# DevOps – Chef and related ecosystem

By
ADITYA PRABHAKARA

# Introducing Myself

**Aditya S P (sp.aditya@gmail.com)**
Freelance trainer and technologist

**Boring Stuff about me:**

- 14+ years of experience in development and training
- Started with Java, moved to Android and now working on Big Data Technologies

**Interesting Things about me:**

- Actually Nothing !

# Getting to know you

## Show of hands please!

- What is the general development experience of this group
  - 0-2 years, 2-5 years, 5 and above
- Any Sys Admins in the group ? Any Developers in the group?
- What area are you currently working on?
  - Java Programming, Python, Unix
- How many of you know Ruby?
- How many of you already know chef?
- Why are you attending this session ?

# Agenda

- Chef and its ecosystem
  - Chef DK
- Vagrant, Git, Jenkins, Dockers

# Chapter: Problem statement

Chef

# 2003 - 2010

**Technology stack**

**Internally consumed**

Solaris

Java/ J2EE

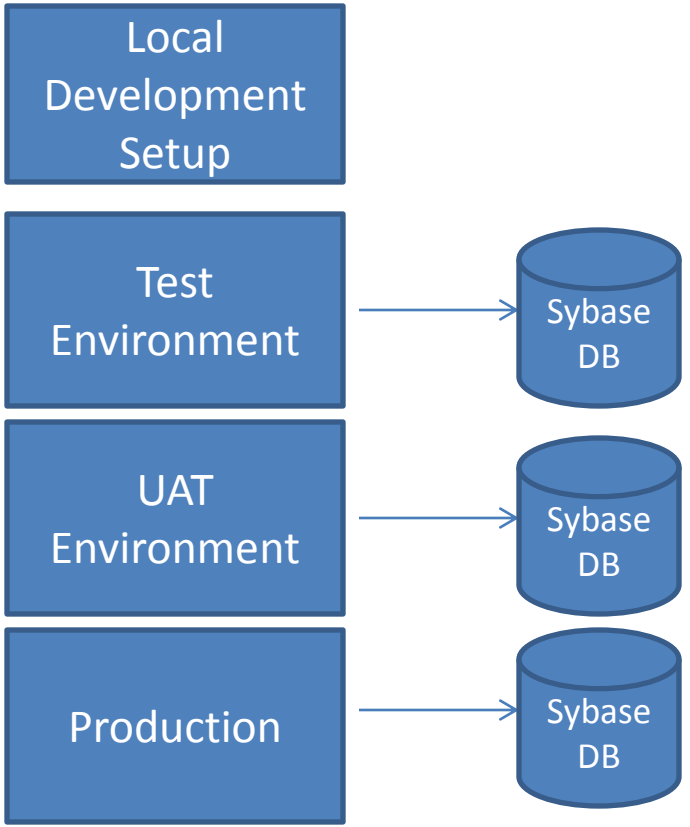
3-tier architecture with scaled up machines

War Application

Shell scripts for batch

- Reconciliation
- Report generation
- Wrong trade data
- Health check

Database on Sybase



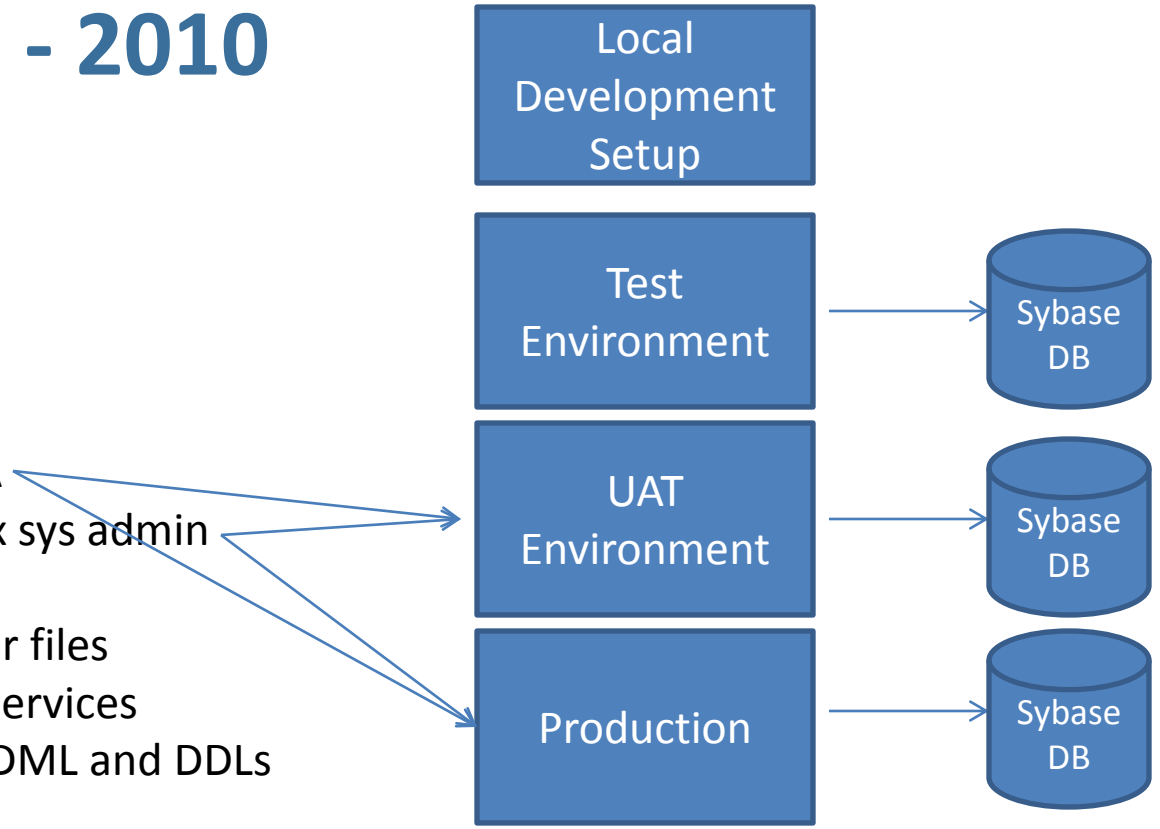
# 2003 - 2010

**Deployment**

A run sheet to DBA
A run sheet to Unix sys admin

Manual copy of war files
Manual restart of services
Manual run of DB DML and DDLs

A roll back script



Chef

# 2010-2012


**Technology stack**

**Product**

Java/J2EE

N-tier architecture

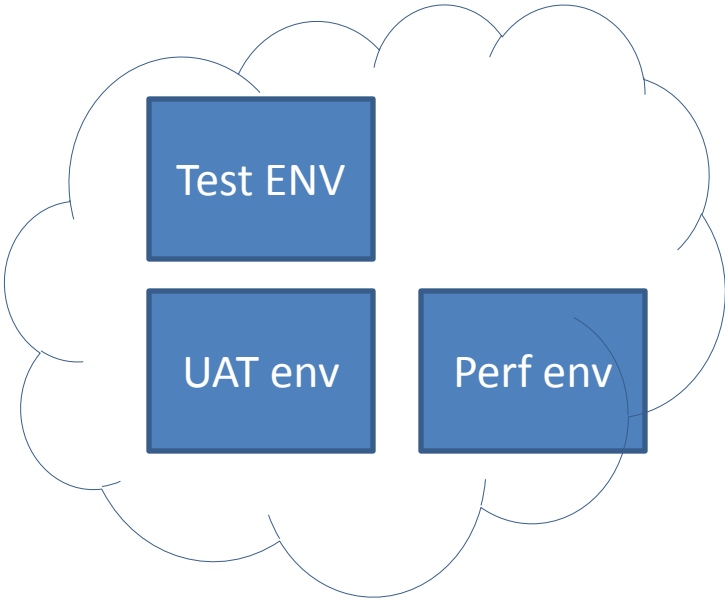
Load Balancers

Queues

Designed to handle high customer load

Multiple supported OS platforms (Linux, Windows)

Multiple supported databases (MSSql, Oracle)




X n Platforms

# Current Consulting

**Technology stack**
**Product**
Microservices on cloud

Nothing else to mention!

