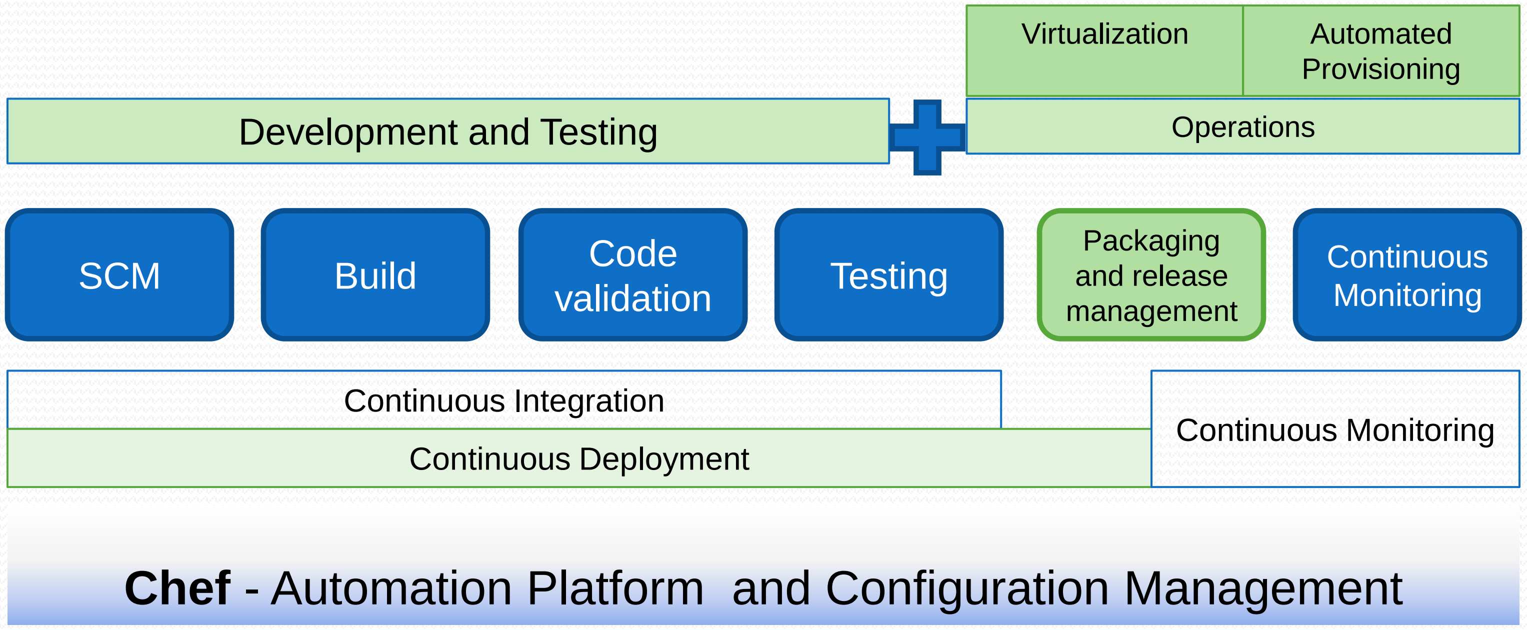
Chef Essentials

By Ganesh Palnitkar

Brief DevOps Overview



Need of a Configuration management / Automation Tool

- Enterprises experience changes to the application environment because of changing business requirements.
- New server added, Applications and App updates are deployed at ever increasing frequency.
- Replicating application environments quickly and managing the same becomes biggest challenge for operations team.
- Challenges like datacenter upgradation, cloud, hybrid environment architecture also pose as big hurdles.
- Scripted automation is not always useful as it involves person dependency. Team changes can hamper stability. Creating an Environment manually, needs lot of home work, understanding the dependencies while installing each dependency.
- Every application to production release becomes a nightmare.
- Configuration management tools help in bringing in scalability, remove human dependency, quick deployments. Quick provisioning, version controlled server environment.

Configuration Management tools

- Chef, Puppet, Ansible, Saltstack

Ansible	Chef	Puppet	Saltstack
SSH-based communication	Used SSH with powerful knife tool	Most mature interface	Input, Output and Config are very consistent – Uses YAML
Agentless operation	Chef-Client resides on Nodes	Agent running on Nodes	Strong community
Shorter learning curve with YAML	Code driven approach gives better control	Simple installation and initial setup	High scalability in the master model with Minions
Simple playbook structure	Rich collection of modules (Cookbooks)	Strong reporting capability	
Much more streamlined code base	Not a simple tools, large code bases	Model driven approach, lesser control	Difficult to setup
Less powerful than other tools	Steep learning curve	Steep learning curve (used RUBY+DSL)	Documentation is challenging
Struggles with performance speed at times	Doesn't support PUSH functionality		Not a great support to Non-Linux OSs

