**Architecture of Chef**

Chef is a powerful automation platform that transforms infrastructure into code. Whether you’re operating in the cloud, on-premises, or in a hybrid environment, Chef automates how infrastructure is configured, deployed, and managed across your network, no matter its size.

This diagram shows how you develop, test, and deploy your Chef code.



* The **Chef DK workstation** is the location where users interact with Chef. On the workstation users author and test [cookbooks](https://docs.chef.io/cookbooks.html) using tools such as [Test Kitchen](https://docs.chef.io/kitchen.html) and interact with the Chef server using the [knife](https://docs.chef.io/knife.html) and [chef](https://docs.chef.io/ctl_chef.html) command line tools.
* **Chef client nodes** are the machines that are managed by Chef. The Chef client is installed on each node and is used to configure the node to its desired state.
* The **Chef server** acts as [a hub for configuration data](https://docs.chef.io/server_components.html). The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by Chef. Nodes use the Chef client to ask the Chef server for configuration details, such as recipes, templates, and file distributions.

**Install Chef**

Install Chef on a CentOS machine. Installing Chef is like a cake walk, you just have to execute few commands.

Chef has three major components:

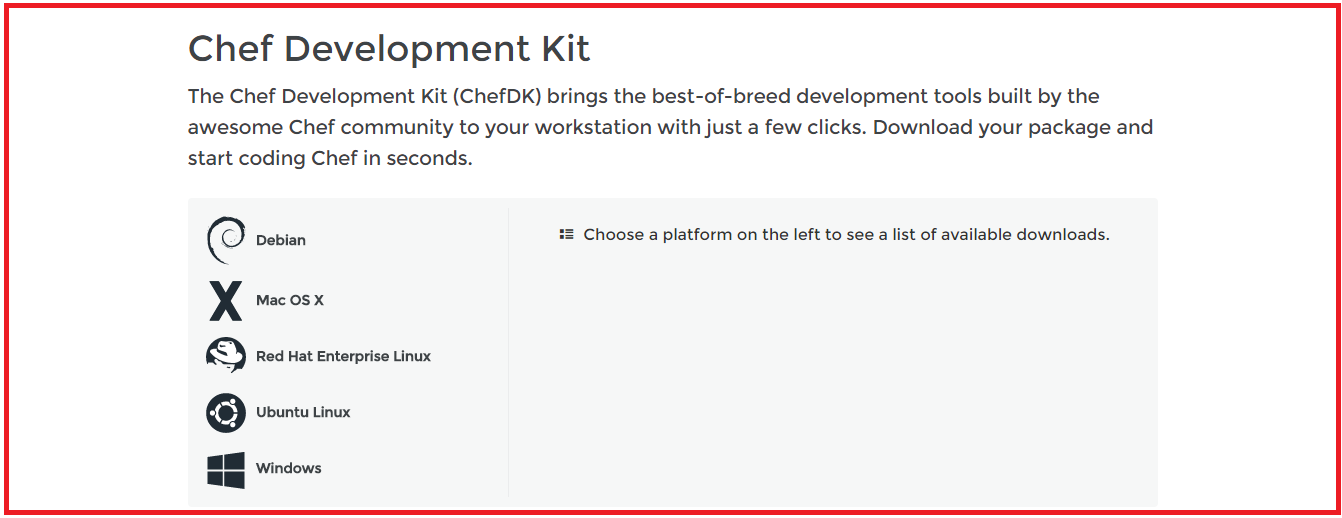
1. Workstation
2. Server
3. Nodes

Following are the steps to install Chef:

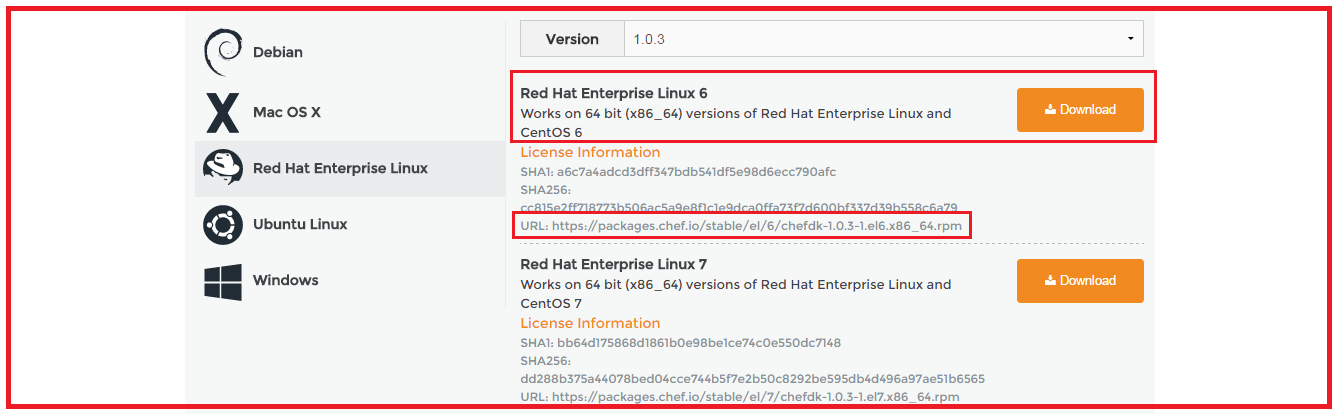
1. Install Chef DK (Development Kit) on Chef Workstation
2. Setup a Chef Server
3. Create a Recipe or a Cookbook / download a Cookbook from Chef Supermarket in Workstation
4. Upload a Cookbook on the Chef Server
5. Connect A Node To The Chef Server
6. Deploy the Cookbook from the Server to the Node

## ****1. Install Chef DK (Development Kit)****

In my Chef Workstation I will install Chef DK. Chef DK is a package that contains all the development tools that you will need when coding Chef. Here is the link to download [***Chef DK***](https://downloads.chef.io/chef-dk/).



Here, choose the operating system that you are using. I am using CentOS 6.8. So, I will click on **Red Hat Enterprise Linux**.



Copy the link according to the version of CentOS that you are using. I am using CentOS 6, as you can see that I have highlighted in the above screenshot.

Go to your Workstation terminal and download the Chef DK by using wget command and paste the link.

**Execute this command:**

|  |  |
| --- | --- |
| 1 | wget https://packages.chef.io/stable/el/6/chefdk-1.0.3-1.el6.x86\_64.rpm |

The package is now downloaded. Now I will install this package using rpm.

