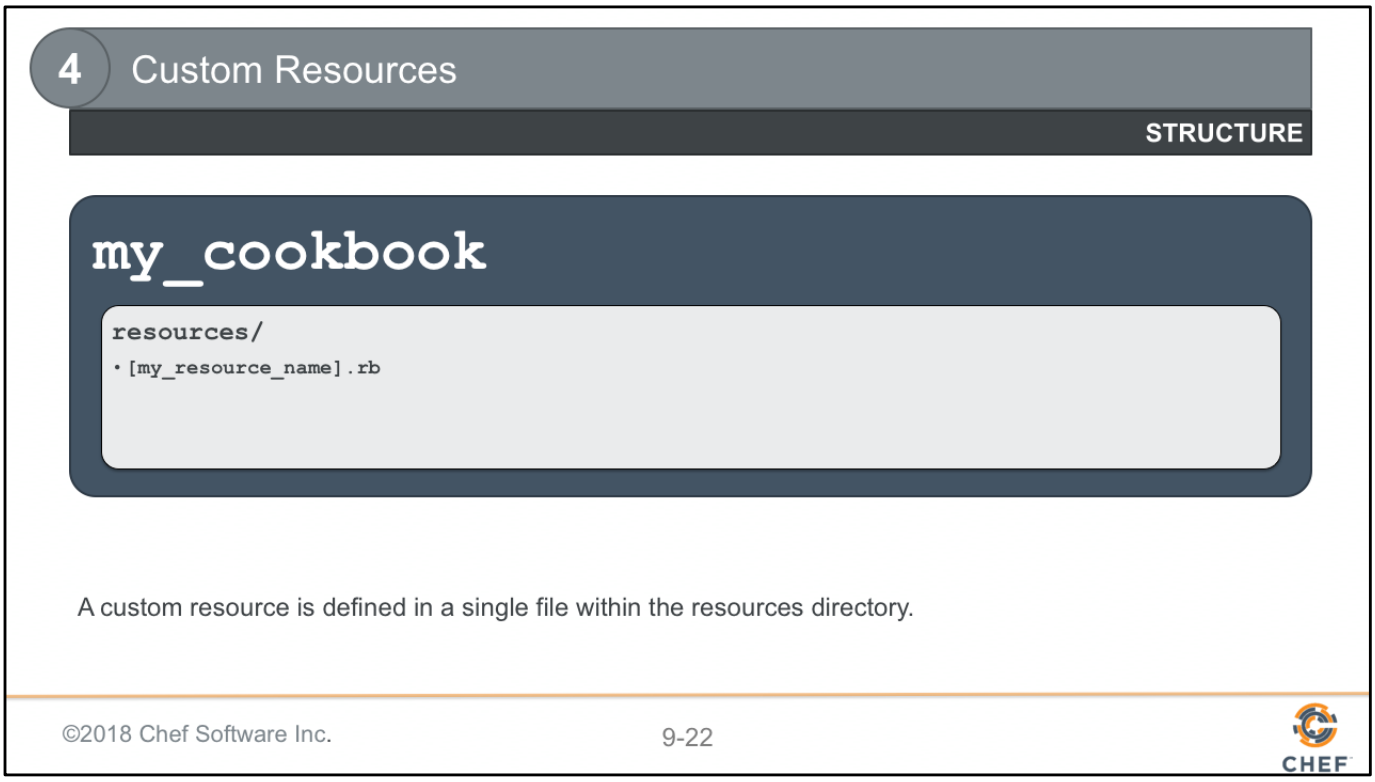


Implementing resources with LWRP is not the favored way to develop a resource in later versions of Chef (Chef 12.5). However, they are still in wide use within older cookbooks like those found within the Chef Supermarket.