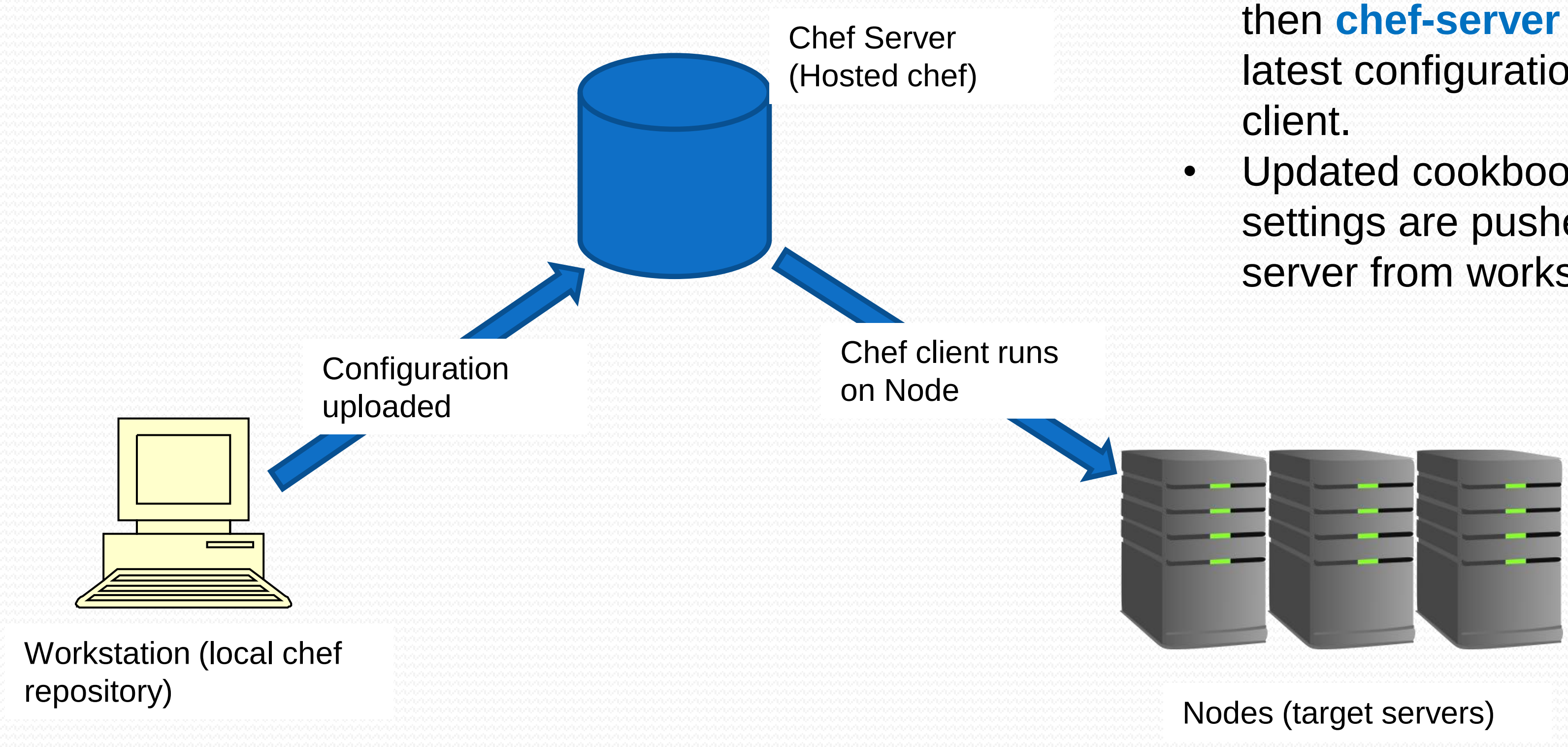
Chef as
Configuration Management

What is Chef

Chef is an Orchestration and Automation platform.

- Moderate learning curve, can be used for Configuration management, App deployment, Workflow Orchestration, Orchestrate Application lifecycle.
- Using Recipes and cookbooks, we can define the action that we want to take to create an environment, deploy an application or a file and so on.
- Chef architecture involves a **Chef server**, **Nodes** and **Workstation**. Chef server is the one where recipes and cookbooks are stored which contains the configuration for targeted nodes. **Workstation** is the machine where the Configuration (cookbook and recipes) is created and pushed to Chef server to be run on the targeted nodes.
- Workstation is equipped with a CLI tool called knife.
- Chef-Client is a program that runs on each node.
- For pre-written cookbooks, one can refer to supermarket site. Incase of Puppet this is available on Forge.

Chef Architecture



- **Chef client** periodically pulls chef server to see if there are any changes to the cookbook.
- If configuration is changed then **chef-server** sends latest configuration to chef-client.
- Updated cookbooks and settings are pushed to Chef server from workstation.

Features inside Chef Ecosystem

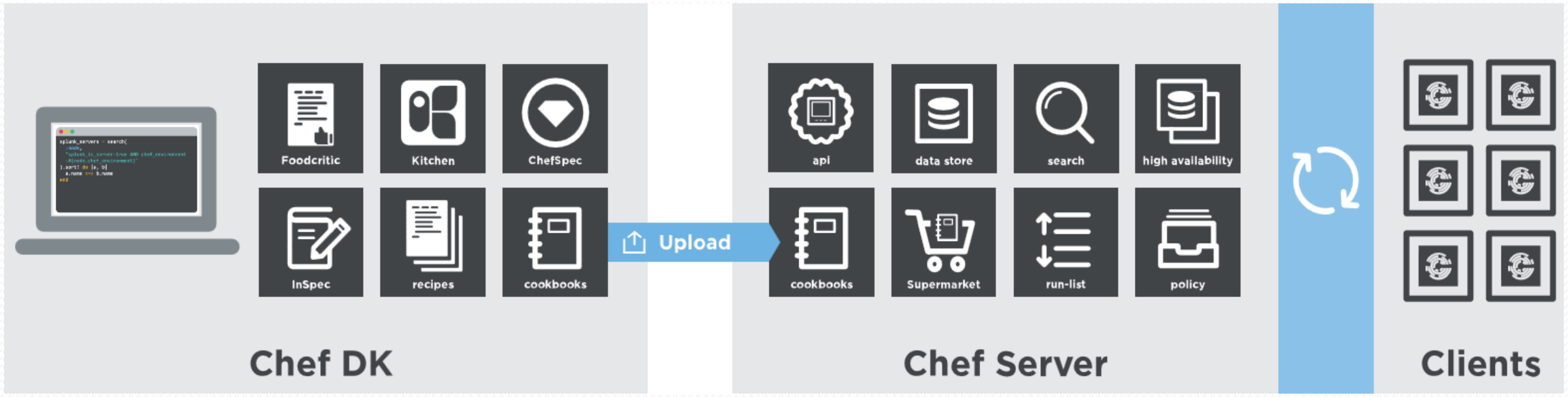
Chef has some interesting tools to test your Ops Code (Chef code, recipes, etc.) before it's moved to production. Chef also has a tools to audit, called 'analytics' to check-back what has gone wrong.