# What are the patterns you observe?

- Multiple technologies
- Emerging architectural types and associated challenges
- Growing deployment complexities
- Not enough to have manual instructions

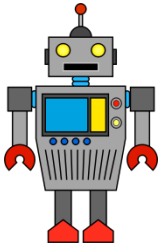
## Fall out:

- Linear increase in the team size as deployments get more complex
- Emergence of System Gurus
- Emergence of “Wall of confusion”
- Emergence of “Blame game”
- Extreme stress on Ops team
- No repeatable success

# Different deployment models

- Manual

- A little bit of automation here and there
  - Script
  - SSH, SCP
  - Database deployment scripts



This is where an opportunity to build a tool emerged

# Many attempts

**Step 1: Depicting your infra as code**

- Ability to use a common language either DSL or a Programming Language
- Ability to mark out all the possible activities and provide them as functions
- Eg. I want to restart a service
- Ability to bundle them together
- Ability to store this as code => repeatability
- Ability to configure your infra setup via
- Some config file which basically can be grouped IP addresses


Infrastructure becomes code!

# Many attempts

**Step 2: Way to deploy**

**PUSH Model**

- From a deployers' machine code generated in step 1 gets pushed to all the recipients
- No other setup required other than machines being visible from the deployers' machine

Eg. Ansible

**PULL Model**

- Deployer simply mentions what change has to be done and where.
- Nodes / Machines themselves pull the configuration
- Generally requires an Agent running on the nodes

Eg. Our hero of today's discussion CHEF !

# Chapter: Way to learn chef

# Learn Chef

**Difficulties**

- Introduces a lot of jargons! – workstation, knife, cookbook, resource, shelf !
- A greater paradigm shift which makes the learning curve steeper
- Too many moving parts and it uses a pull based (Agent based) model
- Too many tools to use
- Difficult to get a full system view
- Steeper learning curve

# Learn Chef

**The best way to learn chef**

- First get a full breadth view of what chef is trying to do
- See it working as a full set up
- Once the breath first view is satisfactorily done
- Then do a deep dive of each of the topics!

# Chapter: Chef - Introduction


CHEF
Achieve speed, scale, and consistency by automating your infrastructure with Chef

# Chef steps in!

- Chef is a powerful automation platform
  that enables Infrastructure automation

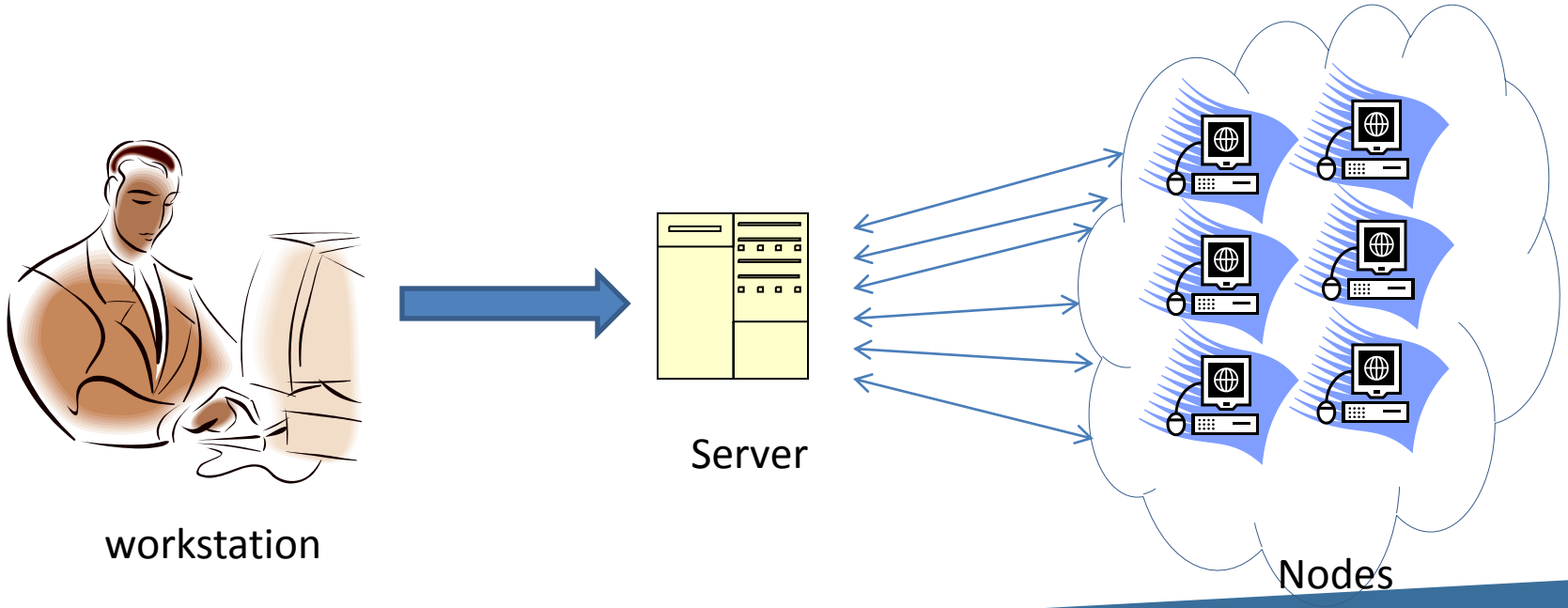
- Keeping in mind varied activity set of Infrastructure team Chef provides a huge set of resources to automate each kind of activity.

- Chef automates how infrastructure is configured, deployed,
  and managed across your network, no matter its size

- Works on cloud, on premises or hybrid environments

# Chefs- Metaphors

- How does chef achieve this?
  - It makes use of the below paradigm



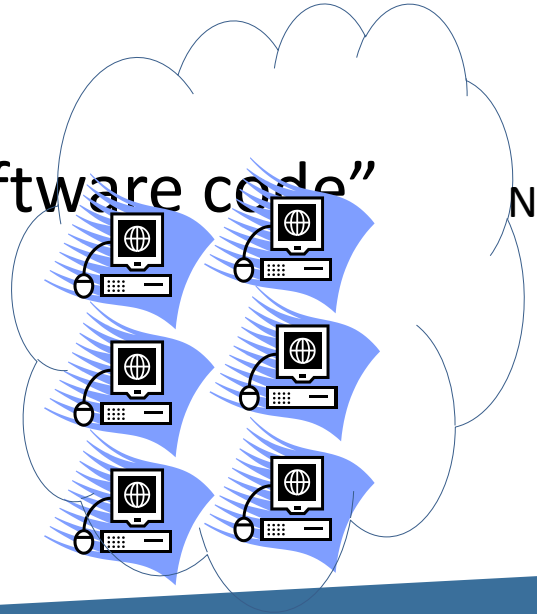
# Workstation

- One or more systems on which “engineers” will author the infrastructure configurations, policy setting, deployments etc

  - Write “**recipes**”
  - Bunch recipes together as “**cookbooks**”
  - Upload to chef server using “**knife**”
  - Needs “chef-dk” to be installed (Chef Development Kit)

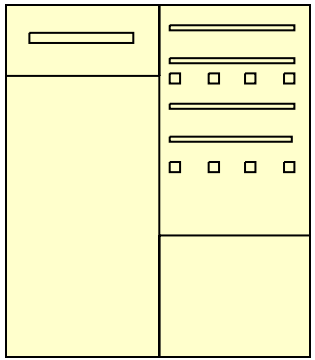
# Nodes

- Any machine – physical, virtual, cloud, device etc on which we need to automate infrastructure tasks via chef
- Nodes should run chef-client
- These are where your “deployed software code” runs too!
  - Prod machines, data servers,



