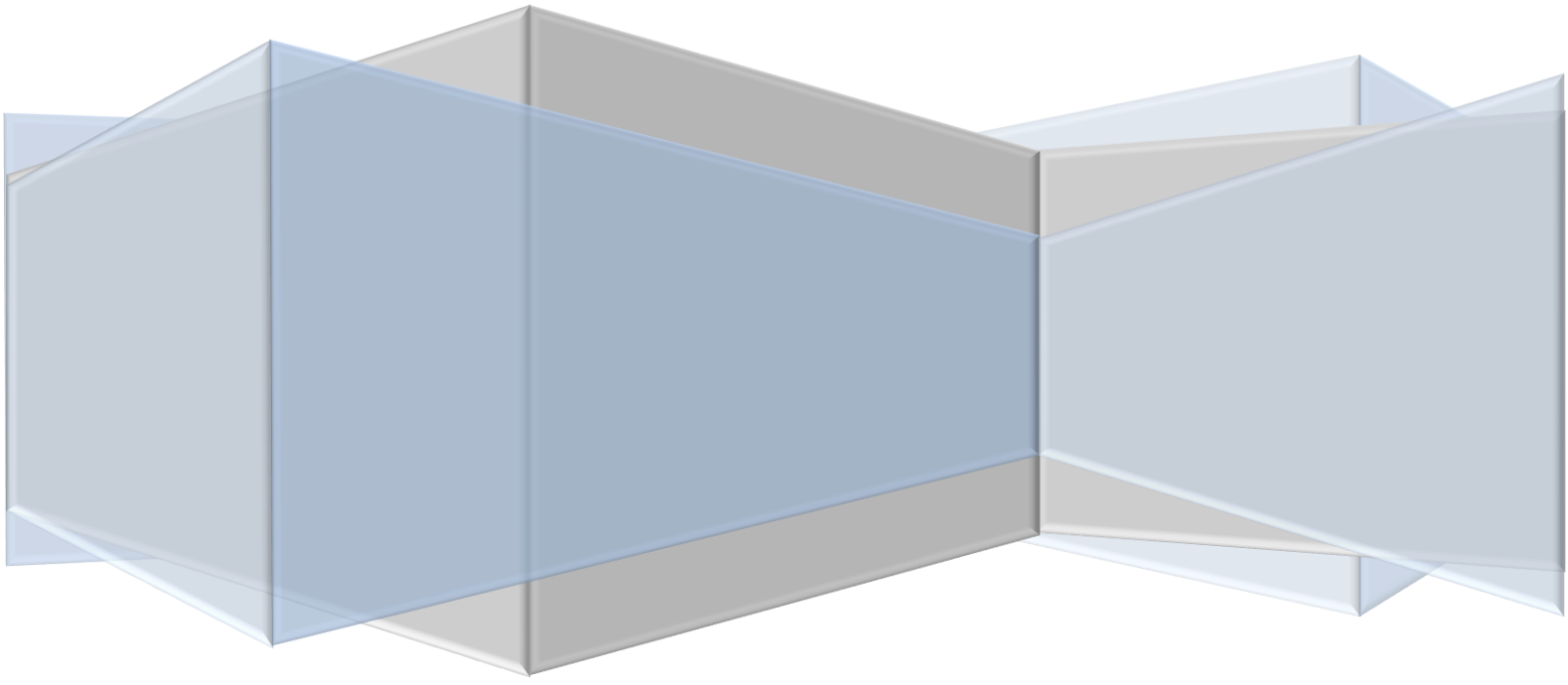
Automation Factory

Chef as Configuration Management tool

Ganesh Palnitkar



Chef Recipe:

- The Recipe in Chef CM tool, is a fundamental unit for applying config changes on the remote infra in Chef Configuration Management.
- The Recipes are written in Chef's own DSL that is based on Ruby.
- Recipes have resources used in it which can be Native and/or Custom in nature.
- Recipe can also be used to act using the collected information like the Facts collected using 'Ohai'. Thus, the Facts collected from different system can have different values and these values can make the Chef recipe to act differently on these systems.
- Recipe can depend upon other Chef recipe or can be a dependency for other recipe.
- Recipe have to stored in a Chef Cookbook.
- In a recipe there can be multiple resources used and applied changes to. Chef implements these resources actions in the same order as these are defined in a recipe.
- If a certain resource implementation fails for some reason, then the next resource will not be applied. This is a implicit order and we do not need to define this dependency.
- Recipe file will always have an extension as '. rb' (stands for ruby block)