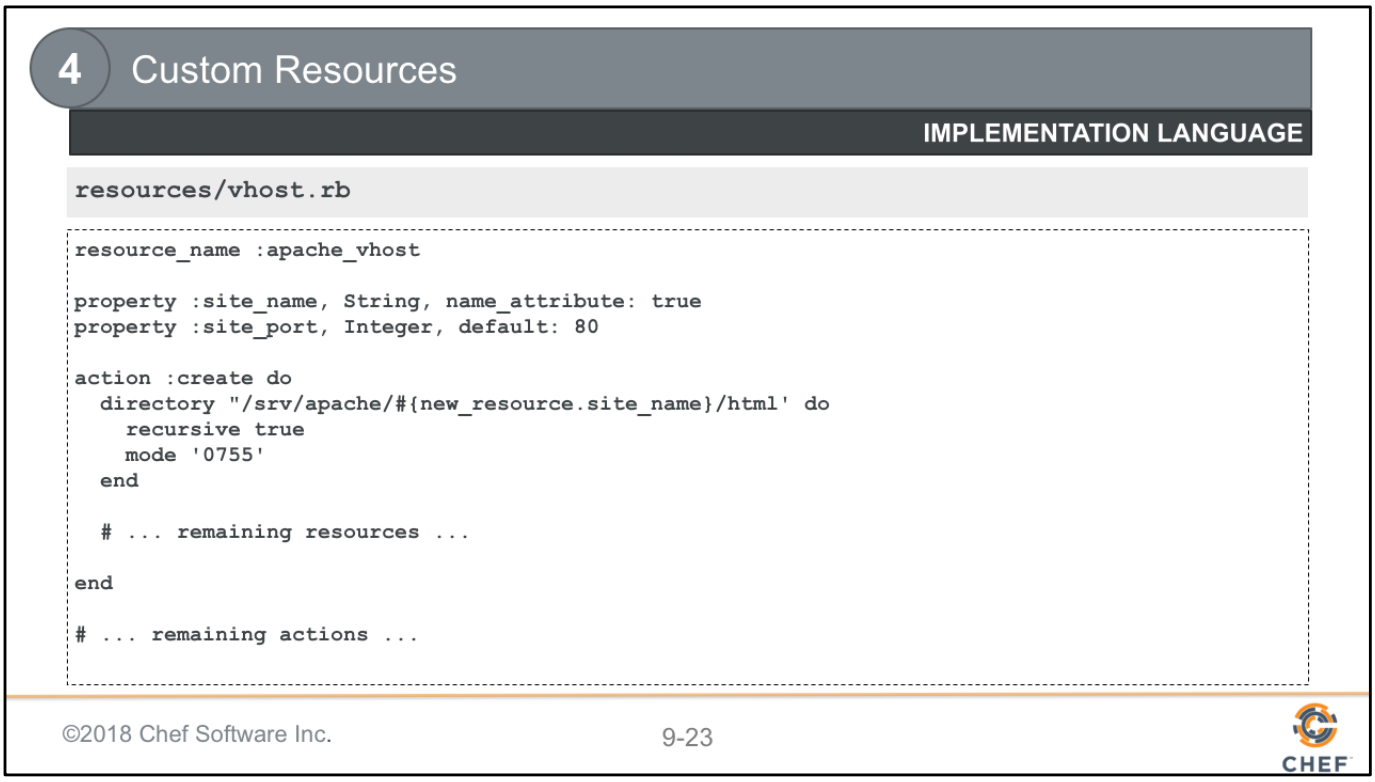


Custom Resources are Chef resources defined in a Domain Specific Language (DSL) that allow you to create resources without having to understand the complexity presented by HWRP. At its core it is a simplification of the work done with LWRP.

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.



