Chef Infra as Code

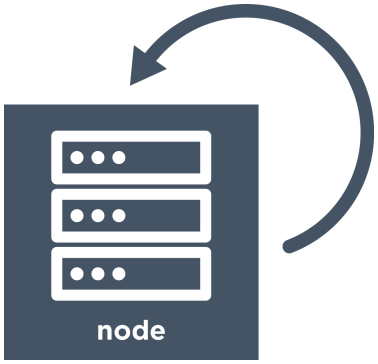
Introduction

In the eternity of Devops, Chef is a weapon to ravage the time consumption in infrastructure setup and application delivery. Chef, basically automate everything, from system configuration to application delivery to end user.
According to chef's official documentation *"Chef is a systems and cloud infrastructure automation framework that makes it easy to deploy servers and applications to any physical, virtual, or cloud location, no matter the size of the infrastructure."* Chef provide single end solution for server automation, infrastructure environment and continuously deliver your application.

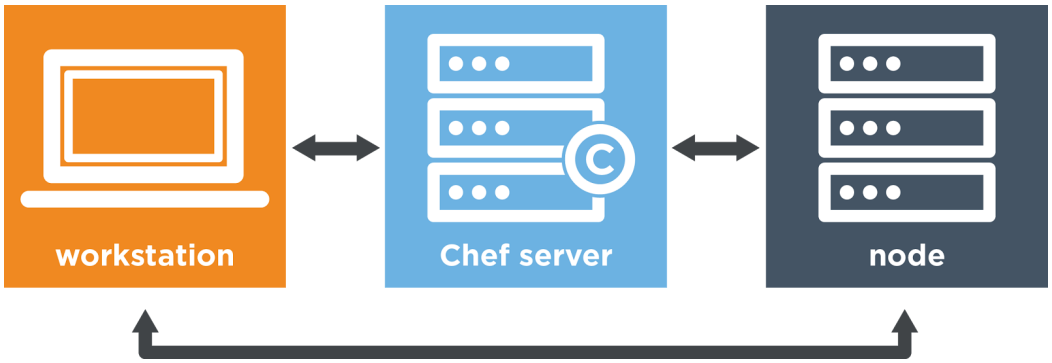
Getting Started

Chef treats infrastructure as code, so it becomes more manageable, versionable and testable. Chef can work in two fashion, a) Standalone and b) Client-Server.

- Standalone is a basic mode of chef, where you can manage a server directly. To perform this, install Chefdk (chef's development kit), it provide tools that allow you to configure a machine directly.



- Chef Client-Server-Workstation mode allows you to manage your multiple remote systems. Workstation allows you to configure your infra as code from numerous ends. Chef-workstations allow users to create, maintain and test cookbooks. Chef-client runs on every chef managed machine, and responsible to update machine state to desirable state. Chef-server is the nerve center of complete setup, chef-workstation upload cookbooks, runlist and policies to chef-server, chef-client pull the corresponding configuration from here.



Bedrock ingredient

Resources

Resources are the fundamental element of chef. These are responsible for the desired state of any chef managed element. Chef resources like package, service, template and file. A resource has two parts ie. action and properties.

- Properties: Properties are the equities which are applicable on target entity such as file resource has below listed properties.

```
file 'name' do
           content                    String
           group                      String, Integer
           mode                       String, Integer
           notifies                   # see description
           owner                      String, Integer
           path                          String # defaults to 'name' if
not specified
           provider                   Chef::Provider::File
           subscribes                 # see description
           verify                     String, Block
           action                      Symbol # defaults to :create if
not specified
end
```

Where

content is string to write in file.
Group to which this file belongs to.
Mode is the string or integer format of permission on this file.
Owner describes the owner of file.
Path is the complete directory lane of file.

You can explicitly define a provider for a resource. Notify and Subscribe are two attributes to trigger another resource from this resource.

Notify triggers successor resource when its parent resource state changes. Although subscribe tune in with another resource and take action if tate of the resource being listened to changes. A resource can notify multiple resource and get subscribed by multiple resources.

```
$ notifies :action, 'resource[name]', :timer
```

```
$ subscribes :action, 'resource[name]', :timer
```

Where :action is :before, :delayed, :immediate or :immediately and :timer is time before that action. Use one notify or subscribe statement for a each resource.

- Action: These are the basic functions that can a resource perform. It depends on the resource that which action is applicable on that resource. For example on file resource these actions are available.

:create	Create the file. If file exist then update.
:create_if_missing	Create only if file is missing.
:delete	Delete the file.

:nothing	Do nothing.
:touch	Create an empty file.