# Server

- Hub of information
- Holds cookbooks uploaded from the workstation
- Responds to nodes and provides “cookbooks” and “policy changes”
- uploaded from the workstation
- Also hosts run data report from the nodes
- Pull Based



Server

# Chef at a bare minimum

- A piece of infrastructure instruction can be represented by a "**resource**" For eg. A file, Database, A service etc
- A resource can have "**attributes**" and "**actions**" For eg. git repository sync, copy a file, start a service
- A resource with its attributes and actions form a "**recipe**"
- Many recipes together form a "**cookbook**"

```
git "/home/vagrant/devopsfoundation/codeforchef" do
        repository "git://github.com/AdityaSP/devopsdemo.git"
        reference "master"
        action :sync
end
```

# Chef at a bare minimum

- Bootstrap a client to the server – one time activity
- Cookbooks are authored on a “**workstation**” using “**knife**”
- After authoring, Cookbooks are uploaded to “**Chef Server**” using “**knife**”

$knife cookbook upload **chef_trials**

knife bootstrap localhost --ssh-port 2200 --ssh-user vagrant --sudo --identity-file private key of the node --node-name **node1-ubuntu** --run-list 'recipe[chef_trials]’

# Chef client runs

- **“Chef clients”** running on each node contacts the server
- Downloads cookbooks to be run
- Interpret the recipes and apply them.
- Upload run data from the nodes

```
knife ssh 10.1.1.34 'sudo chef-client' --manual-list --ssh-
user vagrant --identity-file /vagrant/vagrant_private_key
```

# Chapter: Understanding our setup



# Chapter: Vagrant

# Vagrant

- By Hashicorp
- Open source
- Built on Virtualization
- Development Environments made easy
- Helps in creating light weight reproducible and portable development environments

# Vagrant

- A lot of vagrant boxes available on the net
- https://atlas.hashicorp.com/boxes/search
- vagrant init <box name> will download the box from the internet
- Can choose from varied boxes on the internet

```
➢vagrant init hashicorp/precise32
```

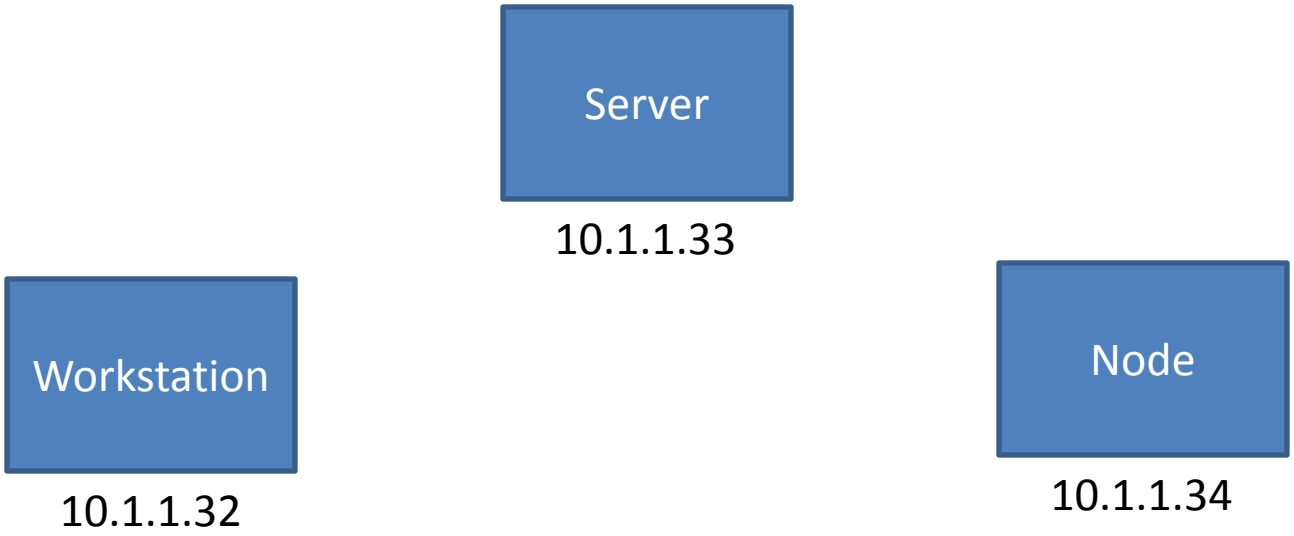
# Vagrant

- vagrant up – is all that is required
- Looks into a file called VagrantFile
- Creates network bridges
- Creates a file mount
- Assigns runtime memory
- Hooks with Virtual box

```
➢vagrant up
```

# Our set up

- Comes with 3 VMs - Workstation, Server and Node
- Imagine these 3 VMS as machines on the cloud
-



# Switch over to Quick run doc

# Chapter: Chef DK

# Chef Development Kit

- Includes all the tools needed to start writing chef code
- Chef DK tools are written in Ruby
- Chef DK comes bundled with a ruby scripting engine

```
#On your workstation
> cd /opt/chefdk
> cd bin
> ls
> cd ../embedded
> ls
```

# Chef DK tools

- Berkshelf – Dependency Manager
- chef-apply - An executable program that runs a single recipe from command line
- chef-client - Agent
- knife - A major component while working with chef
- ohai – installed as a part of chef-client – deals with attributes

# Chef DK and chef-client

- Chef DK is a superset of chef-client
- chef-client is installed on every system which is intended to be managed by chef

```
Check chef-client location on node and workstation
```

# Finding out versions of tools

```
# on your workstation
> chef --version
> chef-client --version
```

# Chapter: Just enough Ruby

# Ruby

- chef dk and chef client are implemented in Ruby
- Can use full ruby power in chef
- Ruby is a relatively new language – founded in 1993
- To replace Perl hence the name Ruby.
- Ruby , like Python is multi paradigm – supports functional, imperative, OOP
- Dynamically typed => you do not need to give datatype of a variable

# Starting ruby shell

- irb : Interactive ruby shell is a REPL for ruby
- The name irb is because ruby files are named with .rb extension

```
> a = 1
> b = 1.2
> c = “Hello”
> d = ‘Hello’
> puts a,b,c,d
```

# Strings

- String interpolation
- Difference between using “” and ‘’
- #{a} gets replaced with the value of a – this is called as interpolation

```
> a = 1
> s = “#{a} is the value”
```

# Arrays

- A very useful data structure

```
> states = ['KA','TN','TL',UP']
> states.length
> states.size
> states.push 'HR'
> states << 'MH'
> states.last
> states.first
> states[2..5]
```

# Dictionaries

- yet another useful data structure
- :ka and :kl are called as symbols

```
> states = { ka: "Karnataka", kl: "Kerala"}
> states[:ka]
> states[:kl]
> states = { :ka => "Karnataka", :kl => "Kerala"}
```

# Conditionals

```
> if a %2 == 0
  puts “Even Number”
end
```

# for loop

```
states = ['KA','TN','TL','UP']
for i in 0..states.length
 puts states[i]
end

states.each do |state|
 puts state
end
```

# Chapter: Chef's DSL

# Chef DSL

- Chef’s Domail specific Language
- Built using Ruby syntaxes
- The whole power of ruby is still available – Will see how in a short while

# Chapter: Recipe

# Chef's Hello World !

- Store the file as helloworld.rb in any folder that you want.

```
file "/tmp/helloworld.txt" do
  content "Hello Mysore!"
end
```

# Chef's Hello World !

chef-apply is an executable program that runs a single recipe from the command line:

- Is part of the Chef development kit
- A great way to explore resources
- Is **NOT** how Chef is run in production

```
> chef-apply helloworld.rb
```

# What does this mean

- file  => Name of the resoure
- content => a property which specifies the contents of a file
- But who told chef to create the file

```
file "/tmp/helloworld.txt" do
  content "Hello Mysore!"
end
```

# What does this mean ..contd

- By default its action :create

```
file "/tmp/helloworld.txt" do
  content "Hello Mysore!"
  action :create
end
```

# Let us try some variations

- **Variation:**
  - re run  chef-apply helloworld.rb
- **Variation:**
  - Change contents of the file at /tmp and re-run chef-apply !
- **Variation:**
  - delete the file hint “action :delete”
- **Variation:**
  - Create an array called **states** with values “ka” and “kl”
  - Use file resource to create files named ka.txt and kl.txt using the states array
  - hint action :touch

# Resource called “service”

- Restart a service
- Resource is “service”
- For apache the name is “apache2”
- Action is :restart

# Chapter: chef-client

# chef-client is more production like

- We used chef-apply to test a resource
- chef-client is the tool used to run in production scenario
- chef-apply is a great tool to explore resources
- Running of chef-client is also known as **“Chef Run”**
- chef-client can be run in three modes
  - local mode
  - client mode
  - solo

# chef-client - Local

**Local Mode**

- local mode simulates a full chef server in local memory
- What ever would be written to the server would be created in a folder called nodes – this is called as write-back

```
chef-client --local-mode helloworld.rb
# a new folder called nodes is created
cd nodes
```

# chef-client - client mode

**client mode**

- The default mode
- Tries to connect to chef-server and find out what to run
- Our beginning example dealt with this

# chef-client - solo mode

**Solo mode**

- Implemented through yet another tool chef-solo
- local-mode is far more convenient than chef-solo
- Will be eventually phased out. Chef encourages use of local-mode
- Difference is chef-solo does not do “writeback”

# Chapter: Cookbooks

# What are cookbooks

- Fundamental component of infrastructure management
- Contain recipes
- Generally each cookbook should only contain instructions for a single task of infra management

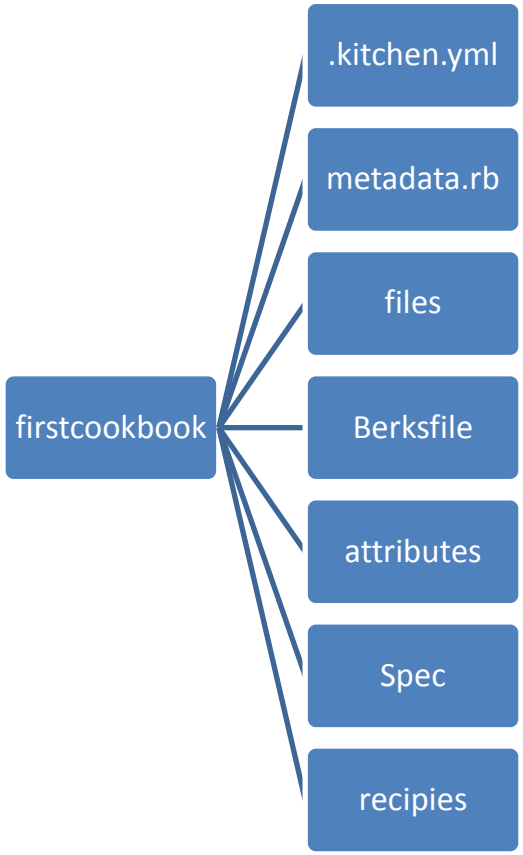
```
chef generate cookbook mycookbook
```

# Creating a cookbook with knife

- Knife cookbook create will be deprecated soon
- Note the differences between chef generate cookbook and knife cookbook create
- Do not forget the --cookbook-path !

```
knife cookbook create --cookbook-path .
```

# Cookbooks directory structure



# Run a cookbook

- Add a recipe in default.rb
- We will use chef-client –local-mode
- Has a parameter called runlist

```
chef-client --local-mode --runlist 'recipe[firstcookbook]'
```

# Understanding a client-run

1. chef-client starts a process on target node
2. constructs a node object in memory with all the attributes as given by “ohai”
3. Accesses the cookbooks provided with the runlist. This is also called as synchronizing the run list
4. Loads all the dependencies
5. Converge – Most important part of the run. This is where the recipes are executed on the node
6. Report

# Variations

- Variation 1: Copy your previous recipe to the recipes folder
  - Observation:  Do you see the recipe running?

- Variation 2 : run by providing recipe name along with runlist as recipe[firstcookbook::helloworld.rb]

- Variation 3 : create another cookbook and provide that too with runlist as recipe[firstcookbook],recipe[secondcookbook]

# Yet another tool - knife tool

- Use knife from workstation to
  - manage cookbooks, nodes, roles, data-bags etc
- knife is the connect between the chef-server and nodes
- Interacts with the server using REST APIs
- General syntax is knife <sub command> <verbs> <options>

```
knife cookbook create mycookbook
```

# Generate other aspects of cookbook

- Files
- Use case
- cookbook_file resource refers to this
- Consider you have a set of logo images which may change often. Make it a part of files and create a cookbook to replace logo images
- There is not need to go through this. Folders can be created manually too!

```
chef generate file file1.txt
```

# Chapter: Attributes

# Automatic attributes

- Ohai
- Collection of node information
- These are also called as automatic attributes
- JSON format
- chef-client reads the json output and converts it into a nodes object

```
> ohai | more
```

# Accessing Automatic attributes

- Node object is available in our cookbooks using “node”
- Nested variables can  be used too
- . Notation will be removed soon – recommended way is to node[‘ipaddress’]

```
log "IP Address : #{node['ipaddress']}"
log "IP Address : #{node[:ipaddress]}"
log "IP Address : #{node.ipaddress}"
log "kernel name: #{node['kernel']['name']}"
log "kernel name: #{node[:kernel][:name]}"
log "kernel name: #{node.kernel.name}"
```

# Our own attributes file

- From within your cookbook
- Create a folder called attributes
- Create a file within called default.rb

```
cookbook1 -> attributes -> default.rb
```

# Creating an attribute

- precedence[‘<name of attribute>’]=‘value of attribute’

```
default['apache']['deploy']='/var/www'
```

# Accessing a custom attribute

- precedence[‘<name of attribute>’]=‘value of attribute’
- node[‘apache’][‘deploy’]
- In your recipe file add the below line

```
log "#{node['apache']['deploy']}"
```

# Let us redefine the variable

- Without changing your attribute file
- add the below lines in your recipe
- Note that while assigning we use variable as an object of node
- Note the usage of default also

```
log "#{node['apache']['deploy']}"
node.default['apache']['deploy']="/var/www-reset"
log "#{node['apache']['deploy']}"
```

# Attribute precedence

- Elaborate attribute precendence
- Combination of precedence and sources !

| Precedence |
| --- |
| default |
| force_default |
| normal |
| override |
| force_override |
| automatic |

| Source |
| --- |
| Node |
| Attribute files |
| Recipes |
| Environments |
| Roles |

# As defined by Chef docs ( of course !)

## Attribute Precedence

Attributes are always applied by the chef-client in the following order:

1. A `default` attribute located in a cookbook attribute file
2. A `default` attribute located in a recipe
3. A `default` attribute located in an environment
4. A `default` attribute located in a role
5. A `force_default` attribute located in a cookbook attribute file
6. A `force_default` attribute located in a recipe
7. A `normal` attribute located in a cookbook attribute file
8. A `normal` attribute located in a recipe
9. An `override` attribute located in a cookbook attribute file
10. An `override` attribute located in a recipe
11. An `override` attribute located in a role
12. An `override` attribute located in an environment
13. A `force_override` attribute located in a cookbook attribute file
14. A `force_override` attribute located in a recipe
15. An `automatic` attribute identified by Ohai at the start of the chef-client run

where the last attribute in the list is the one that is applied to the node.

