**What is Chef?**

It is an **open source** configuration management tool developed by **opscode.** It is written in **Ruby**and **Erlang.** It automates the configuration and maintenance of more than one server. Chef uses **Domain Specific Language (DSL)** for writing system configuration. Chef follows **client-server**architecture.

**Why should we use Chef?**

Large companies have a continually changing infrastructure that requires be configuring and maintaining time to time. Chef provides:

* **Continuous deployment:**Software is deployed continuously enabling a company to keep in pace with the market requirements.
* **Increases system robustness:**Infrastructure automation ensures all bugs are caught and removed before deploying the software.
* **Adapt to the cloud:**The chef is easily integrated with infrastructure on the cloud.

Entire infrastructure can be recorded in the form of a repository that can be used as a blueprint to recreate the infrastructure from scratch.

**Advantages and Dis-advantages of Chef**

**Advantages of Chef**

* Fully automated deployment
* Within a minute you can configure thousands of nodes.
* Integrate with a cloud-based platform like AWS.
* Chef keeps the system under consistent check.
* You can record the entire infrastructure in the form of chef repository.
* The chef plays a vital role in DevOps software lifecycle

**Dis-advantages of Chef**

* Lacking documentation.
* Chef need code based knowledge for scripting the tool which makes it complicated.
* The master node can only be configurable in Linux\ Unix platform.

**Features of Chef**

* The chef is ideal for continuous deployment.
* Chef transforms the infrastructure into code.
* Chef supports multiple platforms like Arch, Debian, and Fedora.
* Chef is highly scalable, secure, and fault-tolerant.
* Chef has an active smart and excellent community support.
* The chef can operate in a distributed stand-alone or a centralized mode requiring a server.

**Comparison among popular automated tools**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metrics** | **Chef** | **Puppet** | **Ansible** | **SaltStack** |
| **Language** | Ruby | Puppet DSL | Python | Python |
| **Code** | Open Source | Open Source | Open Source | Open Source |
| **Infrastructure** | Mutable | Mutable | Mutable | Mutable |
| **Push/ Pull Mechanism** | Pull | Pull | Push | Push |
| **Architecture** | Client/ Server | Client/ Server | Client | Client/ Server |
| **Scalability** | Highly Scalable | Highly Scalable | Highly Scalable | Highly Scalable |
| **Management** | Hard | Hard | Easy | Easy |
| **Cloud Integration** | Yes | Yes | Yes | Yes |
| **Industry Example** | Facebook, Youtube, Splunk, and  Linkedin | Twitter, VMware, Redhat, and Paypal | Apple, juniper, SaveMart, and NASA | Lyft |

**Configuration Management**

It is the collection of engineering practice that provides a systematic way to manage entities for efficient deployment. These entities include **code, infrastructure,**and**people**.

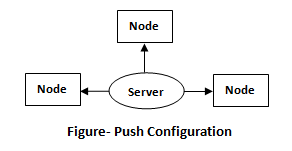
* **Code:** Code needs to be updated and stored.
* **Infrastructure:** Infrastructure needs to be configured.
* **People:** People need coordination.

You need to specify the configuration once on the central chef server and replicate that on thousands of nodes. Configuration management performs the following task-

* Find out which components to change when requirements change.
* Revert to a previous version of the component if you have replaced with a new version.
* Replace the wrong component.
* When requirement have changed redoing an implementation.

There are**two** types of configuration management.

**Push Configuration:** Centralized server pushes configuration to the nodes.



**Pull Configuration:** Nodes checks with the server periodically and fetches the configuration from it. Chef follows pull configuration.

## pull configurationComponents of Chef

**Workstation**

The workstation is the system from which all configurations can manage, later pushed to the central chef server. This code is called **recipe**, and the collection of the recipe is called **cookbook.**These configurations are tested before pushing into the chef server. The workstation has a command line tool called **Knife**that is used to interact with the chef server. Multiple workstations can manage the central chef server. Following are the functions performed by workstation:

* Writing cookbook and recipes that will later be pushed to the central chef server.
* Managing nodes on the central chef server.

