Chef Introduction

Chef is a powerful automation platform that transforms complex infrastructure into code, bringing your servers and services to life. Whether you're operating in the cloud, on-premises, or a hybrid, Chef automates how applications are configured, deployed, and managed across your network, no matter its size.

Chef is built around simple concepts: achieving desired state, centralized modeling of IT infrastructure, and resource primitives that serve as building blocks. These concepts enable you to quickly manage any infrastructure with Chef. These very same concepts allow Chef to handle the most difficult infrastructure challenges on the planet. Anything that can run the chef-client can be managed by Chef.

Chef Components

The following diagram shows the relationships between the various elements of Chef, including the nodes, the server, premium features of 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

Component	Description
	provides an optional (but recommended) set of tooling, including Chef itself, the chef command line tool, 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, perform searches of historical chef-client run data, and then pull down the necessary configuration data. After the chef-client run is finished, the chef-client uploads updated run data to the Chef server (as the updated node object), uploads data generated by audit-mode (for additional rules processing by Chef Analytics), and generates reporting data. 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 Analytics provides real-time visibility into what is happening on the Chef server, including what's changing, who made those changes, and when they occurred. Details are tracked by the chef-client during the chef-client run. These details are uploaded to the Chef server at the end of the chef-client run. This data is used to build reports, run rules against the output of audit-mode, generate notifications based on the results of auditing, and visibility into messages that were generated during the chef-client run.
	Chef Supermarket is the location in which community cookbooks are authored and maintained. 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.

The premium features of the Chef server—Chef management console, Chef Analytics, chef-client run reporting, high availability configurations, and Chef server replication—may all be installed and configured for use with the Chef server. Each of these premium features are easily enabled and can be run as part of any Chef server deployment!

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

Workstations

A workstation is a computer that is configured to run various Chef command-line tools that synchronize with a chef-repo, author cookbooks, interact with the Chef server, interact with nodes, or applications like Delivery.

The workstation is the location from which most users do most of their work, including:

- Developing cookbooks and recipes (and authoring them using Ruby syntax and patterns)
- Keeping the chef-repo synchronized with version source control
- Using command-line tools
- 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

While Chef includes tooling like the Chef development kit, encourages integration and unit testing, and defines workflow around cookbook authoring and policy, 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, the chef-client uses a reasonable default setting that can be easily changed. While Chef encourages the use of the tooling packaged in the Chef development kit, none of these tools should be seen as a requirement or pre-requisite for being successful using Chef.

Tools

Some important components of workstations include:

Component	Description
	The Chef development kit is a package that contains everything that is needed to start using Chef: • chef-client • chef • Ohai • chef-zero • Testing tools like Kitchen, ChefSpec, and Foodcritic • Policy, including policy files • Chef provisioning • Everything else needed to author cookbooks and upload them to the Chef server
	Chef incudes two important command-line tools: • 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

Component	Description
	• 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, definitions, 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 Kitchen to automatically test cookbook data across any combination of platforms and test suites: • Defined in a .kitchen.yml file • 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
	Use ChefSpec to simulate the convergence of resources on a node: • Runs the chef-client on a local machine • Uses chef-zero or chef-solo • Is an extension of RSpec, a behavior-driven development (BDD) framework for Ruby • Is the fastest way to test resources and recipes

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 libraries, definitions, and custom resources

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

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.
	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.

Component	Description
	Use the **cookbook_file** resource to transfer files from a sub-directory ofCOOKBOOK_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 theCOOKBOOK_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

Component	Description
	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 for 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: Kitchen, ChefSpec, and Foodcritic.

Nodes

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

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

Node Type	Description
	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 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

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

Component	Description
	ensures that only nodes that are properly registered with the Chef server can be managed.
	Ohai is a tool that is used to detect attributes on a node, and then provide these attributes to the chef-client at the start of every chef-client run. Ohai is required by the chef-client and must be present on a node. (Ohai is installed on a node as part of the chef-client install process.) The types of attributes Ohai collects include (but are not limited to): • Platform details • Network usage • Memory usage • CPU data • Kernel data • Host names • Fully qualified domain names • Other configuration details Attributes that are collected by Ohai are automatic 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

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:

Feature	Description
	• 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

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.

Feature	Description
	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 is maintained using the Chef management console

Analytics

The Chef Analytics platform is a feature of Chef that provides real-time visibility into what is happening on the Chef server, including what's changing, who made those changes, and when they occurred. Individuals may be notified of these changes in real-time. Use this visibility to verify compliance against internal controls.

Chef Analytics includes:

Feature	Description
	Actions are policy and administrative changes made to the Chef server. The Chef server gathers a lot of data——each node object, the node run history for all nodes, the history of every cookbook and cookbook version, how policy settings, such as roles, environments, and data bags, are applied and to what they are applied, individual user data, and so on.

Feature	Description
	Chef Analytics includes a powerful rules processing system that allows notifications to be generated based on observed events in the data stream, such as: • Cookbook uploads • Modifications to environments • Machines on which chef-client runs have failed • Machines on which audit-mode runs have failed • Resources that were updated as a result of a chef-client run Notifications may be sent to any email address, a chat service like HipChat or Slack, or to a webhook-based service for generic intergrations.
	Reporting is used to keep track of what happened during the execution of chef-client runs across all of the infrastructure that is being managed by Chef. Reports can be generated for the entire organization and they can be generated for specific nodes.
	A control is an automated test that is built into a cookbook, and then used to test the state of the system for compliance. Compliance can be many things. For example, ensuring that file and directory management meets specific internal IT policies—”Does the file exist?”, “Do the correct users or groups have access to this directory?”. Compliance may also be complex, such as helping to ensure goals defined by large-scale compliance frameworks such as PCI, HIPAA, and Sarbanes-Oxley can be met.

Conclusion

Chef is a thin DSL (domain-specific language) built on top of Ruby. This approach allows Chef to provide just enough abstraction to make reasoning about your infrastructure easy. Chef includes a built-in taxonomy of all the basic resources one might configure on a system, plus a defined mechanism to extend that taxonomy using the full power of the Ruby language. Ruby was chosen because it provides the flexibility to use both the simple built-in taxonomy, as well being able to handle any customization path your organization requires.