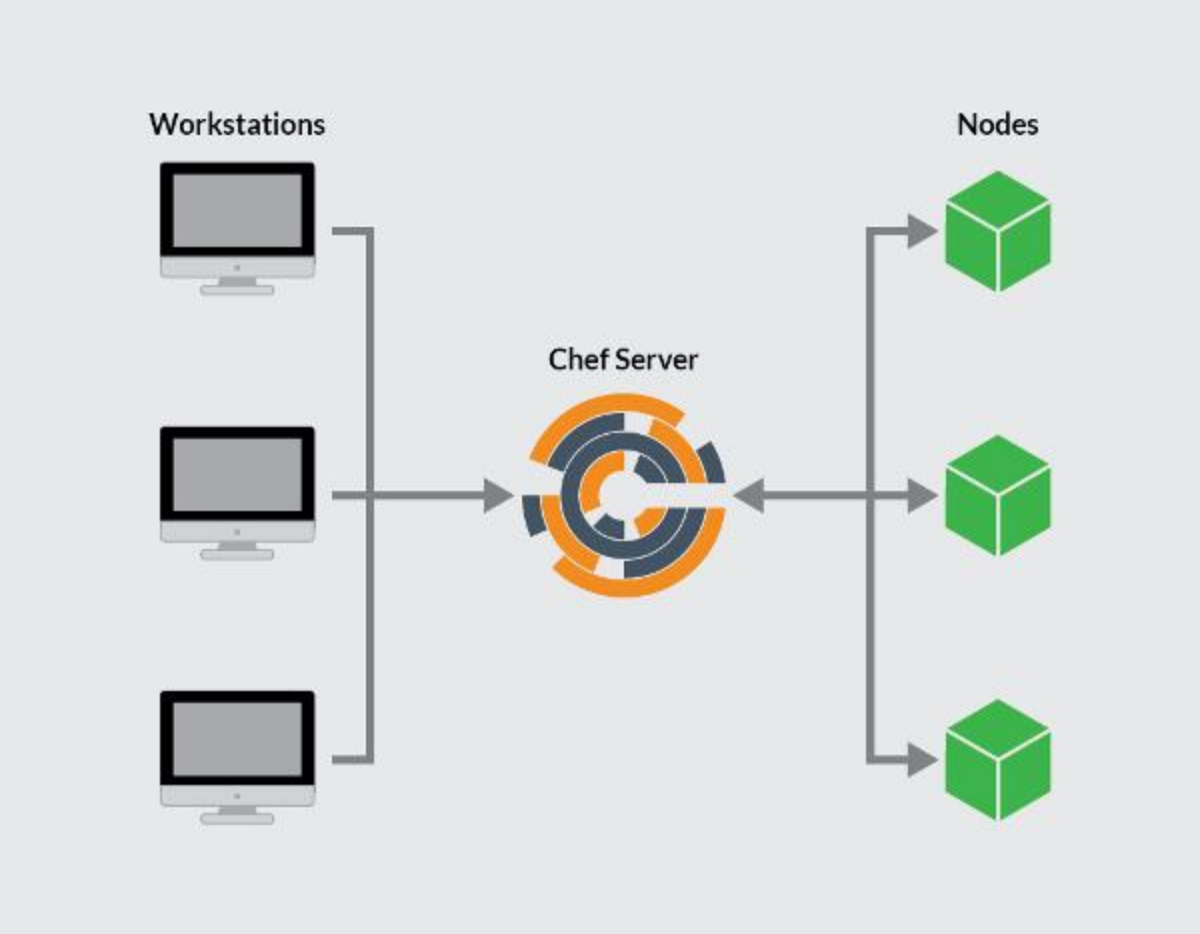
chef is an automation platform that “turns infrastructure into code,” allowing organizations or persons with large frameworks to generate a process that will save time and effort when making changes to part or all of their server fleet.
Chef works with three core components: The Chef server, workstations, and nodes. The Chef server is the hub of Chef operations, where changes are stored for use. Workstations are static computers or virtual servers where all code is created or changed. There can been as many workstations as needed, whether this be one per person or otherwise. Finally, nodes are the servers that need to be managed by Chef – these are the machines that changes are being pushed to, generally a fleet of multiple machines that require the benefits of an automation program.



These three components communicate in a mostly-linear fashion, with any changes being pushed from workstations to the Chef server, and then pulled from the server to the nodes. In turn, information about the node passes to the server to determine which files are different from the current settings and need to be updated.

If you wish to farther explore Chef please see the guides Setting Up a Chef Server, Workstation, and Node on Ubuntu 14.04 and Creating Your First Chef Cookbook.

The Chef Server

The Chef server is the primary mode of communication between the workstations where your infrastructure is coded, and the nodes where it is deployed. All configuration files, cookbooks, metadata, and other information are stored on the server. The Chef server also keeps information regarding the state of all nodes at the time of the last chef-client run.
Any changes made must pass through the Chef server to be deployed. Prior to accepting or pushing changes, it verifies that the nodes and workstations are paired with the server through the use of authorization keys, and then allows for communication between the workstations and nodes.