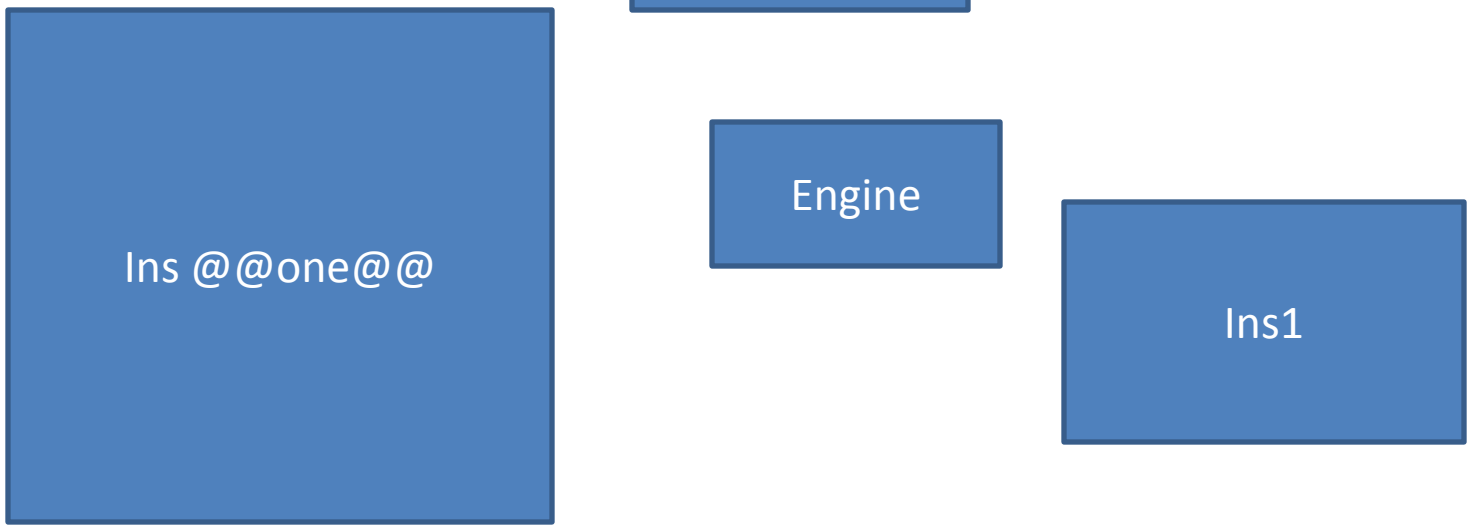
# Accessing a custom attribute

- precedence[‘<name of attribute>’]=‘value of attribute’
- node[‘apache’][‘deploy’]
- In your recipe file add the below line

```
log "#{node['apache']['deploy']}"
```

# Demo

- workinbox.pdf



# Chapter: template resource

# template resource – very important one!

- A variant of the cookbook_file resource that lets you create file content from variables using an Embedded Ruby (ERB) template.

- “templates” directory holds the Chef templates

- ERB – Templating engine which means it checks for syntaxes within a text file and executes them as ruby code

- Helpful while generating files with variable content

# template resource - create

Just like files you can create template in multiple ways

```
chef generate template ports.conf
mkdir -p templates/default
touch ports.conf.erb
```

# template resource - templetize

Put ruby code with in “<%= “ and “%>”

Create a file called ports.conf.erb under templates/default folder

Update ports.conf.erb to have this content

```
…
Listen <%= node['apache']['myport'] %>
…
```

# template resource - recipe

Looks very similar to file or cookbook_file resource

```
node.default['apache']['myport']=8084

template "/etc/apache2/ports.conf" do
  source 'ports.conf.erb'
end
```

# Chapter: knife and cookbooks

# Knife command patterns

- knife <subcommand> <verb> <options>
- Help available at each level

```
knife cookbook list
```

# Knife cookbook list

- List all the cookbook on the server

```
knife cookbook list
```

# Knife cookbook show

- Show cookbook details

```
knife cookbook show <cookbookname>
knife cookbook show <cookbookname> <version>
```

# Knife cookbook upload

- Upload cookbook

```
knife cookbook upload cookbook2
```

# Knife cookbook upload

- Variation1
- copy  your existing cookbook1 as cookbook2
- Upload it to server

# Knife cookbook download

- create another folder like chefdemo
- Note : .chef is necessary that’s all
- Do a cookbook list and choose what you want to download

```
knife cookbook download <name of the cookbook>
```

# Chapter: Roles

For people on the call : We are taking a dinner break now. The bridge will be open
We will come back and update you.

# Roles

- A nice way of bunching cookbooks together
- Can be administered on a node
- Easy to build roles like “webserver”, “developmentbox” etc
- All cookbooks related to one role can be bunched together

# Start creating a role – json file

- mkdir roles
- Create a file called workingboxrole.json

```
{
        "name": "workingboxrole",
        "json_class": "Chef::Role",
        "chef_type": "role",
        "run_list": [
                "recipe[apache-install]",
                "recipe[apache-conf]",
                "recipe[tomcat-install]",
                "recipe[tomcat-conf]"
        ]
}
```

# Start creating a role

- Create a role

```
> knife role from file workingboxrole.json
Updated Role workingboxrole!
```

# Some role commands

- `knife role list`
- `knife role delete <rolename>`
- `knife role show <rolename>`

# Chapter: Managing Nodes

# Bootstrapping a node

- bootstrap is a command to make a node belong to server
- bootstrap subcommand installs chef-client on the intended machine
- First a FQDN or visible ip address is needed
- Bootstrap does the following

1. Ssh into node,
2. install chef-client
3. generate-keys
4. register node with chef-server

```
➢knife bootstrap 10.1.1.34 --ssh-user vagrant --sudo --identity-file
/vagrant/vagrant_private_key --node-name node1-ubuntu
```

# Managing Nodes

- Show nodes already present with the server

```
knife node list
knife node show node1-ubunutu
knife node show node1-ubuntu -l
```

# Use run_list to manage runlists

Add run items

```
➢knife node run_list add node1-ubuntu 'recipe[somerecipename]'
➢knife node run_list add node1-ubuntu 'role[workingboxrole]'
➢knife node run_list add node1-ubuntu
'recipe[COOKBOOK::RECIPE_NAME],recipe[COOKBOOK::RECIPE_NAME],role[ROLE
_NAME]'
```

# Use run_list to manage runlists

Remove run items

```
➢knife node run_list remove node1-ubuntu 'recipe[somerecipename]'
➢knife node run_list remove node1-ubuntu 'role[workingboxrole]'
```

# Use run_list to manage runlists

Reset items

```
➢knife node run_list set node1-ubuntu 'recipe[somerecipename]'
➢knife node run_list set node1-ubuntu 'role[workingboxrole]'
```

# Use run_list to manage runlists

Reset items

```
➢knife node run_list set node1-ubuntu 'recipe[somerecipename]'
➢knife node run_list set node1-ubuntu 'role[workingboxrole]'
```

# Knife client runs

Use knife to trigger client runs

```
➢knife ssh 10.1.1.34 'sudo chef-client' --manual-list --ssh-user
vagrant --identity-file /vagrant/vagrant_private_key
```

# Delete nodes

- knife node delete <node name>

```
➢knife node delete node1-ubuntu
```

# Chapter: Data bags

# Data bag

- Databag are json data stores available to all nodes
- Generally all common values are kept under databag
- Also referred to as shared global data
- For eg. Users

```
{
  "username" : "aditya"
}
```