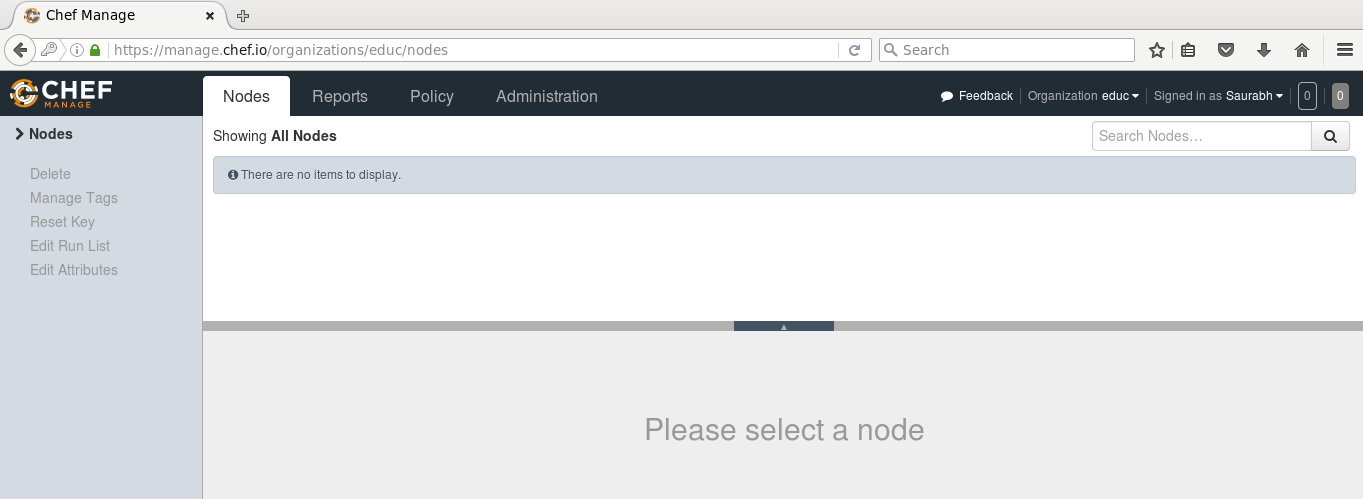
**Execute this:**

|  |  |
| --- | --- |
| 1 | rpm -ivh chefdk-1.0.3-1.el6.x86\_64.rpm |

## ****2. Setup Chef Server****

I will use the hosted version of Chef Server on the cloud but you can use a physical machine as well. This Chef-Server is present at [**manage.chef.io**](http://manage.chef.io/)

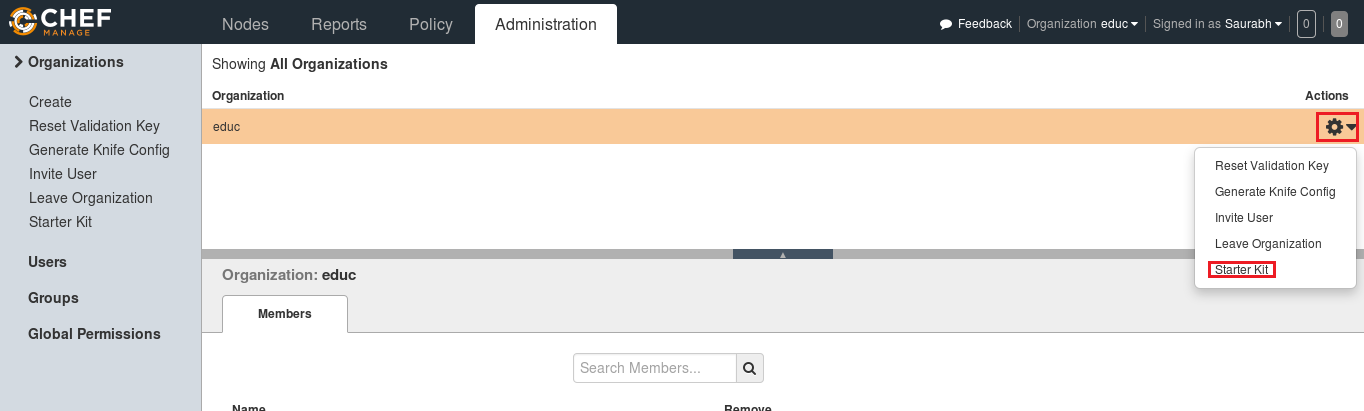
Over here, create an account if you don’t have one. Once you have created an account, sign-in with your login credentials.



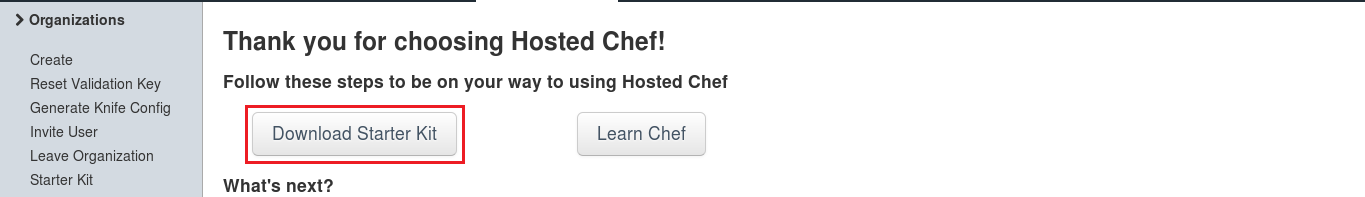
This is how Chef Server looks like.

If you are signing in for the first time, the very first thing that you will be doing is creating an organization. Organization is basically a group of Machines that you will be managing with the Chef Server.