Learnchef.io - Place to learn chef from Chef.io

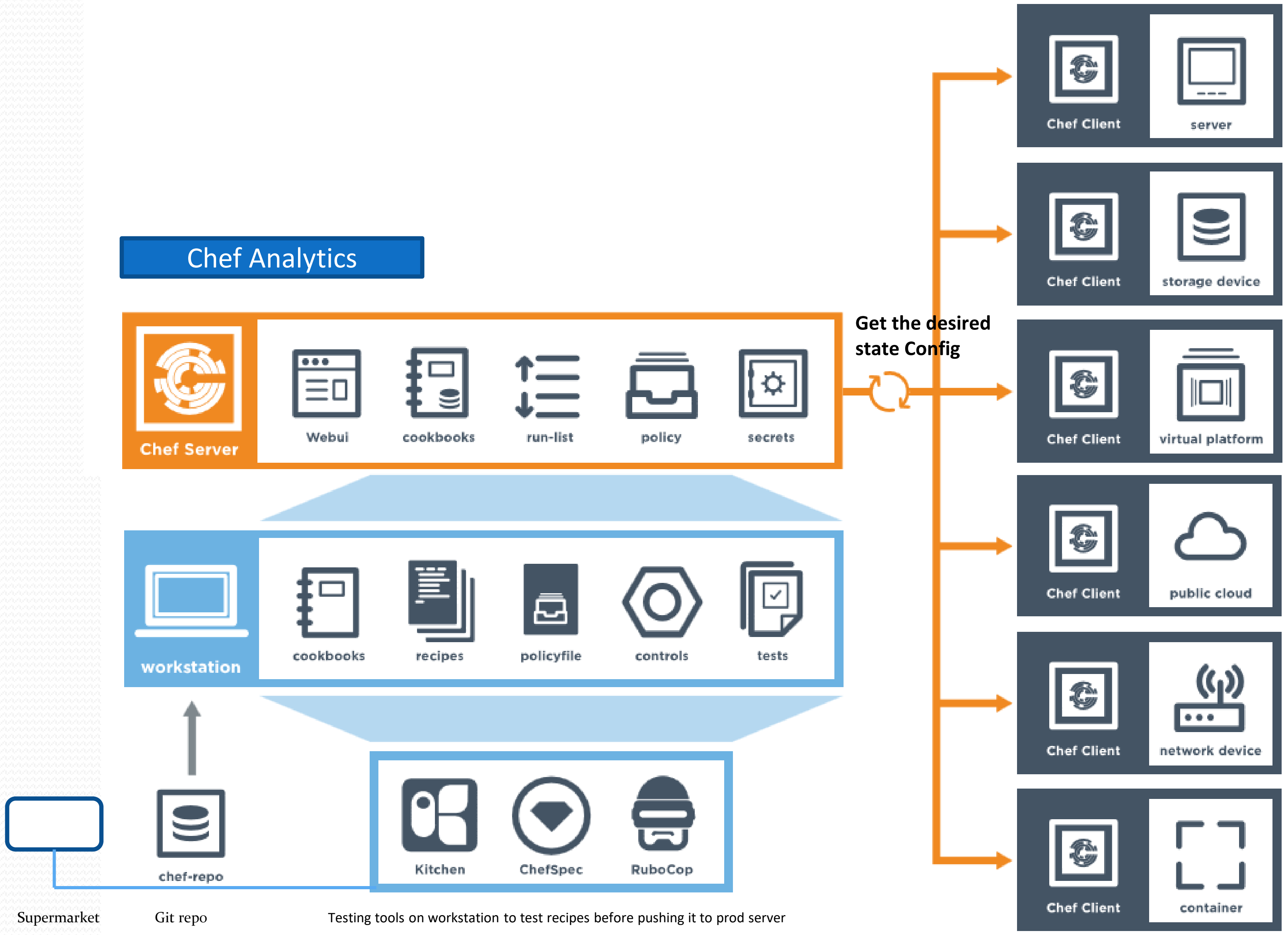
Docs.chef.io - Reference from chef.io

https://docs.chef.io/chef_overview.html

- Hosted Chef offers support for 5 nodes



Chef components



Supermarket Git repo Testing tools on workstation to test recipes before pushing it to prod server

Chef workstation

- The Chef workstation is configured using the Chef Development Kit, ‘ChefDK’.
- ChefDK is developed by Chef Community that contains below tools,
  - 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.
- The ChefDK is available for all platforms (Windows, Linux flavors,).
- Chef Workstation needs personalized key to connect to Chef Server, this is made available from Chef-Server.
- Knife is the tools used to run commands to communicate with chef server.
- Workstation is a dev and test environment before pushing it to the production chef-server.
- Chef-workstation can reside on 64 bit, Windows, Linux or Mac-OS.

Chef Server

- Hosted and Enterprise version.
  - Hosted chef-server version come with free version to manage 5 nodes. Anything above 5 nodes to be manages come in a paid hosted chef server version.
  - Enterprise version come with unlimited nodes manageability and can reside only on 64-bit Linux server OS.
  - Chef-Server provides personalized key for authentication that should be there on the workstation and Chef-client.
  - Managing the nodes (configuring a node and attaching it to environment, and /or a role) is done from the chef-server.
  - Chef-server comes with additional tools like, ‘opscode-manage’ which helps to manage the server and configure nodes using web-interface. This is very similar to the Chef-Server interface that we will see on hosted chef.

Chef Nodes

- Chef node can be any virtual or physical machine or a cloud instance connected on network. This can be a Linux, Windows or a Mac machine and can also be network equipment like a Router, Switch etc.
- Chef node communicate with server over SSH using Chef-client.
- Chef server communicates and controls only those nodes which are authenticated with a authorized certificate.
- The certificate is supplied only once. If the certificate file is lost, the node has to go thru rejoining process again.
- Authentication of node with server is done using ‘bootstrapping’ process run from Chef-workstation using the command,

```
$ knife bootstrap <FQDN or ipaddress> --sudo -x <ssh-username> -P <ssh-password>  -p <ssh-Port> -N
“<node-name> ” -r <run_list>
```

- Chef Bootstrap process install a chef-client on the node, if this is not already present, knife command line tool, ‘ohai’ – the system profiler, certificates supplied by the Chef-server and few more things on the node.
- Ohai the system profiler gathers all the data in JSON format and stored onto the chef-server database and is also made available using search index.

```
$ knife node list.
```

- Every node must have a unique name within the organization. Chef defaults to the FQDN of the node as the node-name.
- Node Object are made up of attributes, (mostly auto discovered using Ohai.)