Recipe example:

```
apt_repository 'powershell' do
  uri https://packages.microsoft.com/ubuntu/#{node['lsb']['release']}/prod
  components ['main']
  arch 'amd64'
  key 'https://packages.microsoft.com/keys/Microsoft.asc'
  action :add
end
```

```
package 'apt-transport-https' do
  action :upgrade
end
```

- The recipe in example is more usable and modular as it uses a variable stated in ruby syntax
- Values for components is in form of an array and can have multiple elements.
- The 2nd resource has direct dependency on the first action, so if the first one fails for any reason, the second tasks will not be implemented.
- In the 2nd resource the default action is in fact 'install'. So as we want it to upgrade an existing installed version, we are using the value of action as 'upgrade'.
- There are always some default action values for every resource.

Some of the default actions in Chef are `:enable :install :start.` While the action value is defined, it is implemented in the same order as it is assigned. e.g.

```
action [:enable, :start]
```

Cookbooks:

- A valid cookbook will always have at least one recipe file. The recipe may not contain a resource but the recipe file has to be pre4sent in order to be a valid cookbook.
- Attributes: attributes are used for overwriting existing values on a node. So these can be the variable values the we want to replace on the target node.
- A cookbook can also have a files folder inside which we can have those files that are needed to be copied on to the targeted node.
- Templates: this is a folder in the cookbook folder structure inside which we can keep templates files that contain queries / values that can be replaced on the targeted node.
- Metadata: A file which has details about owner, version information about the cookbook.
- Supermarket: Is a store on cloud that has prewritten cookbooks that we can use inside our custom cookbook.
  The cookbook folder structure would look like this,

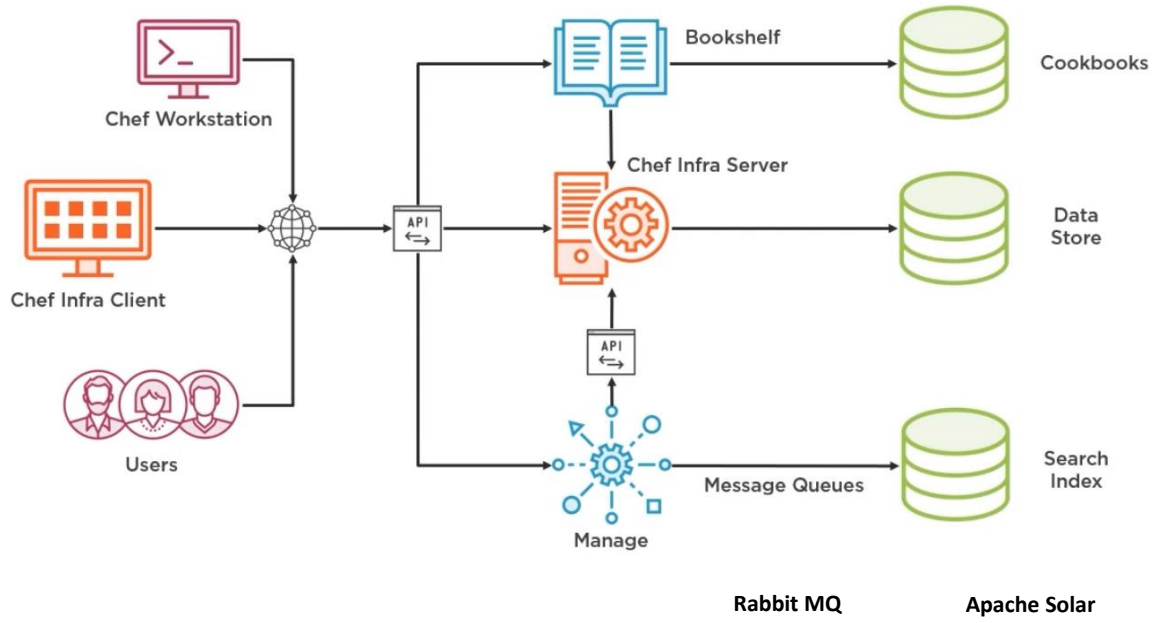
Here, Cookbook is the name of cookbook. Like this one can multiple folders with different name that has a similar folder structure. All cookbook folders can reside inside Cookbooks folder name.
The cookbook will also have Version control repo details.
As a practise in most cases, we may not define resources in the default recipe. Instead, you can write separate recipes inside the same cookbook and call the recipe with the name and cookbook identity. This means we can call recipe file that is in a different cookbook we well.