First, I will go to the administration tab. Over there, I have already created an organization called edu. So I need to download the starter kit in my Workstation. This starter kit will help you to push files from the Workstation to the Chef Server. Click on the settings icon on the right hand side and click on Starter Kit.



Now you will get an option to download the Starter Kit. Just click on it to download the Starter Kit zip file.



Move this file to your root directory. Now unzip this file by using unzip command in your terminal. You will notice that it includes a directory called chef-repo.

**Execute this:**

|  |  |
| --- | --- |
| 1 | unzip chef-starter.zip |

Now move this starter kit to the cookbook directory in chef-repo directory.

**Execute this:**

|  |  |
| --- | --- |
| 1 | mv starter /root/chef-repo/cookbooks |

## ****3. Download A Cookbook From Chef Supermarket In Workstation****

Chef Cookbooks are available in the Cookbook Supermarket, we can go to the Chef Supermarket. Download the required Cookbooks from [***supermarket.chef.io***](http://supermarket.chef.io/). I’m downloading one of the Cookbook to install Apache from there.

**Execute this:**

|  |  |
| --- | --- |
| 1  2 | cd chef-repo  knife cookbook site download learn\_chef\_httpd |

There is Tar ball downloaded for the Apache Cookbook. Now, I will extract the contents from this downloaded Tar file. For that, I will use tar command.

**Execute this:**

|  |  |
| --- | --- |
| 1 | tar -xvf learn\_chef\_httpd-0.2.0.tar.gz |

All the required files are automatically created under this Cookbook. There is no need to make any modifications. Let’s check the Recipe description inside my recipe folder.

**Execute this:**

|  |  |
| --- | --- |
| 1  2 | cd /root/chef-repo/learn\_chef\_httpd/recipes  cat default.rb |

Now, I will just upload this cookbook to my Chef Server as it looks perfect to me.

## ****4. Upload A Cookbook In The Chef Server****

In order to upload the Apache Cookbook that I have downloaded, first move this learn\_chef\_httpd file to the Cookbooks folder in the chef-repo. Then change your directory to cookbooks.

**Execute this:**

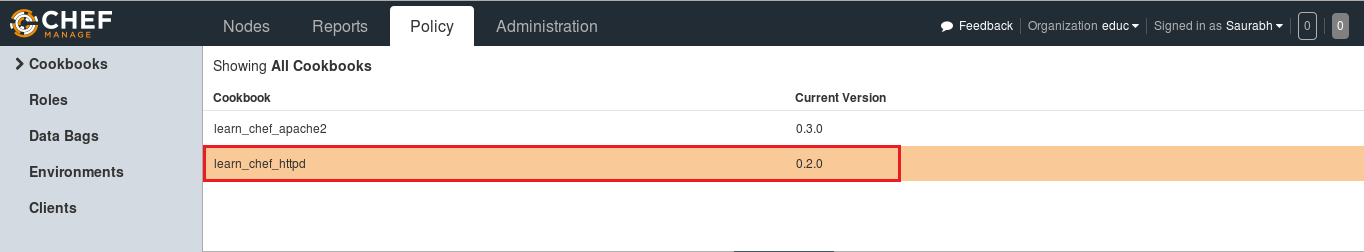
|  |  |
| --- | --- |
| 1  2 | mv /root/chef-repo/learn\_chef\_httpd /root/chef-repo/cookbooks  cd /root/chef-repo/cookbooks |

Now in this directory, execute the below command to upload the Apache Cookbook:

**Execute this:**

|  |  |
| --- | --- |
| 1 | knife cookbook upload learn\_chef\_httpd |

Verify the Cookbook from the Chef Server Management console. In the policy section, you will find the Cookbook that you have uploaded. Refer the screenshot below:



Now, our final step is to add Chef Node. We’ve setup a Workstation, a Chef Server and we need to add our Nodes to the Chef Server for automation.

## ****5. Connect A Node To The Chef Server****

The terminal color of my Node machine is different from the Workstation so that you will be able to differentiate between both.

I just need the IP address of my Node for that I will execute the below command in my Node machine.