Chef Nodes – node object

- Node object is made up of attributes like ip-address CPUs etc that are discovered using Ohai. Other chef objects, like cookbooks, roles, environment also contribute to the node object.
- Nodes are indexed and stored on chef-server.
- Node object data can also be exported in JSON format using command,
- `$ knife node show <node-name> -Fj`

```
{
"name": "node1",
"chef_environment": "env",
"run_list": [ ],
"normal": {"tags":[ ]}
}
```

- `$ knife node show <node-name> -a fqdn` ---- to view specific attribute.
- `$ knife search node "*:*" – a fqdn` ---- used for searching a node object with a specific attribute.

About Chef recipes and cookbooks

- Chef is written in ruby, so all configuration are also written in ruby. Incase of puppet there's a separate DS language used for writing modules.
- Every ruby file (recipe) starts with 'do' and ends with 'end'.
- If attributes are not specifically written, it is default to default settings.
- Chef follows principle of idempotency. So if the configuration is in the expected state, chef will not take any action, if not, it will take action to correct it.
- The chef repository contains the resources and infra code that gets applied to the nodes. To create a repository use below command,

```
$ chef generate repo <repo name>
```

- To create a cookbook use command,

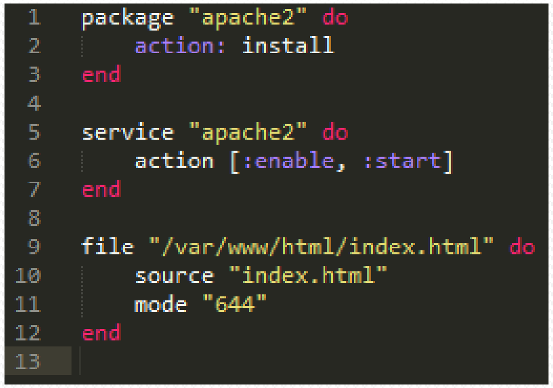
```
$ chef generate cookbook <cookbook-name>
```

- TO create a Template type source the command is,

```
$ chef generate template <cookbook-name> <template-name>
```

- This will create a cookbook with default.rb file which will be the default recipe file. Make changes to this file to address any more recipe files required and or update the ruby code for required action.

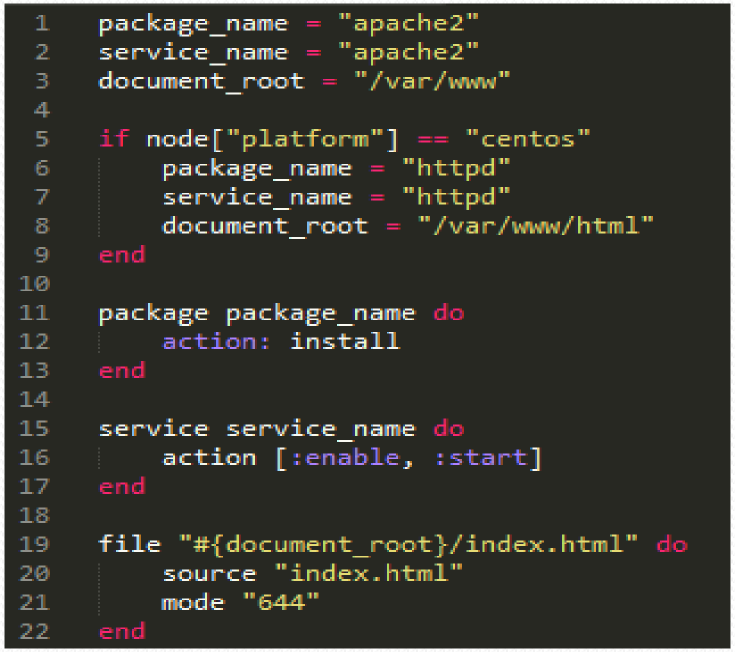
Recipe with simple Ruby



```
package "apache2" do
    action: install
end

service "apache2" do
    action [:enable, :start]
end

file "/var/www/html/index.html" do
    source "index.html"
    mode "644"
end

```

```
$ package ‘apache2’
$ service ‘apache2’ do  ------------------ declaration for a resource
        action [:enable, :start]  -------------------   attribute
    end
$ template ‘var/www/html/index.html’ do
        source ‘index.html.erb’
    end
```

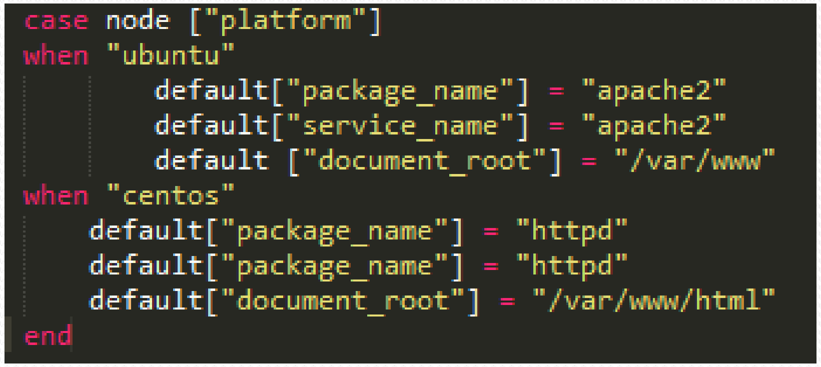
Using variable to simplify



```
package_name = "apache2"
service_name = "apache2"
document_root = "/var/www"

if node["platform"] == "centos"
    package_name = "httpd"
    service_name = "httpd"
    document_root = "/var/www/html"
end

package package_name do
    action: install
end

service service_name do
    action [:enable, :start]
end

file "#{document_root}/index.html" do
    source "index.html"
    mode "644"
end
```