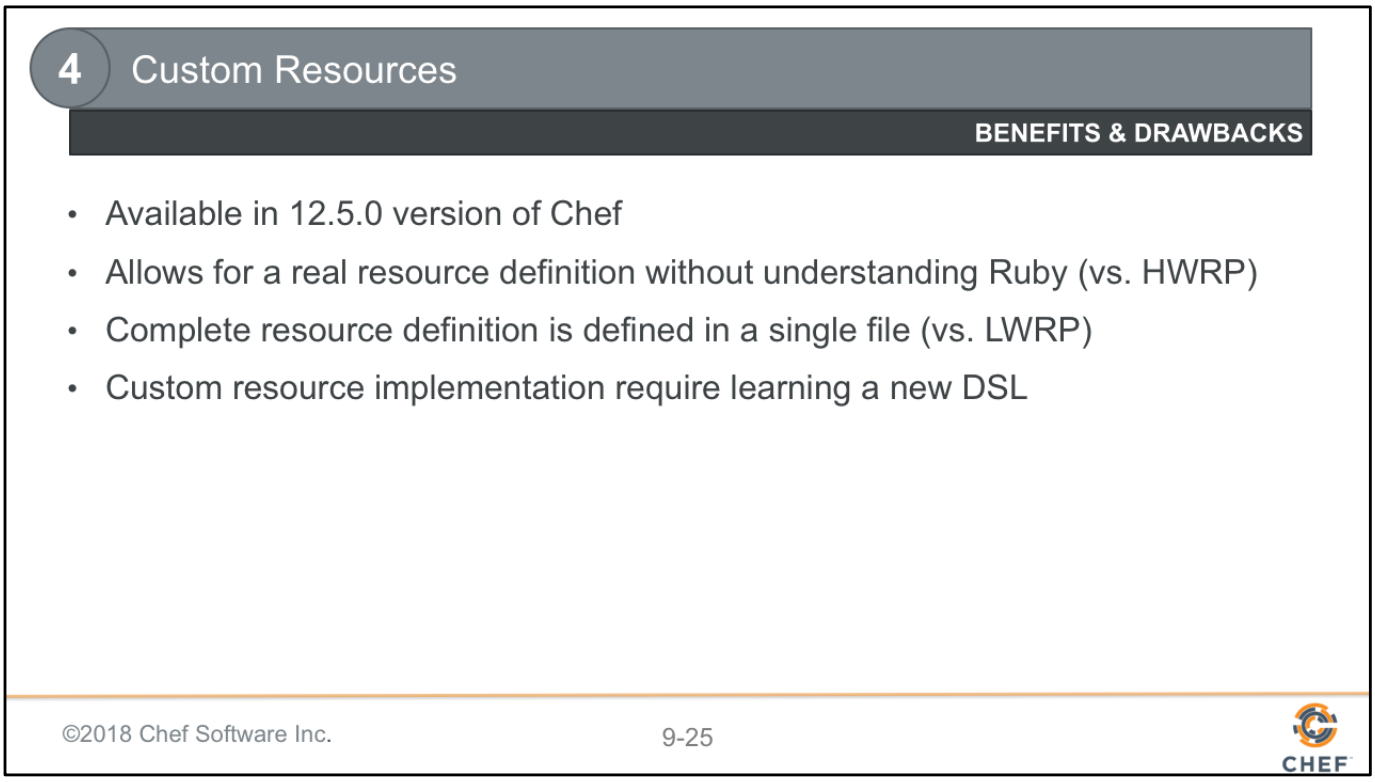
Within the file in the 'resources' directory you define the interface and the implementation for the custom resource. This is written in a custom resource DSL where you can specify the name of the resource, the default action, the properties that may be set, and all the actions that the resource supports.



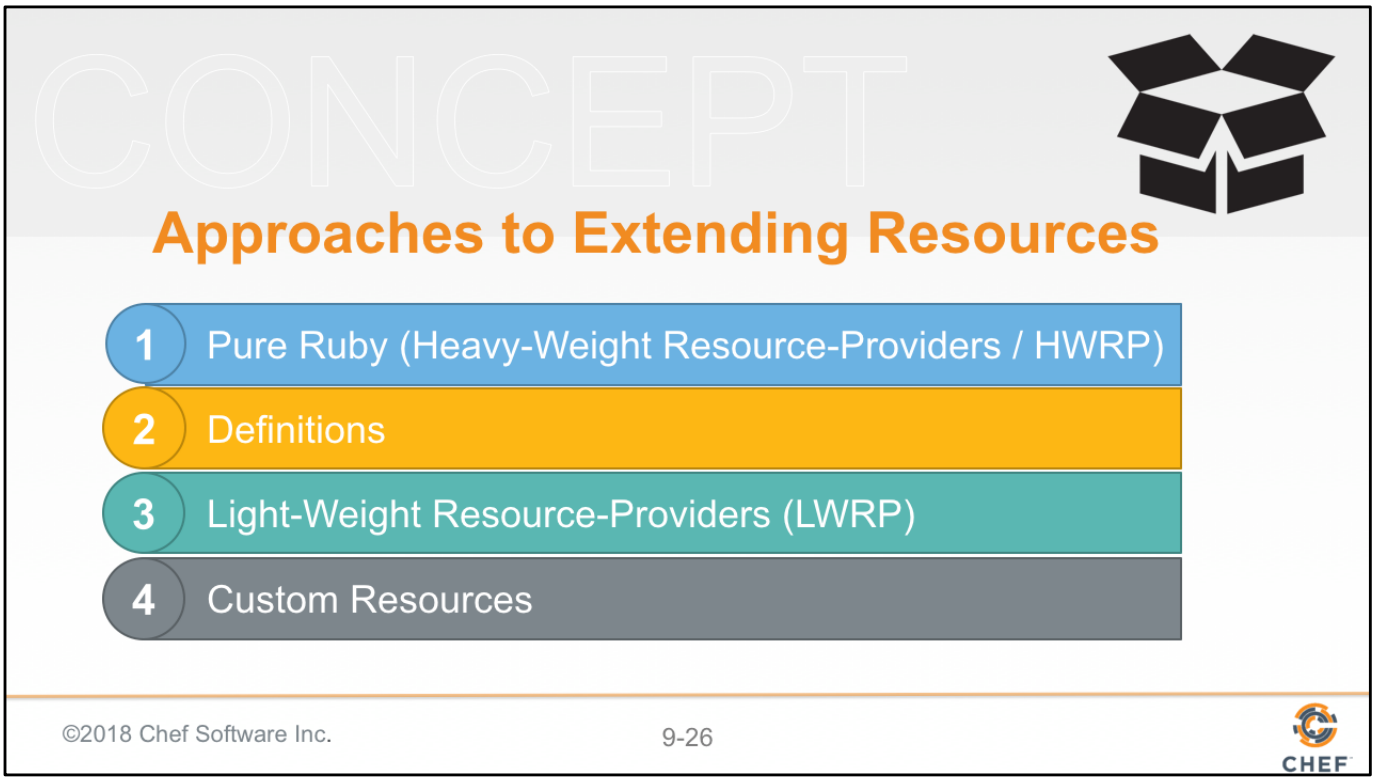
The custom resource implementation is similar to the LWRP except all of the details that describe the resource are combined into a single file. The custom resource DSL is similar to one defined for the LWRP resource and LWRP provider DSL. It is an evolution of the LWRP implementation with some minor 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.