**Chef Nodes**

Node is a physical or cloud-based system that is managed using a central chef server. A component that needs to be present on the node is an agent. The agent will establish communication with the central chef server. This agent is called **Chef Client.**

**Chef Server**

The server stores the cookbooks, the policies and configuration data. This is a centrally located server that holds all the data related to infrastructure. Chef runs in server-client mode. There are**three**types of chef server available-

**Chef Solo-**Solo removes the need of having a centralize chef server to test configurations on nodes. It is a kind of local deployment. Functionality wise, Chef Solo is very similar to Hosted Chef. However, in Chef Solo, you don’t have any limitation of nodes.

**Open Source Chef-** This is completely free an open source can be installed anywhere. Its functionality is similar to Hosted Chef. This also ensures that the open source Chef server remains as the local infrastructure and supports upgrades.

**Hosted Chef-**You are free from the repository of managing a central chef serves yourself. Hosted Chef is hosted by Opscode. It is **software as a service** **(SaaS)** offering. It provides full scalability, all-time availability of resources, and resource-based access control.

**Workstation Components**

**Local chef repository**

A repository is a place where every configuration component of central chef server is stored. Chef repository can be synchronized with the chef server.

**Chef Client**

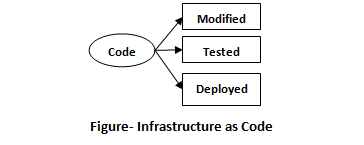
Chef client act as a thick client means it does most of the heavy lifting. Chef client periodically contacts to central chef server downloading required configuration applicable to node compiling those configurations locally on the node. It establishes communication with the central chef server. Following are the functions performed by chef-client:

* Node interacts with the central chef server.
* Periodically checks for the new configuration, if any.
* It manages the registration of the node to the central chef server.
* Node pulls the cookbooks and applies them to the node to configure it.
* Each node can have a different configuration environment.
* **Ohai**fetches the current state of the node it is located in.

**Infrastructure as Code**

Infrastructure as code is a type of infrastructure where the operation team manages the code rather than a manual procedure. It makes policies and configuration management more straightforward and more efficient.

Suppose that you are a system administrator in a company and you need to set up a server and install 25 software applications over it. This process will take too much time to install manually. Let us consider you have a code for installing software rather than doing it manually.



* Modified the code to meet the software installations requirements.
* The code should be tested to catch any bugs.
* The code can be easily deployed, and all installations would take place automatically.

**Building Blocks of Chef**

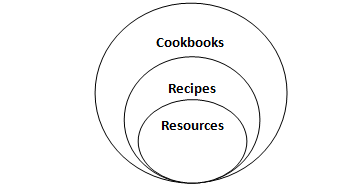
**Recipes**

The recipe describes a series of resources that should be in a particular state that is packages that should be installed, services that should be running or files that should be written. The user writes recipes that describe how the chef manages applications and utilities and how they are configured. A recipe has the following characteristics:

* It is a collection of similar types of resources.
* It is written in the Ruby language with the help of some syntax.
* One recipe can depend on one or more recipes.
* It runs/executes in the same manner as is categorized in the run-list.

**Cookbooks**

These are the key segment contain qualities and data about the coveted condition of a node. Specifies the resources to use and the order in which they are to be applied. The cookbook is the most important and reusable component of a Chef Server. The cookbook is the basic unit used to define rules and regulations for configuration purposes. It contains the recipes that have all the specifications of resource management and also, it has the order of application of resources. It includes the procedure to reuse the collection of resources.



* **Templates**
* **Attribute values**
* **Definition and custom resources**
* **File distributions**

**Template:** It is similar to files, but it is not static. The extension of these files is **.ebr**, which means they conation embedded Ruby. The template is used to update and make the existing file.

**Attribute Values:** Attribute value used in the cookbook. There are different kinds of attributes that can be applied with a different level of precedence over the final setting that the node operates under.

**Definition and Resources:** there can be multiple resources in a recipe which will help in configuring the infrastructure

**File Distribution:** it contains any static file which will be placed on the chef nodes that uses the cookbooks. A recipe can be declared as a resource that moves the files from that directory to the final node. For example cookbook, file, service, package.