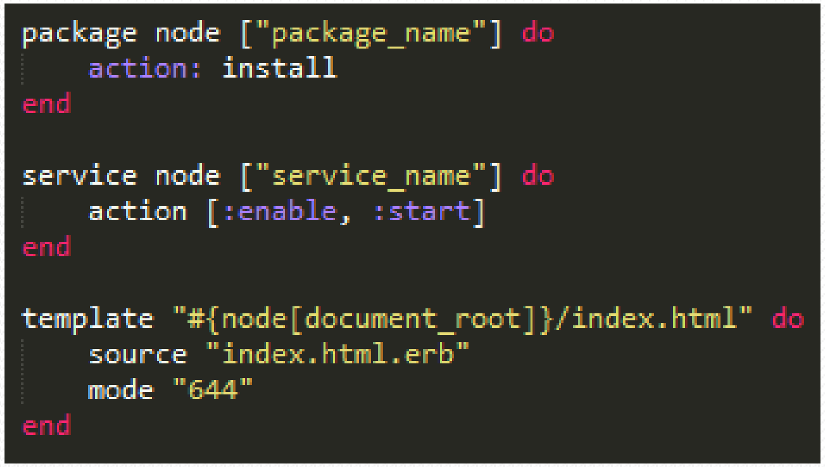
Chef Attributes

By using attribute and creating ‘default.rb’ file inside attributes section in the cookbook we can make the recipe more manageable.



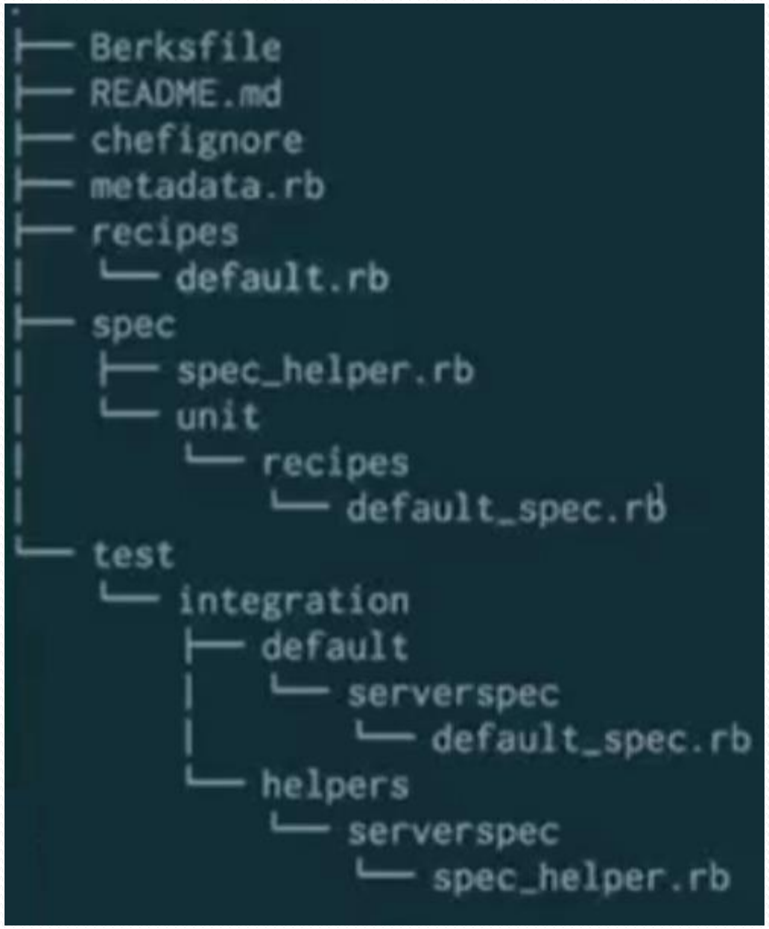
```
case node ["platform"]
when "ubuntu"
        default["package_name"] = "apache2"
        default["service_name"] = "apache2"
        default ["document_root"] = "/var/www"
when "centos"
    default["package_name"] = "httpd"
    default["package_name"] = "httpd"
    default["document_root"] = "/var/www/html"
end
```

Recipe file thus gets more manageable by referring from default.rb from attributes section.



```
package node ["package_name"] do
    action: install
end

service node ["service_name"] do
    action [:enable, :start]
end

template "#{node[document_root]}/index.html" do
    source "index.html.erb"
    mode "644"
end
```

Chef Cookbook



```
├── Berksfile
├── README.md
├── chefignore
├── metadata.rb
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
│           └── default_spec.rb
└── test
    └── integration
        ├── default
        │   └── serverspec
        │       └── default_spec.rb
        └── helpers
            └── serverspec
                └── spec_helper.rb
```

- To run a cookbook the command is,

  `$ chef-client –local-mode –runlist 'cookbook-name'`

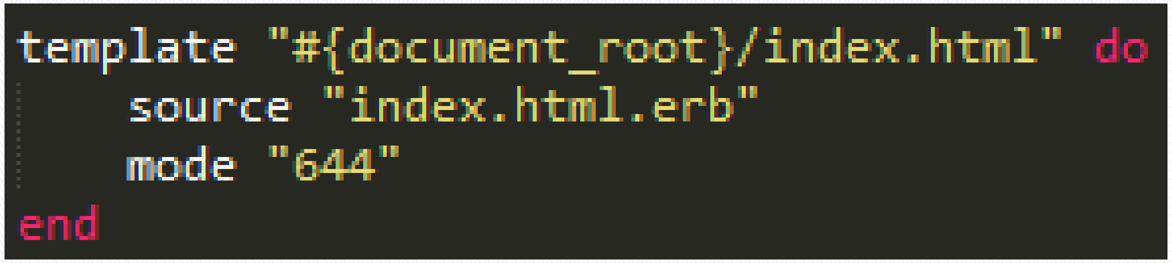
- To generate a template inside cookbook.

  `$ knife generate template <cookbook-name> <template-name>`

- The order in which the cookbooks are executed is in the order the cookbooks are added. We can verify the order by running show cookbook command.