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

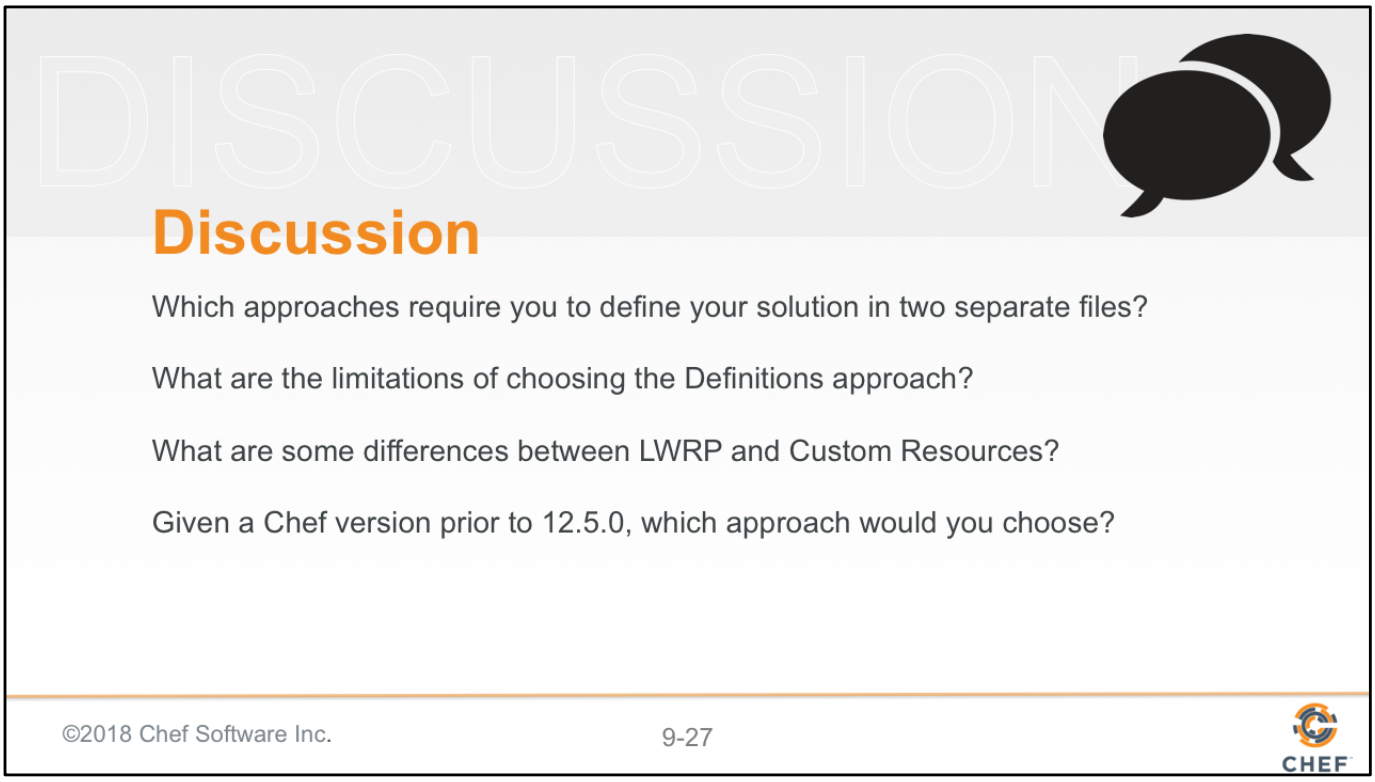
The default action is determined by the first action listed in the custom resource definition.