**Execute this:**

|  |  |
| --- | --- |
| 1 | Ifconfig |

I will add my Chef Node to the Server by executing Knife Bootstrap command in which I will specify the IP address of The Chef Node and its name. Execute the command shown below:

**Execute this:**

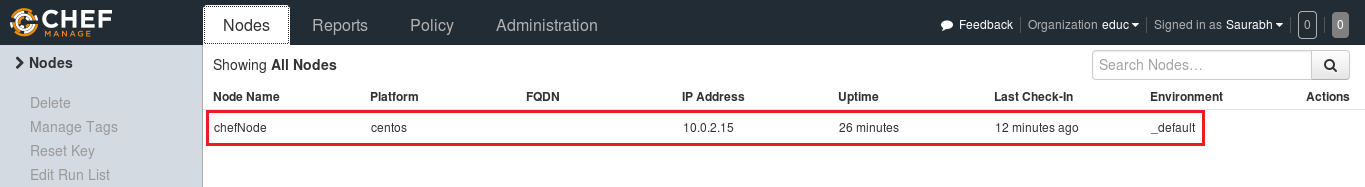
|  |  |
| --- | --- |
| 1 | knife bootstrap 192.168.56.102 --ssh-user root --ssh-password edureka --node-name chefNode |

This command will also initialize the installation of the Chef-Client in the Chef Node. You can verify it from the CLI on the Workstation using the knife command,as shown below:

**Execute this:**

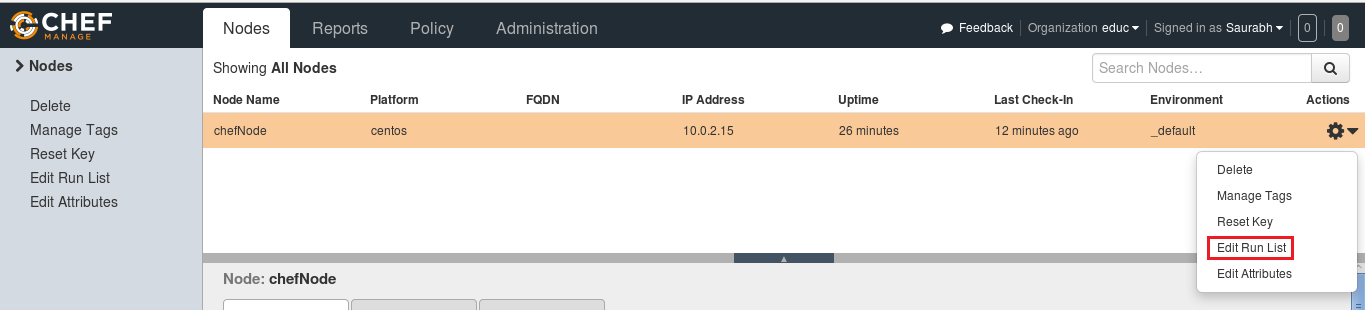
|  |  |
| --- | --- |
| 1 | Knife node list |

You can also verify from the Chef Server. Go to the nodes tab in your Server Management Console, here you will notice that the node that you have added is present. Refer the screenshot below:



## ****6. Deploy The Cookbook From The Server To The Node****

Let’s see how we can add a Cookbook to the Node and manage its Run list from the Chef Server. Run List describes the order in which the Cookbooks should be executed. As you can see in the screenshot below, click the Actions tab and select the Edit Run list option to manage the Run list.



In the Available Recipes,  you can see our learn\_chef\_httpd Recipe, you can drag that from the available packages to the current Run List and save the Run list.

Now login to your Node and just run chef-client to execute the Run List.

**Execute this:**

|  |  |
| --- | --- |
| 1 | chef-client |

## Chef Components[¶](https://docs.chef.io/chef_overview.html#chef-components)

The following diagram shows the relationships between the various elements of Chef, including the nodes, the server, and the workstation. These elements work together to provide the chef-client the information and instruction that it needs so that it can do its job. As you are reviewing the rest of this topic, use the icons in the tables to refer back to this image.