e.g.

```
include_recipe ‘packages::default’
include_recipe ‘packages::webserver’
```

Understanding Chef-Infra-Server:

The Chef Infra server was originally called / referred to as Chef server, but as recently some of the core architectural components have been rebranded and thus the Chef Server as well. **The** purpose of the server or the Infra-server is still the same, i.e., a central hub for all configuration code and data for your organization.
The **infra server** is where all cookbooks, policies, metadata, about nodes which are under active management.
Each node to be managed is installed with a **client** application and is registered with the infra server. We can have more than one infra servers in the environment but the client is registered against only one server. Once the node is registered with a server, the client (node) makes the request to pull the configuration data from server. The node **metadata** saved on the server is indexed and thus searchable.



These all components can be allowed to run on one single server or we split it to run on separate servers.

Chef-Workstation:

This is a single package application platform that contains multiple development components that we can use while working with code development and uploading it to Infra server.

The **Chef-Workstation** replaces the earlier version components such as **Chef-dk** and add a new component such as **Chef-run** and **InSpec**. Chef-run allows us to run configuration changes directly on the remote nodes whereas InSpec allows us to test the configuration code before implementing it on remote nodes.

Chef-Workstation is available for installation using Choco on Windows or Homebrew on MacOS or even is available in a container and can be launched in Azure cloud Shell.

Chef-Workstation also includes, **Chef-client** that is used for Ad-Hoc config mgmt implementation and testing Cookbooks locally.

Also includes **Ohai**, A tool used for collection of System Configuration data on nodes or generating profile of the nodes on which it runs and profile the infra data in JSON format.

Also has **Chef** tools that can be used for generating new chef code, recipes, cookbooks, and other assets and manage the local config data on self.

Knife is a tool that is part of Chef-Workstation to provide connectivity between the workstation and the Chef-infra server, upload Cookbooks to server, query or index data on server, initiate client installation on remote node, etc.

Chef-run is a tool that can be used for performing ad-hoc node management where the nodes need not be a managed node (installed with client app). This will help in eliminating the need of the chef-infra-server or chef-client on the remote node. **Chef-run** can connect to remote node using SSH on non-windows or WinRM on Windows.

This also includes the **Test-Kitchen**, a tool to provide a testing platform for testing cookbooks, config code on multiple platforms that independent of live environment.

InSpec is a tools for security and Compliance testing where we can write the code for testing code compliance for security.

Download Chef-workstation from the link, https://www.chef.io/downloads/tools/workstation?os=ubuntu

And install it using the command,

```
$ sudo apt-get install ./chef-workstation_22.7.1006-1_amd64.deb
```

Once installed, run the command, `$ chef –version` command to get version info.

```
vagrant@node1:~$ chef --version

Chef Workstation version: 22.7.1006
Chef Infra Client version: 17.10.0
Chef InSpec version: 4.56.20
Chef CLI version: 5.6.1
Chef Habitat version: 1.6.420
Test Kitchen version: 3.3.1
Cookstyle version: 7.32.1
```

Now, try running below commands,

```
$ chef --help
```

This will list out all commands that we can work with Chef.

Commands listed have further sub commands to it like,

```
$ chef generate
```

```
Available generators:
  attribute   Generate an attributes file
  cookbook    Generate a single cookbook
  file        Generate a cookbook file
  generator   Copy Chef Workstation's generator cookbook so you can customize it
  helpers     Generate a cookbook helper file in libraries
  input       Generate a Compliance Phase Chef InSpec Input file
  policyfile  Generate a Policyfile for use with the install/push commands
  profile     Generate a Compliance Phase Chef InSpec profile
  recipe      Generate a new recipe
  repo        Generate a Chef Infra code repository
  resource    Generate a custom resource
  template    Generate a file template
  waiver      Generate a Compliance Phase Chef InSpec Waiver file
```

Next run command Ohai on local system. This will pull the information about hardware, network, OS in form of JSON output or can be customized as well.