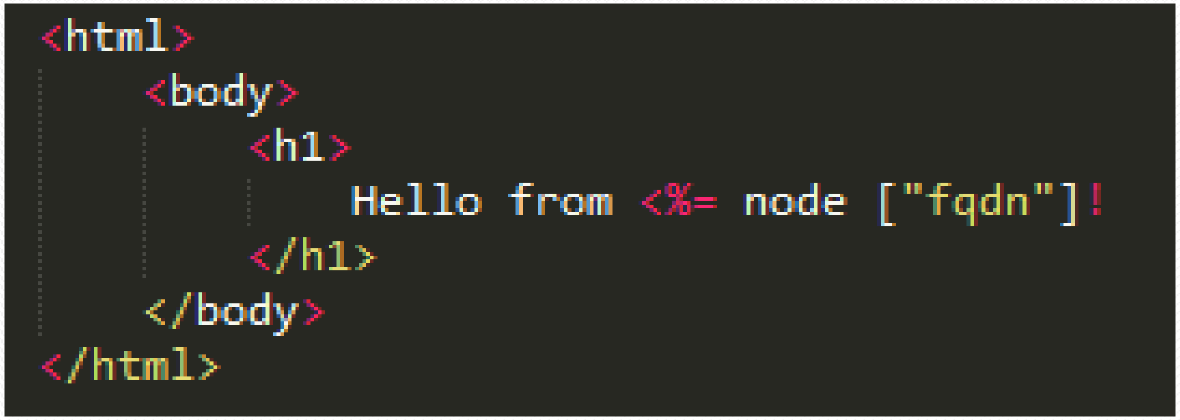
Chef Templates

Using template in your recipe.



```
template "#{document_root}/index.html" do
    source "index.html.erb"
    mode "644"
end
```

To use variables inside a template. The ‘fqdn’ is the attribute of node that we are able to read using the system profiler ‘Ohai’.



```
<html>
    <body>
        <h1>
            Hello from <%= node ["fqdn"]!
        </h1>
    </body>
</html>
```

More about Chef recipes and cookbooks

To download a cookbook from Supermarket use below command,

```
$ knife cookbook site download <cookbook-name>
```

The cookbook is downloaded as a ‘tarball’

To upload cookbook from workstation to Chef-Server use below command,

```
$ knife cookbook upload <cookbook name>
```

To bootstrap a node to chef server, we use below command,

```
$ knife bootstrap <ipaddress or dns-name> --ssh-user <username>
--  ssh-password ‘password’ --sudo --use-sudo-password --node-
name       <hostname> --run-list ‘recipe[cookbook-name]’
```

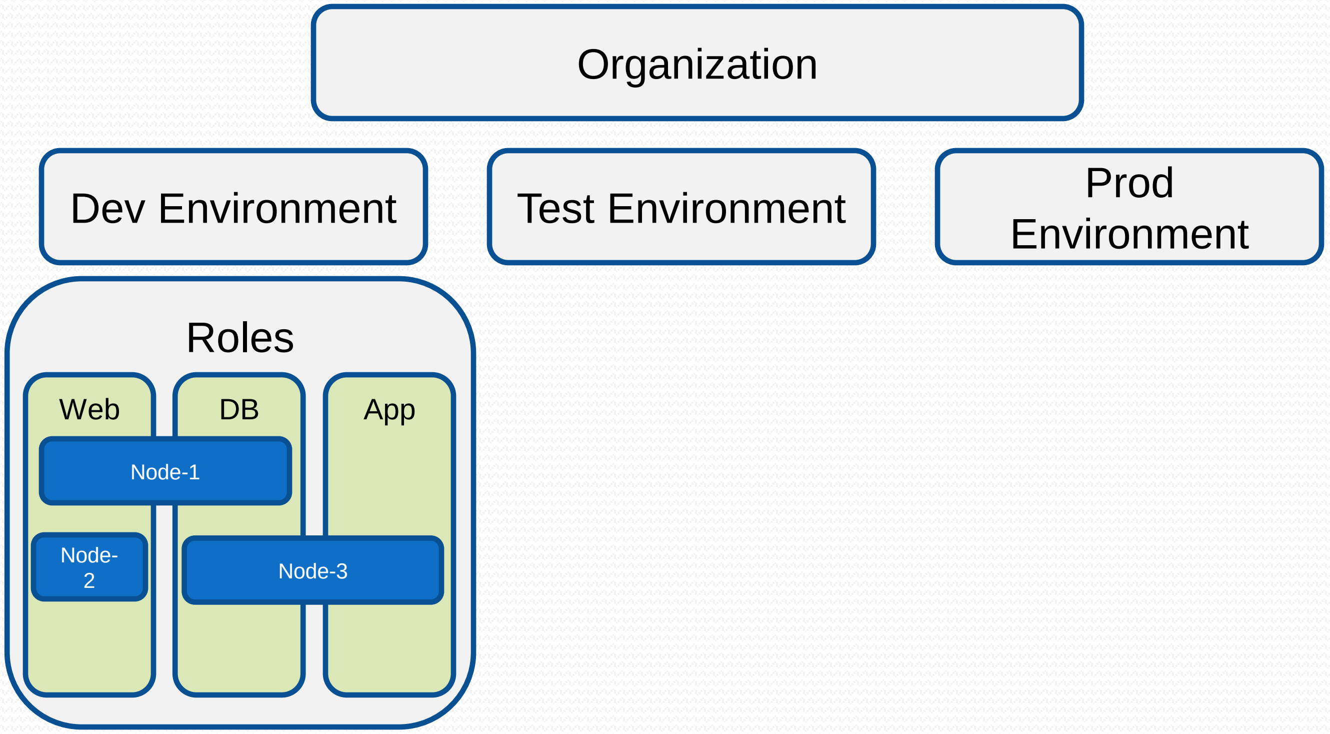
Chef Resources

- Resources are defined with a Type and a Name.