# Preparing to create a databag

- A similar pattern as roles
- Make a folder called “data_bags”
- Create a folder called “users” within “data_bags”
- Create a folder called <username>.json

```
{
  “id”: “aditya”,
  “home”: “/home/aditya”,
  “shell”: “/bin/bash”

}
```

# Add the databag

- First create a databag item
- Then create data items using aditya.json

```
knife data_bag create logins
knife data_bag from file logins aditya.json
```

# List and delete

- Very similar to cookbook commands

- To know further about data_bags we need to understand search. So lets digress a bit and know about Search

```
knife data_bag list
knife data_bag delete logins
knife download data_bags
```

# Demo of creating user using databags

- Demo

# Chapter: Search

# Chef Search

- Ability to query data indexed on the Server
- Search queries can be done from
  - 1. Recipes
  - 2. Using knife commands

# Chef Search using knife

- knife  search <index> <query syntax>
- Index can be
  - Node
  - Client
  - Environment
  - Role
  - Name of data_bag

# Knife search - examples

```
knife search node "*:*"
knife search node "ip*:10.0.*"
knife search node "fq*:node1*"
knife search node "fq*:node1-ubuntu or fq*:node2-ubuntu"
knife search node "*:*" -a recipes
```

# Search within a recipe

- search(index, search_query)

```
search("node", "*:*").each do |matching_node|
log matching_node.to_s
end
```

# Coming back to data_bag search

```
knife search logins “*:*”
knife search logins “id:aditya”
```

# Demo

Creating users based on login
Hint: openssl password -1 “welcome”

# Chapter: Encrypted Databag

# Encrypted data bags

Step1 : create key

```
$ openssl rand -base64 512 | tr -d '\r\n' > encrypted_data_bag_secret
```

Step2 : create a databag apikey

```
$ knife data_bag create apikeys
```

Step3 : encrypt

```
$ knife data_bag from file apikeys gitkey.json --secret-file
encrypted_data_bag_secret data_bag[apikeys:git]
```

Step4: show both encrypted and decrypted

```
$ knife data_bag show apikeys git
$ knife data_bag show apikeys git --secret-file
encrypted_data_bag_secret
```

# Use encrypted data in recipe

```
Step1 : scp the key to /etc/chef on the nodes

Step2 : retrieve key
secret =
Chef::EncryptedDataBagItem.load_secret("/etc/chef/encrypted_data_bag_s
ecret")

Step3 : create a databag apikey
git_key = Chef::EncryptedDataBagItem.load(“apikeys", “git", secret)

Step4 : git_key variable has decrypted values
log git_key[‘api-key’]
```

# Chapter: Environments

# Environments

- One more level of configuration flexibility
- Helps to keep things different between Prod, Dev, Test environments
- Helps in injecting attributes
- Helps in specifying cookbook versions to run
  - These cookbook versions can vary from prod to dev because on prod you would typically have a cookbook which would not change often
  - In Dev you would be using a cookbook version on which you are performing some testing

# Create an env file

- Follows a similar approach to role

```
Step1 : create a directory

$ mkdir environments

Step2 : create your env specific configuration

$ vi environments/dev.json
```

# Contents of env file

- Note the usage of override_attributes and cookbook_versions

```
{
  "name": "dev",
  "description": "For Dev Env",
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "cookbook_versions": {
       "apache2-conf": "= 0.1.0"
  },
  "override_attributes": {
       "apache": {
              "myport": "8087"
       }
  }
}
```

# Upload env file

- Follows a similar approach to role

```
Step3 : upload env file

$ knife environment from file dev.json
Updated Environment dev

Step4 : see the contents of dev

$ knife environment show dev
```

# To use it in a client run

- Create a recipe which prints out a log message. For now this is enough
- -E option along with chef-client

Step5 : run with chef-client

```
$ knife ssh 10.1.1.34 'sudo chef-client -E dev' --manual-list --ssh-user vagrant --identity-file /vagrant/vagrant_private_key
```

# Chapter: Including recipies

# include_recipe

- chef does an in place replace
- Attribute override starts to make sense here.

```
include_recipe 'cookbook1::att'
```

# include_recipe from a different file

- This is where dependency creeps in
- Dependencies are maintained in metadata.rb

```
depends 'includes2'
```

# Chapter: Extra Tools

# Extra Tools

- What we have studied so far is more than enough to manage chef
- Now because we have configured our infrastructure as code we need to treat our code better => according to some standards, perform tests etc
- Most of the tools are geared towards making our work with chef easy For eg : berkshelf
- A few of them to help us test : Kitchen, ChefSpec
- A few of them for code quality checks foodcritic

# Chapter: foodcritic

# What is foodcritic

- Lint tool
- Checks common problems in your cookbooks
- Has inbuilt rules around 60 rules
- Easy to use

```
$ foodcritic includes
```

# Tags

- Rules of foodcritic are tagged as different values helping us to choose what we want.
- Only a few rules may be useful to you and tag helps you to choose them

```
Tags Available for foodcritic
attributes
chef11
chef12
correctness
definitions
deprecated
environments
files
Libraries
templates
```

```
metadata
notifications
portability
process
readme
recipe
roles
search
services
strings
style
supermarket
```

# Choosing what rules to run

- rule number is also a tag
- -t helps to choose a tag

```
# one particular tag
$ foodcritic -t style
# or selection
$ foodcritic -t style, correctness
# and selection
$ foodcritic -t style -t correctness
# Exclusion
$ foodcritic -t ~FC023
```

# Creating a .foodcritic file

- The command line options that we used before can be make as a part of a file called .foodcritic at the root of cookbook

```
style,correctness
~FC023
```

# Chapter: Rubocop

# Rubocop

- It’s a ruby static code analyzer
- Enforce style conventions and best practices
- Nothing specific to chef per se. Foodcritic is more specific to Chef
- Rubocop rules are referred to as cops

# Rubocop

- All available checks are present in
- /opt/chefdk/embedded/lib/ruby/gems/2.3.0/gems/rubocop-0.39.0/config
- Navigate to your cookbook and execute
- https://github.com/bbatsov/rubocop/blob/master/config/disabled.yml

```
$ rubocop .
```

# Rubocop

- Understanding its output
- The below output implies
  - File 1 : C => Issue with Convention
  - File 5 : C => Issue with Convention
  - File 6 : W => Warning
  - File 2,3,4 : . => nothing to report
- Others are  E for error and F for fatal error

```
inspecting 6 files
C...CW
```

# Chapter: ChefSpec

# ChefSpec

- Unit testing framework for chef
- Built on RSpec + Chef's DSL
- Chef uses the same constructs of Rspec
- Each resource will have a specific way of testing. The best way to learn whole of ChefSpec is by running it on multiple resources and looking at the examples

```
$ Chef exec rspec
```

# Let us see a RSpec example

- Create a cookbook called “fortest”
- Remove all files within spec directory – we will revisit this later
- Create a file called default_spec.rb

```
describe "check addition" do
    it 'equals 2' do
      a = 2
      b = 1
      res = 2 * 1
      expect(res).to eq(2)
      expect(res).not_to eq(3)
    end
end
```

# Let us see a RSpec example

- Testing Scenario
- Name of testing scenario
- What are you testing
- Set of expectations
- Object being tested

```
describe "check multiply" do
    it 'equals 2' do
      a = 2
      b = 1
      res = 2 * 1
      expect(res).to eq(2)
      expect(res).not_to eq(3)
    end
end
```

# Let us see a RSpec example

- Testing Scenario
- Name of testing scenario
- What are you testing
- Set of expectations
- Object being tested

```
describe "check multiplication" do
  let(:res) { 1 * 2 }
  it 'equals 2' do
    expect(res).to eq(2)
    expect(res).not_to eq(3)
  end
end
```

“let” statement helps create a symbol and we can use it elsewhere.
The code can be written in the block with { and } or even with a do – end block