$ ohai

Next try running the $ knife –version command. Knife allows you to connect with chef infra server and work with cookbook, recipes etc.

```
$ knife --version
```

Next is the chef-client. Chef-client is a command to invoke the chef client in implementation mode. We would run this command on managed node where there will be a recipe file available in the folder where from we are running the command.

`$chef-client --version` or `$ chef-client --help`

Next try running the kitchen command as kitchen –help this command helps in deploying a test environment for testing cookbooks, recipes.

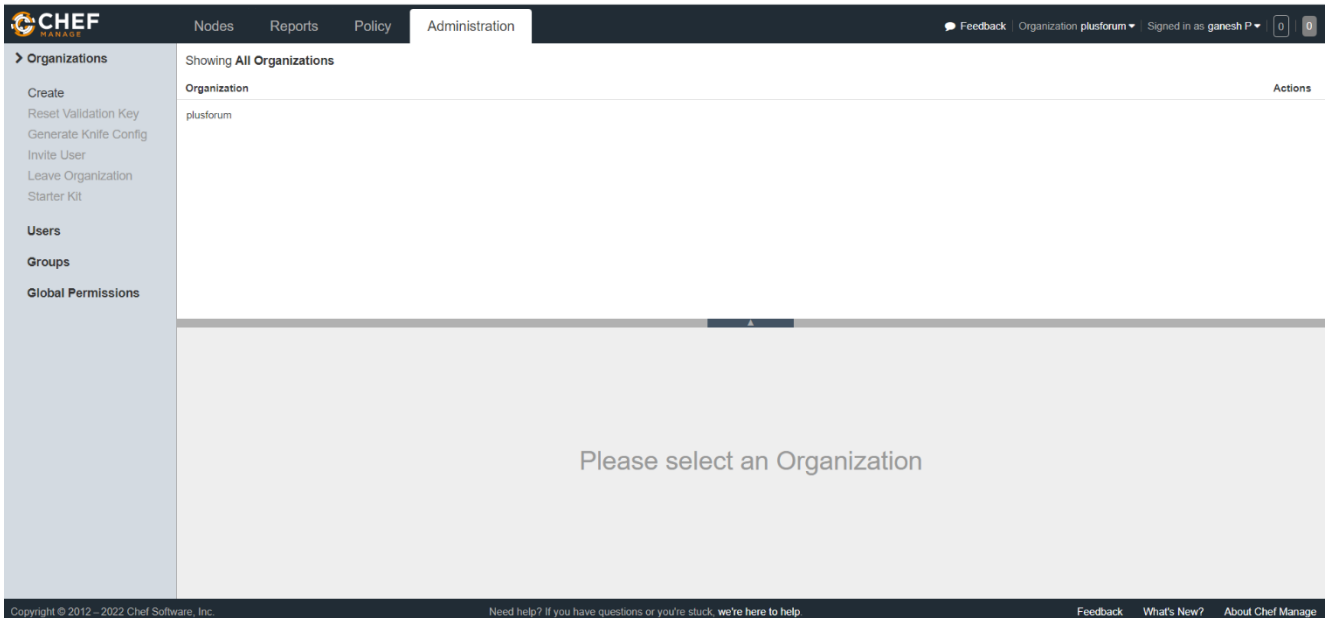
```
$ kitchen --help
```

Try running the `$ inspec` command to invoke the utility. This is used for compliance and security validation.

Working with Hosted Chef:

https://manage.chef.io is the link for getting started with the cloud hosted chef infra server.
On the infra server, all configuration data, node metadata, cookbooks, roles, environments are stored in a database that is accessed using a web console. We also have to create a

parent component called as organization under which this config data is stored. We can have multiple organizations created, but components registered under one org can't be accessed in other org.
On the infra server every legitimate node is registered by accepting SSL certificate. This is done by the **Chef-Infra-Client.** Once registered the Chef-client on the node will get invoked every certain interval to apply configuration / desired state configuration on self. Certificates are org specific. The DSC is stored on the Infra server and is applied to a specific node using roles and environment objects. The **infra-client** pulls the cookbook, compiles supporting resources and libraries and generates and run-list. The DSC is applied onto the node and converged to the desires state as defined on Infra-server. The updated state of node is reported back to the Infra-Serer thus storing the current state of the Node.



We can take this further to add the nodes to the chef server.