apt_package	execute	mount	ruby_block
batch	file	ohai	script
bff_package	freebsd_package	package	route
breakpoint	gem_package	pacman_package	rpm_package
chef_gem	git	perl	ruby
cookbook_file	group	portage_package	log
cron	homebrew_package	powershell_script	macports_package
csh	http_request	python	mdadm
deploy	ifconfig	reboot	easy_install_package
directory	ips_package	registry_key	env
dpkg_package	ksh	remote_directory	erl_call
dsc_script	link	remote_file	

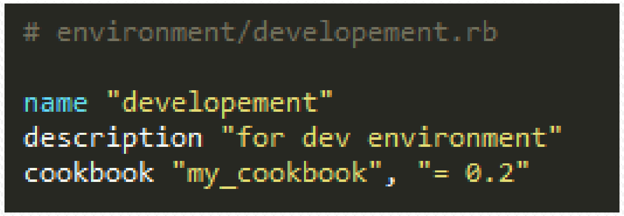
Chef Organization

- While working on enterprise chef-server, we create the user and then the organization.
- Organization represents a development unit, a development unit for an application or suit of applications, department and so on.
- Inside organization we create environments that represents, development, testing, production environment etc.



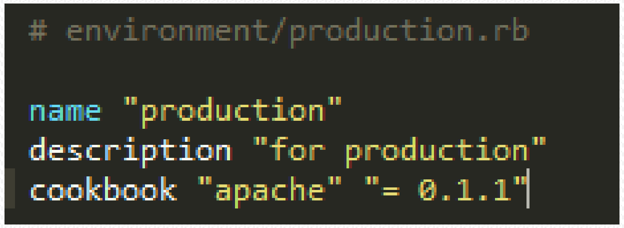
Chef Environment

- On Chef server, inside an organization, environment represents the dev, prod type of environment topology.
- Each environment may have different requirements, like dev environment needs a particular version of Java to be installed, or has all windows or Ubuntu servers and thus need windows installer or Apt installer for Ubuntu and so on.
- Resources can be shared between environment , but no resources can be shared across Organization.



```
# environment/developement.rb

name "developement"
description "for dev environment"
cookbook "my_cookbook", "= 0.2"
```

File to define role for development environment



```
# environment/production.rb

name "production"
description "for production"
cookbook "apache" "= 0.1.1"
```

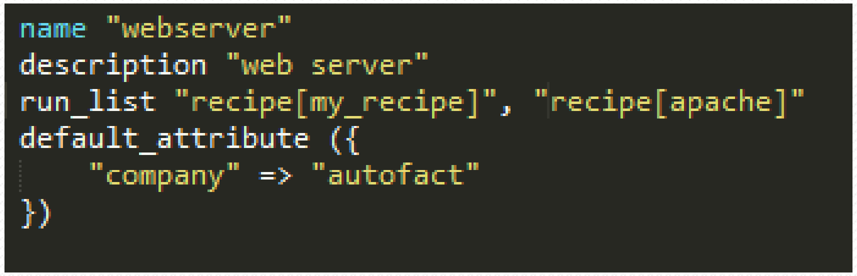
File to define role for production environment

- Environment are applied using command,

  $ knife environment from file development.rb
- The ability in chef to define and apply nodes to a particular environment allows us to rollout changes to the required environment in manageable way.
  - $ knife environment list
  - $ knife environment show <environment_name>

Chef Roles

- Roles helps in encapsulating the run list and attributes required for a server to "be" what it is designed to be.
- Roles make it easier to configure many servers (nodes) identically without repeating each time.
- Each role has a name, description and run_list.



```
name "webserver"
description "web server"
run_list "recipe[my_recipe]", "recipe[apache]"
default_attribute ({
    "company" => "autofact"
})
```

Required fields in defining Role.

- To apply a role we use below command,
  - `$ Knife role from file webserver.rb`
- To delete a role use command,
  - `$ knife role delete <role-name>`
- With the roles applied for a environment / server, we can then manage the recipes in a better way. Instead of applying a cookbook directly to an environment / server we apply role which in-turn applies the required cookbook on the env / server.
  - `$ knife role list`
  - `$ knife role show <role_name>`