# Create a recipe with a file resource

- default.rb

```
file '/tmp/trial.txt' do
  action :create
end
```

# Corresponding ChefSpec unit test case

- default_spec.rb

```
file '/tmp/trial.txt' do
  action :create
end
```

# Let us see a RSpec example

ChefSpec DSL are marked

```
require 'chefspec'

describe "fortest::default" do

  let(:chef_run) {

        ChefSpec::SoloRunner.new(platform: 'ubuntu', version:
'12.04').converge(described_recipe)

        }
  it 'creates a file' do
    expect(chef_run).to create_file('/tmp/trial.txt')
  end

end
```

# Let us see a RSpec example

- Testing Scenario
- Name of testing scenario
- What are you testing
- Set of expectations
- Object being tested

```
describe "check multiplication" do
  let(:res) { 1 * 2 }
  it 'equals 2' do
    expect(res).to eq(2)
    expect(res).not_to eq(3)
  end
end
```

“let” statement helps create a symbol and we can use it elsewhere.
The code can be written in the block with { and } or even with a do – end block

# Similar checks for template and package

ChefSpec DSL are marked

```
# for template with action :create
expect(chef_run).to create_template('/tmp/explicit_action')

expect(chef_run).to create_template('/tmp/with_attributes').with(
        user: 'user',
        group: 'group',
        backup: false, )

# for package with action :remove
expect(chef_run).to remove_package(‘apache2')
```

# A ton of examples available at

```
https://github.com/sethvargo/chefspec/tree/master/examples
```

# Chapter: Berkshelf

# BerkShelf

- A dependency manager
- Any cookbook created will have a Berksfile created.
- Maven:Java::Berkshelf:Chef
- It’s a Command line Tool – Comes automatically installed with chefDK
- Berksfile is an important part of Berkshelf
- Pom.xml:Maven::Berkshelf:Berksfile
- Maven:mv::Berkshelf:berks

```
$ berks --version
```

# BerkShelf – A look at Berksfile

- Pasted below are the contents of Berksfile
- Why is a dependency manager important?
  - First let us take a look at Supermarket

```
source 'https://supermarket.chef.io"
metadata
```

# Community cookbooks:Chef Supermarket

A place to explore and view chef cookbooks
Tons of pre made cookbooks available.
It is possible for us to reuse a cookbook. Thus reducing coding and testing times.
Stats : 3,245 Cookbooks 71,658 Chefs

```
# the website - default
https://supermarket.chef.io

# Actual Git
https://github.com/chef-cookbooks

# Restful APIs
https://supermarket.chef.io/api/v1
```

# Chef Supermarket: End points

Can access through knife, Berks
Even through APIs directly if you are writing a lot of internal code

```
#Berksfile
cookbook 'java', '~> 1.48.0'

#Knife
knife cookbook site download java
```

# Berkshelf and community cookbook

- berks init will install the missing pieces that Berks require from your cookbook
- berks install will download all the dependencies for you

```
$chef generate cookbook berkstrial
$cd berkstrial
$berks init
$berks install
```

# Berksfile anatomy

- Sources will help tell berks from where to download  a dependency
- Order is important
- As soon as Berkshelf finds a suitable resolution, the search ends there
- Metadata keyword will ask berkshelf to take care of metadata.rb dependencies

```
source source1URL
Source source2URL
```

# Berksfile anatomy – cookbook keyword

- cookbook keyword tells what cookbook to include as a dependency

```
cookbook “<cookbook name>” [, “<version and constraint>"] [, “source”]
cookbook git
cookbook git , “~> 0.1.0”
cookbook git , “~> 0.1.0”, path: “..\git”
cookbook git , “~> 0.1.0”, git: “https://github.com/AdityaSP/git.git”
```

# Berksfile anatomy – cookbook keyword

- cookbook dependencies

```
Pessimistic                    ~>
Equal to                       =
Greater than or equal to       >=
Greater than                   >
Less than                      <
Less than or equal to          <=
```

This symbol is called a
“twiddle-Wikka”
or
“twiddle-rocket”

# Berksfile command line interface

- A nice suite of command line commands to help manage dependency

```
$ berks init
$ berks install
$ berks info mycookbook
$ berks list
$ berks outdated
$ berks verify
$ berks viz
```

# Chapter: Kitchen

# Kitchen

- Execute configured code in isolation
- Uses Vagrant
- Uses inspec for compliance and verification

# Kitchen

```
demo
```

# Chapter: Inspec

# Kitchen

```
demo
```

# Chapter: Chef Server

# Server

Central element or brain of Chef framework
Detailed information of registered components
Hub of information

## Main Functionality

Stores various policies that are applicable for different cookbooks
Stores metadata that has the information of each node that is registered with Chef server
Provides facility to search any type of data with the help of search indexes using indexing

# Chef

Written in Ruby

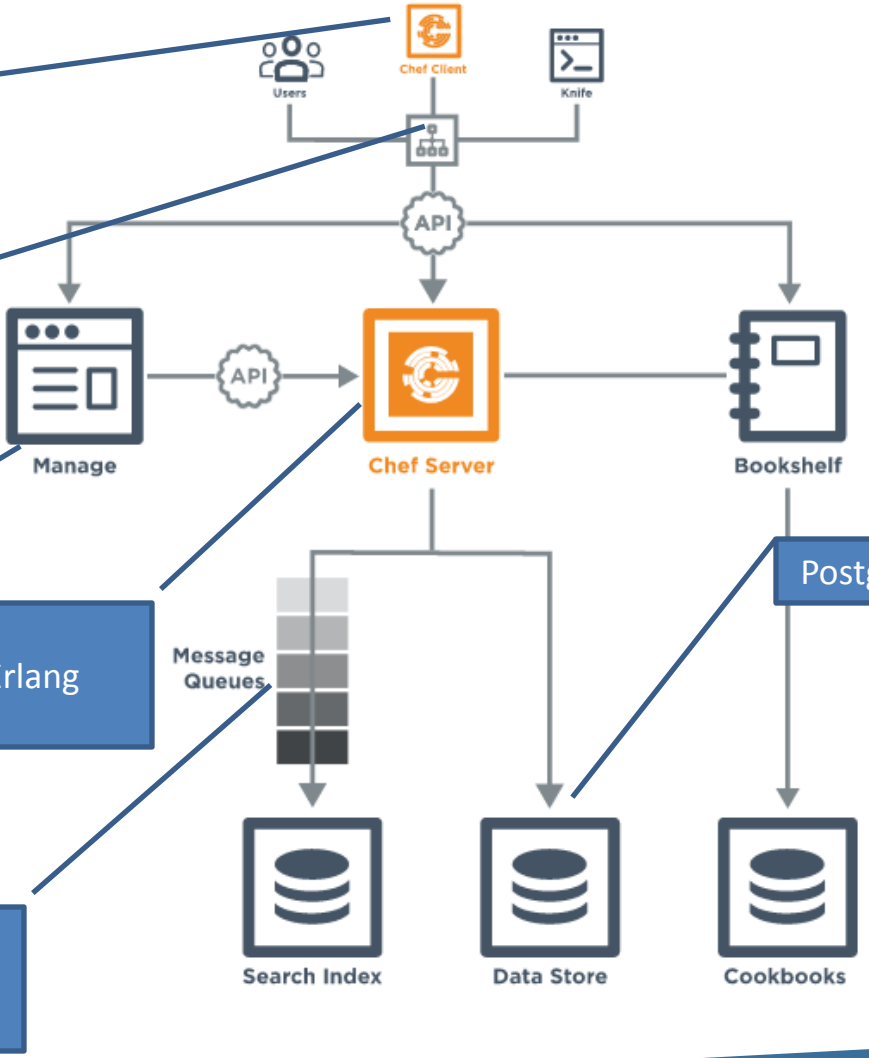
Load balancer- Nginx

Web UI built on RoR

Erlang

RabbitMQ

PostgreSql



# Server – Different types of server

1 . Hosted Chef Server

- Hosted by Opscode
- No management overhead – Opscode will take care of this
- Full scalability , full availability and resource based access control
- Freemium model – Free until 5 nodes. Paid - Standard and Premium Models
- Economically viable due to shared infrastructure costs

| PRO | CON |
|---|---|
| Maintenance free | Expensive |
| Easy for newbies | Security concerns due to shared infa |
| Fully supported by opscode/ Chef Inc | |