Chef has the following major components:

| **Component** | **Description** |
| --- | --- |
|  | One (or more) workstations are configured to allow users to author, test, and maintain cookbooks. Cookbooks are uploaded to the Chef server from the workstation. Some cookbooks are custom to the organization and others are based on community cookbooks available from the Chef Supermarket.  Ruby is the programming language that is the authoring syntax for cookbooks. Most recipes are simple patterns (blocks that define properties and values that map to specific configuration items like packages, files, services, templates, and users). The full power of Ruby is available for when you need a programming language.  Often, a workstation is configured to use the Chef Development Kit as the development toolkit. The Chef Development Kit is a package from Chef that provides a recommended set of tooling, including Chef itself, the chef command line tool, Test Kitchen, ChefSpec, Berkshelf, and more. |
|  | A node is any machine—physical, virtual, cloud, network device, etc.—that is under management by Chef.  A chef-client is installed on every node that is under management by Chef. The chef-client performs all of the configuration tasks that are specified by the run-list and will pull down any required configuration data from the Chef server as it is needed during the chef-client run. |
|  | The Chef server acts as a hub of information. Cookbooks and policy settings are uploaded to the Chef server by users from workstations. (Policy settings may also be maintained from the Chef server itself, via the Chef management console web user interface.)  The chef-client accesses the Chef server from the node on which it’s installed to get configuration data, performs searches of historical chef-client run data, and then pulls down the necessary configuration data. After the chef-client run is finished, the chef-client uploads updated run data to the Chef server.  Chef management console is the user interface for the Chef server. It is used to manage data bags, attributes, run-lists, roles, environments, and cookbooks, and also to configure role-based access for users and groups. |
|  | Chef Supermarket is the location in which community cookbooks are shared and managed. Cookbooks that are part of the Chef Supermarket may be used by any Chef user. How community cookbooks are used varies from organization to organization. |

Chef management console, chef-client run reporting, high availability configurations, and Chef server replication are available as part of Chef Automate.

The following sections discuss these elements (and their various components) in more detail.

## Workstations[¶](https://docs.chef.io/chef_overview.html#workstations)

A workstation is a computer running the [Chef Development Kit](https://docs.chef.io/about_chefdk.html) (Chef DK) that is used to author cookbooks, interact with the Chef server, and interact with nodes.

The workstation is where users do most of their work, including:

* Developing and testing cookbooks and recipes
* Testing Chef code
* Keeping the Chef repository synchronized with version source control
* Configuring organizational policy, including defining roles and environments, and ensuring that critical data is stored in data bags
* Interacting with nodes, as (or when) required, such as performing a bootstrap operation

The Chef Development Kit tooling encourages integration and unit testing, and defines workflow around cookbook authoring and policy, but it’s important to note that you know best about how your infrastructure should be put together. Therefore, Chef makes as few decisions on its own as possible. When a decision must be made tools uses a reasonable default setting that can be easily changed. While Chef encourages the use of the tooling packaged in the Chef DK, none of these tools should be seen as a requirement or pre-requisite for being successful using Chef.

### Workstation Components and Tools[¶](https://docs.chef.io/chef_overview.html#workstation-components-and-tools)

Some important tools and components of Chef workstations include:

| **Component** | **Description** |
| --- | --- |
|  | The Chef Development Kit is a package that contains everything that is needed to start using Chef:   * chef-client and ohai * chef and knife command line tools * Testing tools such as Test Kitchen, ChefSpec, Cookstyle, and Foodcritic * InSpec * Everything else needed to author cookbooks and upload them to the Chef server |
|  | ChefDK includes two important command-line tools:   * Chef: Use the chef command-line tool to work with items in a chef-repo, which is the primary location in which cookbooks are authored, tested, and maintained, and from which policy is uploaded to the Chef server * Knife: Use the knife command-line tool to interact with nodes or work with objects on the Chef server |
|  | The chef-repo is the repository structure in which cookbooks are authored, tested, and maintained:   * Cookbooks contain recipes, attributes, custom resources, libraries, files, templates, tests, and metadata * The chef-repo should be synchronized with a version control system (such as git), and then managed as if it were source code   The directory structure within the chef-repo varies. Some organizations prefer to keep all of their cookbooks in a single chef-repo, while other organizations prefer to use a chef-repo for every cookbook. |
|  | Use [Test Kitchen](http://kitchen.ci/) to automatically test cookbook data across any combination of platforms and test suites:   * Defined in a .kitchen.yml file. See the [configuration](https://docs.chef.io/config_yml_kitchen.html) documentation for options and syntax information. * Uses a driver plugin architecture * Supports cookbook testing across many cloud providers and virtualization technologies * Supports all common testing frameworks that are used by the Ruby community * Uses a comprehensive set of base images provided by [Bento](https://github.com/chef/bento) |
|  | Use ChefSpec to simulate the convergence of resources on a node:   * Is an extension of RSpec, a behavior-driven development (BDD) framework for Ruby * Is the fastest way to test resources and recipes |