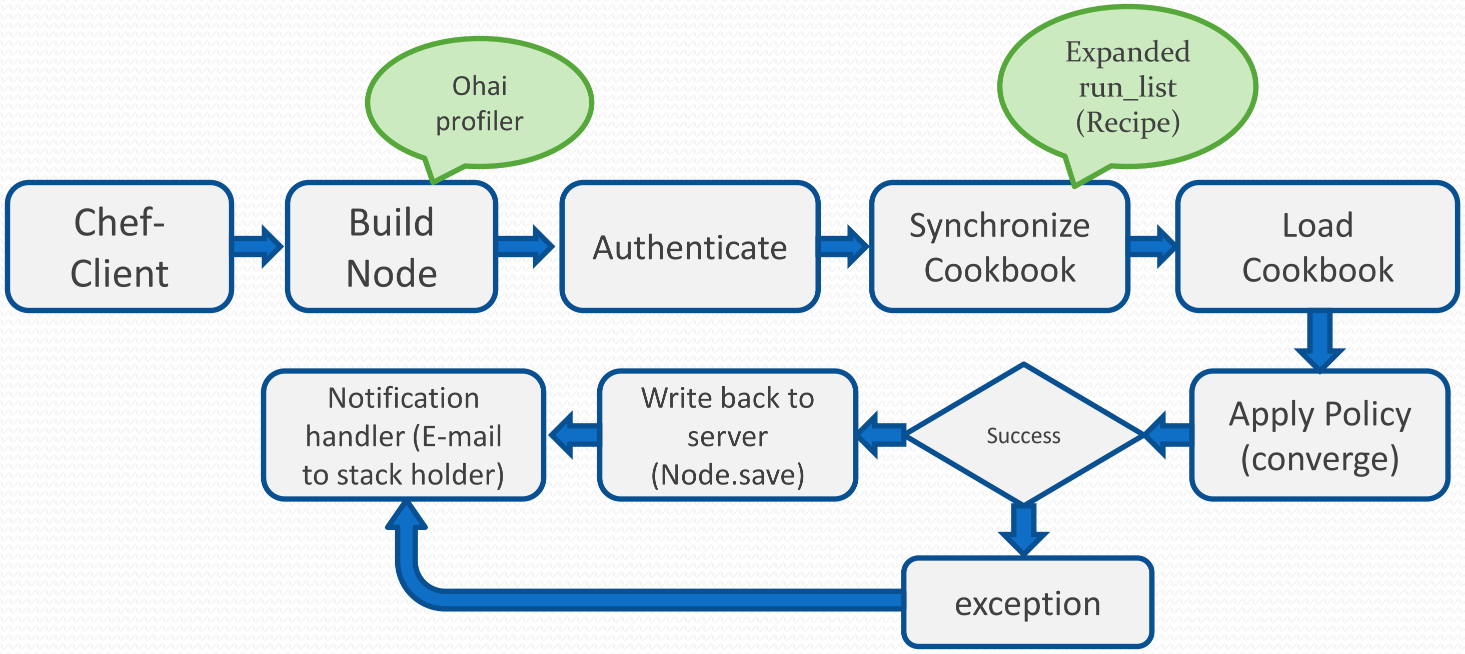
Merge order Precedence

When different attributes, roles, recipes, cookbooks are merged for an environment / organization, the merge order and / or precedence are followed as displayed below.

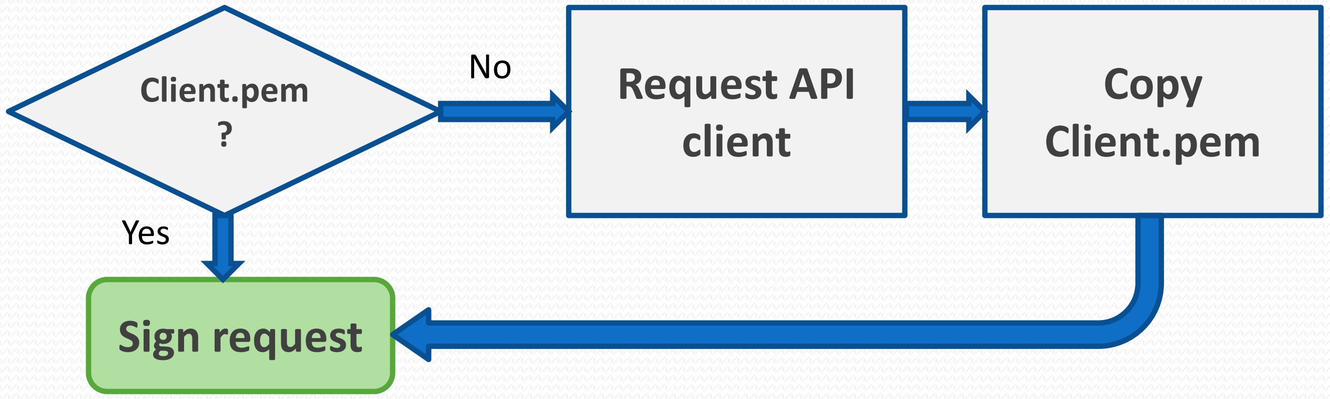
	Attribute file/ node recipes	Node / recipe	Environment	Role
Default	1	2	3	4
Force_default	5	6		
Normal	7	8		
Override	9	10	12	11
Force_override	13	14		
Automatic	15			

Chef Security

- Chef-client and chef servers communicate with each other on SSH.
- In the authentication process, Private keys allows Chef-client to authenticate itself with Chef-Server (Client.pem). ‘Validator.pem’ is now deprecated. At first run the client.pem is supplied to the node during the bootstrap process.
- At first run of Chef-client on a node, the process follows with building the node object(Ohai). Ohai collects the data attributes, like node_name, platform, platform_version, etc. that contributes to the node_object.



Client sign request



During converge process,

- The client compiles the code (resource collection, checks the current state with the policy and if this is not as per policy, executes the resource collection)

```
$ package ‘apache2’
$ service ‘apache2’ do
    action [:enable, :start]
    end
$ template ‘var/www/html/index.html’ do
    source ‘index.html.erb’
    end
```

Additional Chef Tools - Testing you Infra code

- Test kitchen is used to spin a virtual machine
  - Run below command to view available VM instances to spin,
    $ kitchen list
  - Use below command to spin up a instance.
    $ kitchen create
- Test Driven Development (TDD) in Chef.
  - Define the test and set of Unit Test first – define what is expected out of the code / Program.
  - Run the test and start debugging failed test by updating the code.
- Behavior Driven development (BDD) in Chef.
  - Development carried out with business requirements in mind. In this case Desired behavior (Desired state configuration).
- Unit test and Integration test is embedded in the Cookbook structure while it is generated.
- $ kitchen create --- creates an instance, installs Chef client on it and applies the ‘runlist’.
- $ kitchen converge --- applies the ‘runlist’. A way to verify everything is working fine.
- $ kitchen verify --- executes the test suite.

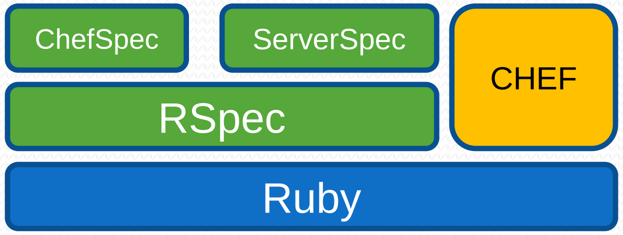
Additional Chef tools

conti.

- Testing your config code.
  - $ kitchen test --- destroys existing instance, creates a **new one**, apply the runlist and run test suite.
  - $ kitchen converge
  - $ kitchen verify

    'Converge' and 'verify' will ensure the policy applies without error on existing node.
  - $ kitchen destroy --- destroys existing test node.

- Remove external dependencies to make the feedback cycle faster
- RSpec is a Domain Specific Language (DSL) , that allows you to express and execute expectations.
- ChefSpec is a helper on top of RSpec that provides help and tools to express expectations about state of resource collection.



Thank You