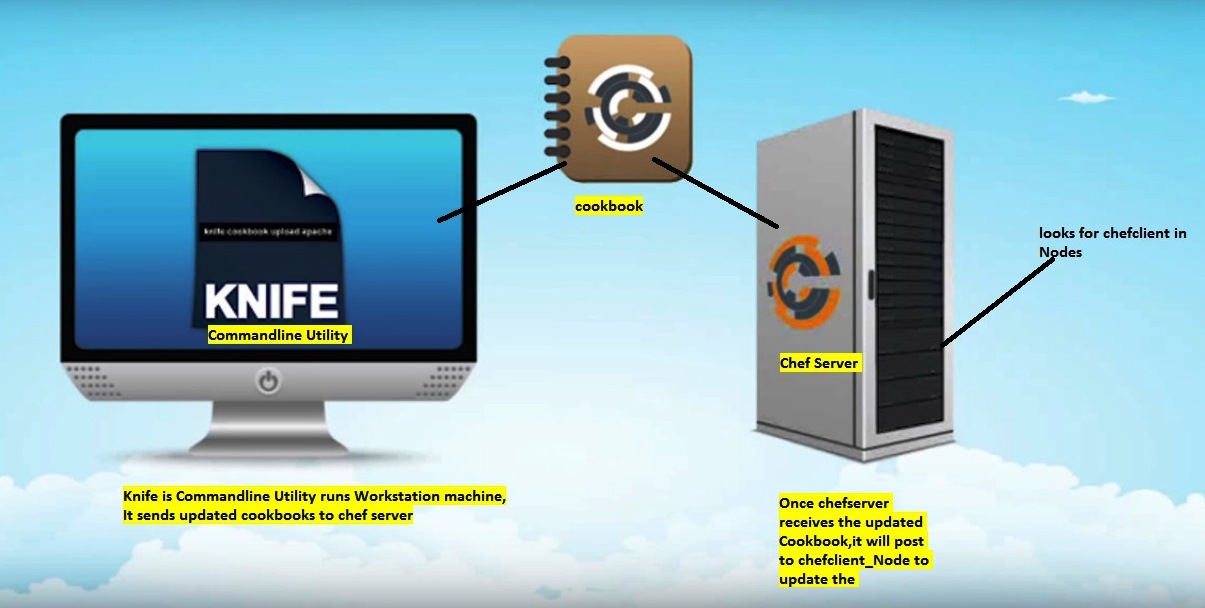
**Cookbooks**

A cookbook is the fundamental unit of configuration and policy distribution. A cookbook defines a scenario and contains everything that is required to support that scenario:

* Recipes that specify the resources to use and the order in which they are to be applied
* Attribute values
* File distributions
* Templates
* Extensions to Chef, such as custom resources and libraries

The chef-client uses Ruby as its reference language for creating cookbooks and defining recipes, with an extended DSL for specific resources. A reasonable set of resources are available to the chef-client, enough to support many of the most common infrastructure automation scenarios; however, this DSL can also be extended when additional resources and capabilities are required.



Cookbooks are comprised of the following components:

|  | **Description** |
| --- | --- |
|  | An attribute can be defined in a cookbook (or a recipe) and then used to override the default settings on a node. When a cookbook is loaded during a chef-client run, these attributes are compared to the attributes that are already present on the node. Attributes that are defined in attribute files are first loaded according to cookbook order. For each cookbook, attributes in the default.rb file are loaded first, and then additional attribute files (if present) are loaded in lexical sort order. When the cookbook attributes take precedence over the default attributes, the chef-client will apply those new settings and values during the chef-client run on the node. |
|  | A definition is code that is reused across recipes, similar to a compile-time macro. A definition is created using arbitrary code wrapped around built-in chef-client resources—**file**, **execute**, **template**, and so on—by declaring those resources into the definition as if they were declared in a recipe. A definition is then used in one (or more) recipes as if it were a resource.  Though a definition behaves like a resource, some key differences exist. A definition:   * Is not a resource or a custom resource * Is defined from within the /definitions directory of a cookbook * Is loaded before resources during the chef-client run; this ensures the definition is available to all of the resources that may need it * May not notify resources in the resource collection because a definition is loaded **before** the resource collection itself is created; however, a resource in a definition **may** notify a resource that exists within the same definition * Automatically supports why-run mode, unlike custom resources   Use a defintion when repeating patterns exist across resources and/or when a simple, direct approach is desired. There is no limit to the number of resources that may be included in a definition: use as many built-in chef-client resources as necessary. |
|  | Use the **cookbook\_file** resource to transfer files from a sub-directory of COOKBOOK\_NAME/files/ to a specified path located on a host that is running the chef-client. The file is selected according to file specificity, which allows different source files to be used based on the hostname, host platform (operating system, distro, or as appropriate), or platform version. Files that are located in the COOKBOOK\_NAME/files/default sub-directory may be used on any platform. |
|  | A library allows arbitrary Ruby code to be included in a cookbook, either as a way of extending the classes that are built-in to the chef-client—Chef::Recipe, for example—or for implementing entirely new functionality, similar to a mixin in Ruby. A library file is a Ruby file that is located within a cookbook’s /libraries directory. Because a library is built using Ruby, anything that can be done with Ruby can be done in a library file. |
|  | Every cookbook requires a small amount of metadata. A file named metadata.rb is located at the top of every cookbook directory structure. The contents of the metadata.rb file provides hints to the Chef server to help ensure that cookbooks are deployed to each node correctly. |
|  | A recipe is the most fundamental configuration element within the organization. A recipe:   * Is authored using Ruby, which is a programming language designed to read and behave in a predictable manner * Is mostly a collection of resources, defined using patterns (resource names, attribute-value pairs, and actions); helper code is added around this using Ruby, when needed * Must define everything that is required to configure part of a system * Must be stored in a cookbook * May be included in a recipe * May use the results of a search query and read the contents of a data bag (including an encrypted data bag) * May have a dependency on one (or more) recipes * May tag a node to facilitate the creation of arbitrary groupings * Must be added to a run-list before it can be used by the chef-client * Is always executed in the same order as listed in a run-list   The chef-client will run a recipe only when asked. When the chef-client runs the same recipe more than once, the results will be the same system state each time. When a recipe is run against a system, but nothing has changed on either the system or in the recipe, the chef-client won’t change anything.  The Recipe DSL is a Ruby DSL that is primarily used to declare resources from within a recipe. The Recipe DSL also helps ensure that recipes interact with nodes (and node properties) in the desired manner. Most of the methods in the Recipe DSL are used to find a specific parameter and then tell the chef-client what action(s) to take, based on whether that parameter is present on a node. |
|  | A resource is a statement of configuration policy that:   * Describes the desired state for a configuration item * Declares the steps needed to bring that item to the desired state * Specifies a resource type—such as package, template, or service * Lists additional details (also known as resource properties), as necessary * Are grouped into recipes, which describe working configurations   Where a resource represents a piece of the system (and its desired state), a provider defines the steps that are needed to bring that piece of the system from its current state into the desired state.  Chef has many built-in resources that cover all of the most common actions across all of the most common platforms. You can build your own resources to handle any situation that isn’t covered by a built-in resource. |
|  | A cookbook template is an Embedded Ruby (ERB) template that is used to dynamically generate static text files. Templates may contain Ruby expressions and statements, and are a great way to manage configuration files. Use the **template** resource to add cookbook templates to recipes; place the corresponding Embedded Ruby (ERB) template file in a cookbook’s /templates directory. |
|  | Testing cookbooks improves the quality of those cookbooks by ensuring they are doing what they are supposed to do and that they are authored in a consistent manner. Unit and integration testing validates the recipes in cookbooks. Syntax testing—often called linting—validates the quality of the code itself. The following tools are popular tools used for testing Chef recipes: Test Kitchen, ChefSpec, and Foodcritic. |