## Cookbooks[¶](https://docs.chef.io/chef_overview.html#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.

### Components[¶](https://docs.chef.io/chef_overview.html#components)

Cookbooks are comprised of the following components:

| **Component** | **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. |
|  | 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   Chef has [many built-in resources](https://docs.chef.io/resource_reference.html) that cover all of the most common actions across all of the most common platforms. You can [build your own resources](https://docs.chef.io/custom_resources.html) 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. |

## Nodes[¶](https://docs.chef.io/chef_overview.html#nodes)

A node is any machine—physical, virtual, cloud, network device, etc.—that is under management by Chef.

### Node Types[¶](https://docs.chef.io/chef_overview.html#node-types)

The types of nodes that can be managed by Chef include, but are not limited to, the following:

| **Node Type** | **Description** |
| --- | --- |
|  | A physical node is typically a server or a virtual machine, but it can be any active device attached to a network that is capable of sending, receiving, and forwarding information over a communications channel. In other words, a physical node is any active device attached to a network that can run a chef-client and also allow that chef-client to communicate with a Chef server. |
|  | A cloud-based node is hosted in an external cloud-based service, such as Amazon Web Services (AWS), OpenStack, Rackspace, Google Compute Engine, or Microsoft Azure. Plugins are available for knife that provide support for external cloud-based services. knife can use these plugins to create instances on cloud-based services. Once created, the chef-client can be used to deploy, configure, and maintain those instances. |
|  | A virtual node is a machine that runs only as a software implementation, but otherwise behaves much like a physical machine. |
|  | A network node is any networking device—a switch, a router—that is being managed by a chef-client, such as networking devices by Juniper Networks, Arista, Cisco, and F5. Use Chef to automate common network configurations, such as physical and logical Ethernet link properties and VLANs, on these devices. |
|  | Containers are an approach to virtualization that allows a single operating system to host many working configurations, where each working configuration—a container—is assigned a single responsibility that is isolated from all other responsibilities. Containers are popular as a way to manage distributed and scalable applications and services. |

### Chef on Nodes[¶](https://docs.chef.io/chef_overview.html#chef-on-nodes)

The key components of nodes that are under management by Chef include:

| **Component** | **Description** |
| --- | --- |
|  | A chef-client is an agent that runs locally on every node that is under management by Chef. When a chef-client is run, it will perform all of the steps that are required to bring the node into the expected state, including:   * Registering and authenticating the node with the Chef server * Building the node object * Synchronizing cookbooks * Compiling the resource collection by loading each of the required cookbooks, including recipes, attributes, and all other dependencies * Taking the appropriate and required actions to configure the node * Looking for exceptions and notifications, handling each as required   RSA public key-pairs are used to authenticate the chef-client with the Chef server every time a chef-client needs access to data that is stored on the Chef server. This prevents any node from accessing data that it shouldn’t and it ensures that only nodes that are properly registered with the Chef server can be managed. |
|  | Ohai is a tool that is used to collect system configuration data, which is provided to the chef-client for use within cookbooks. Ohai is run by the chef-client at the beginning of every Chef run to determine system state. Ohai includes many built-in plugins to detect common configuration details as well as a plugin model for writing custom plugins.  The types of attributes Ohai collects include but are not limited to:   * Operating System * Network * Memory * Disk * CPU * Kernel * Host names * Fully qualified domain names * Virtualization * Cloud provider metadata   Attributes that are collected by Ohai are automatic level attributes, in that these attributes are used by the chef-client to ensure that these attributes remain unchanged after the chef-client is done configuring the node. |