What's a Node?
A node that can be managed using Chef can be any physical machine, virtual machine, desktop, any operating systems that support chef-client.
This can even be a cloud platform like AWS or AZURE which can be managed using knife utility.
We can also manage Networking devices like Routers / switches that can be managed remotely.
Docker container environment can be managed using chef as a node.
Look into https://docs.chef.io/nodes.html for more information on node configuration options.
How to Manage a Node?

Ohai installed on the node as part of chef-client, collects and sends data about the node to the Infra server.
Assigning Cookbooks and actions to the nodes is done using roles. Role is a component on Server having that can be assigned to a node, thus assigning cookbooks to the node.
Run list is dynamically generating action list which is applied to the nodes.

Run_list makes sure that all resources added are applied with changes. Environments are high level objects that are also used for declaring cookbook names we want to

On the **Chef-infra** server the node information is seen under the Nodes tab. Select one of the node and we will see details of the selected node with Attributes. Attributes are populated using the Ohai utility.

The attributes of a node can be queried from workstation using the knife tool.

```
$ knife search –help
```

Here knife is used for querying the chef-infra server and the node attributes stored on it.

Kitchen tool for testing Chef code like cookbooks.

- Kitchen is an automated framework for testing cookbooks.
- The Testing framework and testing environment creation files is stored as code based assets.
- Test environment deployed are temporary environments and are dynamically deployed.
- The testing is done using the InSpec and ServerSpec tools.
- Testing environment deployed uses VirtualBox, or vagrant with any other virtualization. Environment can be deployed on a cloud environment as well.
- The Kitchen tool makes it easy to test the code making the Chef config management code more reliable and a stable platform.

Kitchen also establishes the **test-driven development approach**.

- Kitchen as a test environment enables early detection of code problems making the code more reliable, high in quality.
-

Environment variable: Some of the very important steps while configuring Chef in a Lab environment.

This might not be applicable while in an enterprise environment.

- Update /etc/hosts file with IP-address and corresponding hostname in the lab environment.
- Install git to manage version control

1) **ssl fetch:** In order to run knife command on Chef Workstation to connect to Chef Server, certificate has to be fetched from server and stored on workstation.

   Run below command to fetch the certificate.

   `$ knife ssl fetch` ….. By default the certificate file gets stored on workstation at `/chef-repo/.chef/trusted_certs.`

2) **Local execution:** For executing / testing the cookbook in local mode use below command on the Chef workstation.

   `$chef-client --local-mode --runlist 'recipe[httpd]'` … this will apply the cookbook recipe on the workstation.

3) **Knife.rb**: In order to use the knife tool, there needs to be a 'knife.rb' file used to configure the tool stored on workstation under .chef folder.
   A typical knife.rb file has below entries in it.

```
current_dir = File.dirname(__FILE__)
log_level                :info
log_location             STDOUT
node_name                "ganeshhp"
client_key               "#{current_dir}/ganeshhp.pem"
chef_server_url
"https://chefserver/organizations/autofact"
cookbook_path            ["#{current_dir}/../cookbooks"]
```

4) **Environments:** Creating environment using the '.json' or '.rb' file. Below example shows environment created using '.rb' file and using the command to create environment on Chef Server.

```
# /environments/development.rb

name "development"
description "this file is for creating development environment"
cookbook "apache", "=0.1.0"  …. Or
```

```
cookbook_versions ({
  "cooksbook_name" => "=<version>",
    "cooksbook_name" => "=<version>"
})
```

```
$ knife environment from file development.rb
```

5) **Roles:** Creating Roles using the .rb or .JSON file. Below example shoes role creation using .rb file and using command to create the role on the chef server.

```
# /roles/webserver.rb
  name "webserver"
  description "this file is for creating the webserver role on
  the server"
  runlist "recipe[apache]","recipe[mysql]"
```

```
$ knife role from file webserver.rb
```

6) **Cookbook Dependencies**: While writing a cookbook in order to express dependency of a cookbook into our custom cookbook, this is mentioned in the 'metadata.rb' file. In the metadata file,

```
depends 'cookbook_name', '~> <version_number>'
```

ex.,

```
depends 'apt', '~> 0.1.0'
```

This will cause the cookbook to download using Berks utility for the mentioned version number.