Bookshelf

The Bookshelf is a versioned repository where cookbooks are stored on the Chef server (generally located at `/var/opt/opscode/bookshelf`; full root access is needed). When a cookbook is uploaded to the Chef server, the new version is compared to the one already stored; if there are changes, a new version is stored. The Chef server only stores one copy of a file or template at once, meaning if resources are shared between cookbooks and cookbook versions, they will not be stored multiple times.

Workstations

Workstations are where users create, test, and maintain cookbooks and policies that will be pushed to nodes. Cookbooks created on workstations can be used privately by one organization, or uploaded to the Chef Supermarket for others to use. Similarly, workstations can be used to download cookbooks created by other Chef users and found in the Supermarket.

Workstations are set up to use the *Chef Development Kit* (ChefDK), and can be located on virtual servers or on physical workstation computers. Workstations are set to interact with only one Chef server, and most work will be done in the `chef-repo` directory located on the workstation.

chef-repo

The `chef-repo` directory is the specific area of the workstation where cookbooks are authored and maintained. The `chef-repo` is always version-controlled, most often through the use of Git, and stores information and history that will be used on nodes, such as cookbooks, environments, roles, and data bags. Chef is able to communicate with the server from the `chef-repo` and push any changes via the use of the `knife` command, which is included in the ChefDK.
Originally the `chef-repo` had to be pulled from GitHub using git commands, but that action is now integrated into Chef through the use of the `chef generate repo chef-repo` command.

Knife

The `knife` command communicates between the `chef-repo` located on a workstation and the Chef server. `knife` is configured with the `knife.rb` file, and is used from the workstation:

`~/chef-repo/.chef/knife.rb`

```
log_level                :info
log_location             STDOUT
node_name                'username'
client_key               '~/chef-repo/.chef/username.pem'
validation_client_name   'shortname-validator'
validation_key           '~/chef-repo/.chef/shortname.pem'
chef_server_url          'https://123.45.67.89/organizations/shortname'
syntax_check_cache_path  '~/chef-repo/.chef/syntax_check_cache'
cookbook_path [ '~/chef-repo/cookbooks' ]
```

The default `knife.rb` file is defined with the following properties:

- **log_level:** The amount of logging that will be stored in the log file. The default value, `:info`, notes that any informational messages will be logged. Other values include `:debug:`, `:warn`, `:error`, and `:fatal`.
- **log_location:** The location of the log file. The default value, `STOUT` is for*standard output logging*. If set to another value standard output logging will still be performed.
- **node_name:** The username of the person using the workstation. This user will need a valid authorization key located on the workstation.
- **client_key:** The location of the user's authorization key.
- **validation_client_name:** The name for the server validation key that will determine whether a node is registered with the Chef server. These values must match during a chef-client run.
- **validation_key:** The path to your organization's validation key.
- **chef_server_url:** The URL of the Chef server, with `shortname` being the defined shortname of your organization. This can also be an IP address.`/organizations/shortname` must be included in the URL.
- **syntax_check_cache_path:** The location in which `knife` stores information about files that have been checked for appropriate Ruby syntax.
- **cookbook_path:** The path to the cookbook directory.

Nodes

A *node* is a system configured to run the chef-client. This can be any system, as long as it is being maintained by Chef.
Nodes are validated through the `validator.pem` and `client.pem` certificates that are created on the node when it is bootstrapped. All nodes must be bootstrapped over SSH as either the root user or a user with elevated privileges.
Nodes are kept up-to-date through the use of the chef-client, which runs a convergence between the node and the Chef server. What cookbooks and roles the node will take on depends on the run list and environment set for the node in question.

chef-client

The chef-client checks the current configuration of the node against the recipes and policies stored in the Chef server and brings the node up to match. The process begins with the chef-client checking the node's run list, loading the cookbooks required, then checking and syncing the cookbooks with the current configuration of the node.
The chef-client must be run with elevated privileges in order to properly configure the node, and should be run periodically to ensure that the server is always up to date – often this is achieved through a cron job or by setting up the chef-client to run as a service.

Run Lists

Run lists define what cookbooks a node will use. The run list is an ordered list of all cookbooks and recipes that the chef-client needs to pull from the Chef server to run on a node. Run lists are also used to define roles, which are used to define patterns and attributes across nodes.

Ohai

Ohai collects information regarding nodes for the Chef server. It is required to be present on every node, and is installed as part of the bootstrap process.

The information gathered includes network and memory usage, CPU data, kernel data, hostnames, FQDNs, and other automatic attributes that need to remain unchanged during the chef-client run.

Environments

Chef environments exist to mimic real-life workflow, allowing for nodes to be organized into different "groups" that define the role the node plays in the fleet. This allows for users to combine environments and versioned cookbooks to have different attributes for different nodes. For example, if testing a shopping cart, you may not want to test any changes on the live website, but with a "development" set of nodes.

Environments are defined in `chef-repo/environments` and saved as Ruby or JSON files.