## The Chef Server[¶](https://docs.chef.io/chef_overview.html#the-chef-server)

The Chef server acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the chef-client. Nodes use the chef-client to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server). This scalable approach distributes the configuration effort throughout the organization.

| **Feature** | **Description** |
| --- | --- |
|  | Search indexes allow queries to be made for any type of data that is indexed by the Chef server, including data bags (and data bag items), environments, nodes, and roles. A defined query syntax is used to support search patterns like exact, wildcard, range, and fuzzy. A search is a full-text query that can be done from several locations, including from within a recipe, by using the search subcommand in knife, the search method in the Recipe DSL, the search box in the Chef management console, and by using the /searchor /search/INDEX endpoints in the Chef server API. The search engine is based on Apache Solr and is run from the Chef server. |
|  | Chef management console is a web-based interface for the Chef server that provides users a way to manage the following objects:   * Nodes * Cookbooks and recipes * Roles * Stores of JSON data (data bags), including encrypted data * Environments * Searching of indexed data * User accounts and user data for the individuals who have permission to log on to and access the Chef server |
|  | A data bag is a global variable that is stored as JSON data and is accessible from a Chef server. A data bag is indexed for searching and can be loaded by a recipe or accessed during a search. |
|  | Policy defines how business and operational requirements, processes, and production workflows map to objects that are stored on the Chef server. Policy objects on the Chef server include roles, environments, and cookbook versions. |

### Policy[¶](https://docs.chef.io/chef_overview.html#policy)

Policy maps business and operational requirements, process, and workflow to settings and objects stored on the Chef server:

* Roles define server types, such as “web server” or “database server”
* Environments define process, such as “dev”, “staging”, or “production”
* Certain types of data—passwords, user account data, and other sensitive items—can be placed in data bags, which are located in a secure sub-area on the Chef server that can only be accessed by nodes that authenticate to the Chef server with the correct SSL certificates
* The cookbooks (and cookbook versions) in which organization-specific configuration policies are maintained

Some important aspects of policy include:

| **Feature** | **Description** |
| --- | --- |
|  | A role is a way to define certain patterns and processes that exist across nodes in an organization as belonging to a single job function. Each role consists of zero (or more) attributes and a run-list. Each node can have zero (or more) roles assigned to it. When a role is run against a node, the configuration details of that node are compared against the attributes of the role, and then the contents of that role’s run-list are applied to the node’s configuration details. When a chef-client runs, it merges its own attributes and run-lists with those contained within each assigned role. |
|  | An environment is a way to map an organization’s real-life workflow to what can be configured and managed when using Chef server. Every organization begins with a single environment called the \_default environment, which cannot be modified (or deleted). Additional environments can be created to reflect each organization’s patterns and workflow. For example, creating production, staging, testing, and developmentenvironments. Generally, an environment is also associated with one (or more) cookbook versions. |
|  | A cookbook version represents a set of functionality that is different from the cookbook on which it is based. A version may exist for many reasons, such as ensuring the correct use of a third-party component, updating a bug fix, or adding an improvement. A cookbook version is defined using syntax and operators, may be associated with environments, cookbook metadata, and/or run-lists, and may be frozen (to prevent unwanted updates from being made).  A cookbook version is maintained just like a cookbook, with regard to source control, uploading it to the Chef server, and how the chef-client applies that cookbook when configuring nodes. |
|  | A run-list defines all of the information necessary for Chef to configure a node into the desired state. A run-list is:   * An ordered list of roles and/or recipes that are run in the exact order defined in the run-list; if a recipe appears more than once in the run-list, the chef-client will not run it twice * Always specific to the node on which it runs; nodes may have a run-list that is identical to the run-list used by other nodes * Stored as part of the node object on the Chef server * Maintained using knife and then uploaded from the workstation to the Chef server, or maintained using Chef Automate |