# Server – Different types of server

1 . Private Chef

- Hosted on private infrastructure
- To alleviate privacy and security concerns

| PRO | CON |
|---|---|
| Higher customization | Expensive licenses and support costs |
| Better security | Requires expertise |
| Faster deployments due to inhouse setup | Not suitable for beginners |

# Server – Different types of server

1 . Open Source Chef

- Free
- This is the setup we used in our demos

| PRO | CON |
|---|---|
| Free – as in free lunch! | Open source commitments |
| Open source and can create custom implementations through source code changes | No upgrades and support |
| | Requires expertise |

# Server – setup – lot of steps

- Requires FQDN
- apt-get -y install curl
- apt-get -y install ntp
- Download chef-server and install using deb (on ubuntu)
- sudo chef-server-ctl reconfigure
- sudo chef-server-ctl restart

- sudo chef-server-ctl user-create admin Bob Admin admin@4thcoffee.com insecurepassword --filename admin.pem

- sudo chef-server-ctl org-create 4thcoffee "Fourth Coffee, Inc." --association_user admin --filename 4thcoffee-validator.pem



# Jenkins

# Jenkins Setup

- Three ways to setup jenkins

```
1. Run Jenkins using inbuilt server (WinStone)
2. Install Jenkins in a J2EE servlet container like Tomcat etc
3. Windows service
```

# Jenkins Setup

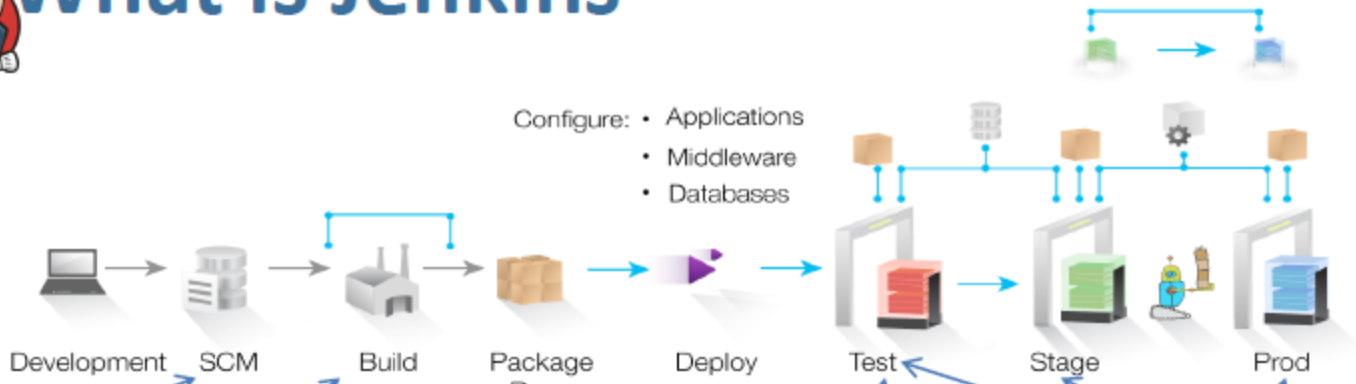
- Run Jenkins using inbuilt server

```
java -jar jenkins.war --httpPort=8090
```

# What is Jenkins

- A continuous build tool
- Automate build, artefact management, deployment process
- Simple, UI based easy to use tool
- Inbuilt scheduler
- Very essential to the “continuous feedback loop”

# What is Jenkins




Git, SVN

Ant, Gradle, Make, Maven

jUnit, Selenium, AntUnit DbUnit

INFRASTRUCTURE Chef, Puppet

Continuous Integration

Continuous Deployment and Delivery

Jenkins for CI and CD

# Jenkins – pipeline and freestyle

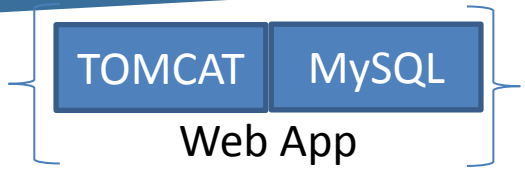
- Freestyle
  - Configure the whole Jenkins process as tool runs
  - Configure Jenkins as build step
- Pipeline
  - Configure using groovy script
  - Better visualization of build steps
  - More control in terms of branching etc

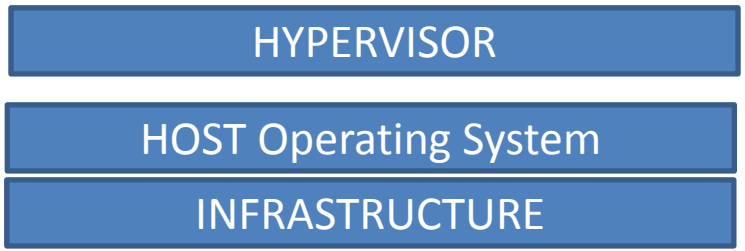
# Chef and Jenkins integration

demo

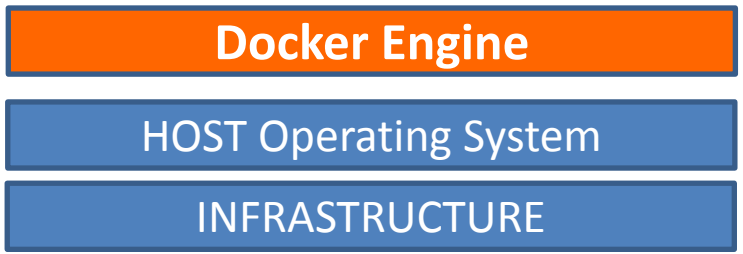
# Chapter: Docker Introduction

# Docker




VIRTUAL MACHINES




DOCKERS CONTAINERIZATION

# Docker – is it old wine in a new bottle?

- **Containers**
  1. Linux Containers have been around for a long time now
  2. Seen as difficult to setup and maintain
  3. Difficult technology for mass uptake
- **Enter Docker**
  1. Easy to use
  2. Docker ecosystem
  3. Skips the laborious process of setting up environments
  4. Isolation capabilities
  5. Eliminates environment inconsistencies

# Docker – A quick demo

# Docker – Chef on Docker demo

```
docker_image 'nginx' do
  tag 'latest'
  action :pull
end
# Run container exposing ports

docker_container 'my_nginx' do
   repo 'nginx'
   tag 'latest'
   port '80:80' binds [ '/some/local/files/:/etc/nginx/conf.d' ]
   host_name 'www'
   domain_name 'computers.biz'
   env 'FOO=bar'
   subscribes :redeploy,
   'docker_image[nginx]'
end
```

# Chapter: notify and conditionals

# notify

```
template '/etc/chef/server.rb' do
  source 'server.rb.erb'
  owner 'root'
  group 'root'
  mode '0755'
  notifies :restart, 'service[chef-solr]', :delayed
  notifies :restart, 'service[chef-solr-indexer]', :delayed
  notifies :restart, 'service[chef-server]', :delayed
end
```

# not if

```
file '/tmp/myfile.txt' do
  not_if { ::File.exists?('/tmp/myfile.txt') }
end

package 'apache2' do
  only_if { node['kernal']['name'] == 'Linux' }
end
```

# Chapter: Summary

# Chapter: Thank You