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.

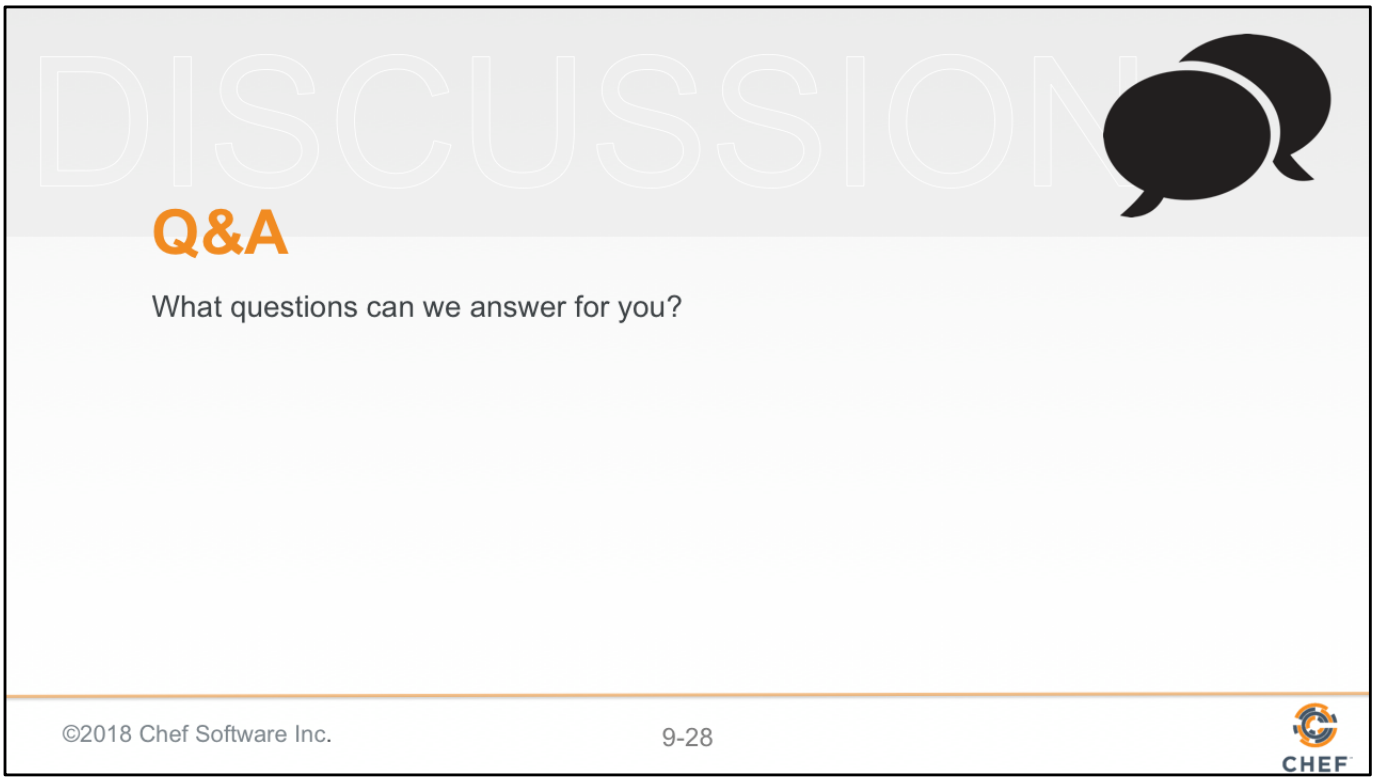


As you can see there are more than a few ways to extend Chef and create a resource or resource-like implementation within your recipes.



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.



What questions can we answer for you?


CHEF™