Below is an typical example of directory resource.

```
directory '/opt/mydir' do
  owner 'root'
  group 'root'
  mode '0755'
  action :create
end
```

This will ensure the /opt/mydir with root as owner and group and 0755 permissions. Start exercise with a basic resource.

```
$ sudo chef-apply -e "package 'nginx'"
```

(This will install the nginx server on target instance.)
For more information refer to the Resources Reference.

Recipes

Chef recipes are purposeful batch of resources. The language backing behind the recipes is ruby. These are the imperative factor of any cookbook. Apart from resources some attributes can also be declared here to override. Recipes can be included any other recipe to justify the dependencies. Let's setup nginx vhost using multiple recipes in a row. Go through the below printed example.

```
## installnginx.rb
package 'epel-release' do
 action :install
end
package 'nginx' do
 action :install
end
service 'nginx' do
 action [ :enable, :start ]
end
```

This recipe comprises package and service resources to install and start nginx on a centos machine. To apply this recipe with

```
$ sudo chef-apply installnginx.rb
```

Now configure virtual host using below printed recipe.

```
## configurevirtualhost.rb

directory '/usr/share/nginx/blog' do

     recursive 'true'

     action :create

end

file  '/usr/share/nginx/blog/index.html' do

   content  '<html><title>Chef  Session</title><body><h1>Hello  this  is
blog from opstree !!</h1></body></html>'

end

file '/etc/nginx/conf.d/blog.opstree.com.conf' do
  content IO.read('/vagrant/recipes/blog.opstree.com.conf')
end
service 'iptables' do
  action :stop
end
service 'nginx' do
  action [:stop, :start]
end
```

This recipe uses *directory*, *file* and *service* resources with their corresponding attributes and actions. *Directory* resource ensure that '/usr/share/nginx/blog' directory path should be present. Recursive attribute confirms the creation of parent directories recursively.

First *file* resource creates an index.html file with content specified in content attribute. Second *file* resource read the content of a file /vagrant/recipes/blog.opstree.com.conf and put this into /etc/nginx/conf.d/blog.opstree.com.conf file using IO.read ruby function.

```
## /vagrant/recipes/blog.opstree.com.conf
server {
    listen       80;
    server_name  blog.opstree.com;

    location / {
        root   /usr/share/nginx/blog;
```

```
        index  index.html index.htm;
    }

    error_page  404              /404.html;
    location = /404.html {
        root   /usr/share/nginx/blog;
    }

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/blog;
    }

}
```

Prior service resource stop iptables service. Next service resource triggers stop and then start action for nginx web server.
Now apply configurevirtualhost.rb to setup virtual host.

```
$ sudo chef-apply configurevirtualhost.rb
```

This will setup a virtual host blog.opstree.com on your machine.

For more information refer to the About Recipes.

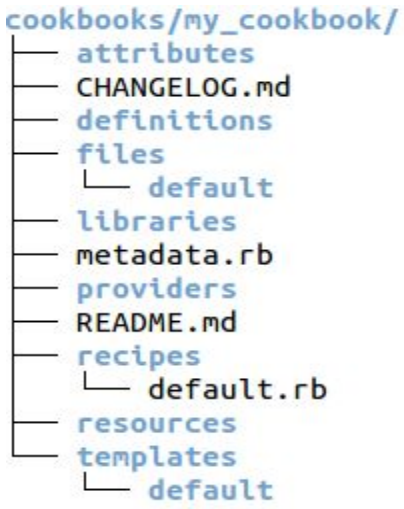
Cookbooks

Cookbooks are the building walls of chef's house. These are the set of everything that is required to manage a node. With cookbooks it is possible to manage an infrastructure as a code. It consist of

- Recipes
- Attributes
- Files
- Templates
- Libraries
- Definitions and custom resources.

It is answerable for the the whole plot of desired state of a client.

Basic cookbook directory structure is like



- Attributes — This directory contains attributes for that are used by the resources in the cookbook.
- Definition — This directory create definition of new resources.
- Files — Files folder consist static files related to cookbook.
- Libraries — Contains ruby code to empower the cookbook.
- Providers — The steps to do when a resource is called. can create a custom provider for custom resources.
- Recipes — Directory contains all recipes of cookbook. Recipe is collection of predefined and custom resources. Can have ruby code and may include other recipes to resolve dependencies.
- Resources — Consist of custom resources to perform cookbook specific task.
- Templates — Holds all the templates files for dynamic configuration.

Generate a cookbook using chef command

$ chef generate cookbook my_cookbook

This will generate a minimal set of components of any cookbook. To handle cookbook with other full capability use “Knife” commands.

Let’s took an sample cookbook to install nginx and setup a vhost in single hand.