Once the metadata file is updated, update the defaults.rb file with code to include the dependency with below line of code.

```
include _recipe '<cookbook_name::default>'
```

Ex.,

```
include_recipe 'apt::default'
```

7) **Generate recipe:** A recipe can also be generated like we create a cookbook using the generate command.

```
$ chef generate recipe <recipe_name (path optional)>
```

8) **Generate attributes:** Attribute can be generated using the generate command.

```
$ chef generate attribute <attribute_name (path optional)>
```

Once create attribute the 'default.rb' file inside attribute can be updated like the one mentioned below.

Once created update the attribute file as mentioned below.

```
default['<cookbook_name>'] ['attribute_resource_name'] =
'attribute_value'
```

ex.,

```
default['webapp']['user'] = 'web_user'
```

#A recipe for installing and configuring apache2 on ubuntu.

#install apache and start apache service

```
apache2_service '<webapp_name>' do
   mpm 'prefork'
   action [:create, :start]
end
```

add the site configuration

```
apache2_config '<webapp_name>' do
  instance '<webapp_name>'
  source 'webapp_name.conf.erb'
  notifies :restart, 'apache2_service[webapp_name]'
end
```

create a document root directory

```
directory '/var/www/webapp_name/html/' do
  recursive true
end
```

create a default home page

```
file '/var/www/webapp_name/html/index.html' do
  content '<html>Hello world!</html>'
  owner 'user_name'
  group 'group_name'
  mode '0644'
  action :create
end
```

9) A **community Cookbook** is available to control the firewall rules.

```
depends 'firewall', '~> 2.3.0'
```

Using the firewall resource in chef, we can modify the firewall rule. Use below syntax.

```
firewall_rule 'http' do
  port 80
  protocol :tcp
  action :allow
end
```

10) Using kitchen testing tool:

First run the kitchen list command… this will use the kitchen.yml file and list available vagrant boxes that can be used for creating test environment.

`$ kitchen list` - List all available platforms
`$ kitchen create` - Provisions a vagrant box as an test environment
`$ kitchen converge` - test the chef code on the kitchen box.
`$ kitchen destroy` - to delete the kitchen test instance.
`$ kitchen login <instance name>` – helps to login to the running kitchen test instance and then test / run commands / output manually.

11) Bootstrap 'EC2' instance using below command,

```
$ knife bootstrap <node FQDN / Hostname> -x <username> -P
<password> --sudo –r "recipe[cookbook_name]"
```

In order to bootstrap EC2 instance, generate the public key on the chef workstation and copy the public key file to the EC2 instance and save it to the `'~/.ssh/authorized_keys'` file location.

12) Conditional statement to choose Platform and take appropriate action.

```
case node['platform_family']
when 'debian'

  package_name = "apache2"
  service_name = "apache2"
  doc_root = "/var/www/"

when 'rhel'

  package_name = "httpd"
  service_name = "httpd"
  doc_root = "/var/www/html/"
end

package "#{package_name}" do
  action [:install]
end
service "#{service_name}" do
  action [:enable, :start]
end
template "#{doc_root}/index.html" do
  source 'index.html.erb'
end
```

13) Updating Node using 'environment_set' option'

```
$ knife node environment_set <NODE_NAME> <ENVIRONMENT_SET>
```

14) Compare all cookbook versions for all environments.

```
$ knife environment compare –all ….
```

15) Creating environment using environment create option.

```
$ knife environment create <environment name> --description
"<description>"
```

16) Uploading chef cookbooks to Artifactory repository (Virtual / Custom Chef Super market)
- Create new repository on Artifactory selecting repository type as Chef.
- In order to allow your chef workstation to connect to Artifactory repo, use below commands and configuration.

General

In order to configure your Knife client to work with Artifactory, you need to edit its *knife.rb* file (which can usually be found under <*user-home-dir*>*/.chef/*) and add a reference to your Artifactory Chef repository as a "supermarket_site". For example:

```
knife[:supermarket_site] = 'http://localhost:8081/artifactory/api/chef/chef-local'
```

To support authentication which may be required by Artifactory, you need to install the *knife-art* plugin. For installation instructions, please refer to the Artifactory User Guide). Once the plugin is installed, you can specify your credentials at the beginning of the url as shown below:

```
http://admin:AP78qTq9sZidVod8TTfbeqf9QXF@localhost:8081/artifactory/api/chef/chef-local
```

Deploy

To deploy a cookbook using Knife, run:

```
knife artifactory share <cookbook-name> [CATEGORY]
```

Resolve

To install a cookbook using Knife, use the below command:

```
knife artifactory install <cookbook-name> [VERSION]
```