As a Ruby file:

chef-repo/environments/environame.rb

```
name "environmentname"
description "environment_description"
cookbook_versions  "cookbook" => "cookbook_version"
default_attributes "node" => { "attribute" => [ "value", "value", "etc."
] }
override_attributes "node" => { "attribute" => [ "value", "value", "etc."
] }
```

As a JSON:

chef-repo/environments/environame.json

```
{
  "name": "environmentname",
  "description": "a description of the environment",
  "cookbook_versions": {

  },
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": {

  },
  "override_attributes": {

  }
```

All nodes are automatically set to the “default” environment upon bootstrap. To change this, the environment should be defined in the `client.rb` file found in `/etc/chef` on the nodes.

Cookbooks

Cookbooks are the main component of configuring nodes on a Chef infrastructure. Cookbooks contain values and information about the *desired state*of a node, not how to get to that desired state – Chef does all the work for that, through their extensive libraries.
Cookbooks are comprised of recipes, metadata, attributes, resources, templates, libraries, and anything else that assists in creating a functioning system, with attributes and recipes being the two core parts of creating a cookbook. Components of a cookbook should be modular, keeping recipes small and related.

Cookbooks can and should be version controlled. Versions can help when using environments and allow for the easier tracking of changes that have been made to the cookbook.

Recipes

Recipes are the fundamental part of cookbooks. Recipes are written in Ruby and contain information in regards to everything that needs to be run, changed, or created on a node. Recipes work as a collection of *resources* that determine the configuration or policy of a node, with resources being a configuration element of the recipe. For a node to run a recipe, it must be on that node's run list.

Attributes

Attributes define specific values about a node and its configuration. These values are used to override default settings, and are loaded in the order cookbooks are listed in the run list. Often attributes are used in conjunction with templates and recipes to define settings.

Files

These are static files that can be uploaded to nodes. Files can be configuration and set-up files, scripts, website files – anything that does not been to have different values on different nodes.

Libraries

Although Chef comes with a number of libraries built in, additional libraries can be defined. Libraries are what bring recipes to life: If a recipe is the *desired state*of a node, than added libraries contain the behind-the-scenes information Chef needs for the nodes to reach this state. Libraries are written in Ruby, and can also be used to expand on any functionalities that Chef already contains.

Providers and Resources

Providers and resources are also used to define new functionality to use in Chef recipes. A *resource* defines a set of actions and attributes, whereas *provider*informs the chef-client how to commit each action.

Templates

Templates are embedded Ruby files (.erb) that allows for content based on the node itself and other variables generated when the chef-client is run and the template is used to create or update a file.

Create a file

```
file '/tmp/something' do
  owner 'root'
  group 'root'
  mode '0755'
  action :create
end
```

Create a file in Microsoft Windows

To create a file in Microsoft Windows, be sure to add an escape character—\—before the backslashes in the paths:

```
file 'C:\\tmp\\something.txt' do
  rights :read, 'Everyone'
  rights :full_control, 'DOMAIN\\User'
  action :create
end
```

Remove a file

```
file '/tmp/something' do
  action :delete
end
```

Set file modes

```
file '/tmp/something' do
  mode '0755'
end
```

Linux Commands execution chef

```
bash 'install_something' do
  user 'root'
  cwd '/tmp'
```

```
  code <<-EOH
  wget http://www.example.com/tarball.tar.gz
  tar -zxf tarball.tar.gz
  cd tarball
  ./configure
  make
  make install
  EOH
end
```

Start a service

```
service 'example_service' do
  action :start
end
```

Start a service, enable it

```
service 'example_service' do
  supports :status => true, :restart => true, :reload => true
  action [ :enable, :start ]
end
```

Use a pattern

```
service 'samba' do
  pattern 'smbd'
  action [:enable, :start]
end
```

Use the :nothing common action

```
service 'memcached' do
  action :nothing
```

```
  supports :status => true, :start => true, :stop => true, :restart => true
end
```

Use the supports common attribute

```
service 'apache' do
  supports :restart => true, :reload => true
  action :enable
end
```

```
#########################################################################
```

Manage a service, depending on the node platform

```
service 'example_service' do
  case node['platform']
  when 'centos','redhat','fedora'
    service_name 'redhat_name'
  else
    service_name 'other_name'
  end
  supports :restart => true
  action [ :enable, :start ]
end
```

For example, installing multiple packages:

```
package ['package1', 'package2']
```

Upgrading multiple packages:

```
package ['package1', 'package2']  do
  action :upgrade
end
```

Removing multiple packages:

```
package ['package1', 'package2']  do
  action :remove
end
```

```
user 'a user' do
  comment 'A random user'
  uid '1234'
  gid '1234'
  home '/home/random'
  shell '/bin/bash'
  password '$1$JJsvHslasdfjVEroftprNn4JHtDi'
end
```

Connect to git

```
git "/path/to/check/out/to" do
  repository "git://github.com/opscode/chef.git"
  reference "master"
  action :sync
end
```