Generate a cookbook nginx using following command

```
$ knife cookbook create nginx
```

- Create a recipe with following content.
  ```
  $ vim recipes/default.rb
  ```

```
package 'epel-release' do
 action :install
end
package 'nginx' do
 action :install
end
directory "#{node['nginx']['webroot']}" do
  recursive true
end
template "/etc/nginx/conf.d/#{node['nginx']['conffile']}" do
  source 'chefmanagedconf.conf.erb'
  variables(
    :port => "#{node['nginx']['port']}",
    :servername => "#{node['nginx']['servername']}",
    :webroot => "#{node['nginx']['webroot']}"
  )
end
file  '#{node['nginx']['webroot']}/index.html' do
   content  '<html><title>Chef  Session</title><body><h1>Hello  this  is
blog from opstree !!</h1></body></html>'
end
```

```
service 'nginx' do
 action [ :enable, :start ]
end
```

This recipe uses package, directory, template, file and service resources. Every resource is having its specific attributes to specify the state of machine. As this file have some variables like webroot, conffile and servername we have to define them somewhere.

- Create a attribute file.

  ```
  $ vim attributes/default.rb
  ```

```
default['nginx']['port'] = "80"
default['nginx']['webroot'] = "/usr/share/nginx/blog"
default['nginx']['servername'] = "blog.opstree.com"
default['nginx']['conffile'] = "blog.opstree.com.conf"
```

This file have default values of the variables used in this cookbook.

- Create a template for conf file under templates folder.

  ```
  $ vim templates/default/chefmanagedconf.conf.erb
  ```

```
server {
    listen       <%= @port  %>;
    server_name  <%= @servername  %>;

    location / {
        root   <%= @webroot  %>;
        index  index.html index.htm;
    }

    error_page  404              /404.html;
    location = /404.html {
        root   <%= @webroot  %>;
```

```
    }
    # redirect server error pages to the static page /50x.html
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   <%= @webroot  %>;
         }
   }
```

The template resource used in default.rb recipe demands for a template file in cookbook. File chefmanagedconf.conf.erb is the template file for nginx vhost configuration. Chef template have .erb extension. This template file create a configuration file with supplied variables.

- Create a runlist json file.

  ```
  $ vim runlsit.json
  ```

```
{
  "run_list": [ "recipe[nginx]" ]
}
```

To run a cookbook add it to a runlist file. Which define chef-client the order of cookbooks to run.

- To run this cookbook type following command.

  ```
  $ chef-client --local-mode -j /path/to/file.json
  ```

This will run nginx cookbook and install nginx and setup nginx vhost on your machine automatically.

Download cookbooks from Chef Supermarket, it is the hub of chef community cookbooks.

Chef-knife

As per official website *"knife is a command-line tool that provides an interface between a local chef-repo and the Chef server"*. Knife can manage different chef components.

- ❏ Nodes
- ❏ Cookbooks
- ❏ Roles
- ❏ data bags

- ❏ Environments

Type **knife --help** to look for all option provided by knife.

```
$ knife --help
```

To manage cookbooks look for cookbooks command in knife help

```
$ sudo knife -h |  sed -n '/COOKBOOK COMMANDS/,/^$/p'
```

- ➔ To create a cookbook

```
$ knife cookbook create my_cookbook -C "testCookbook" -m
"my@email.com" -I mycookbookv2 -r md
```

This will create a my_cookbook with copyright, email, license, and readme format options.

- ➔ List all cookbooks

```
$ knife cookbook list
```

- ➔ Create databag

```
$ knife data bag create database
```

- ➔ Delete databag

```
$ knife data bag delete
```

- ➔ Create Role

```
$ knife role edit webapp
```

- ➔ Delete Role

```
$ knife role delete webapp
```

- ➔ Create Environment

```
$ knife environment create prod
```

- ➔ Delete Environment

```
$ knife environment delete dev
```

Chef-kitchen

It is required to test your cookbook on every environment and different platform. Chef provide its own tool for this testing purpose. Kitchen can test cookbooks across many cloud providers and virtualization technologies. Chef kitchen has four phases to test a cookbook.

- ❏ Kitchen create

- ❏ Kitchen converge
- ❏ Kitchen login
- ❏ Kitchen verify
- ❏ Kitchen destroy

Kitchen firstly creates bare machine with specified platform, next converg list of cookbooks into it, then login into the machine and verify the state of machine and finally destroy it.

To know more visit Chef-Kitchen Do it simply..

Chef-Foodcritic

It is a lint tool for chef cookbooks. You can verify cookbook syntax and other standards with foodcritic. It has 61 built-in rules that diagnose cookbook and find problems related to syntax and style inconsistency.

- ➔ Install foodcritic

```
$ gem install foodcritic
```

- ➔ Use foodcritic against your cookbook.

```
$ foodcritic my_cookbook_dir
```

For more info visit Foodcritic

We are on our way to publish a complete blog series on "Chef". To have more information visit Chef Start here with ease..