Linux Academy

Certified Chef Developer

This course is focused around the Chef Fluency Badge.

This course is part of an overall learning path to include all three courses for the Chef Certified Developer Certification.

This course, and learning Chef through Linux Academy, does not require any prior Ruby development experience. We will teach you everything you need to know!

Linux Academy

Additional Certification Tracks **W**here This Badge Will Work

Certified Chef Developer

Certified Chef Architect

Linux Academy

The Exam

- Given and proctored online.
- This exam is multiple choice.
- 60 minutes (one hour) allotted to take and complete the exam.
- **This course will cover ALL (100%) of the required information to pass the exam. No additional resources are required.**

Linux Academy

The Exam

- The exam is not issued or proctored by Linux Academy.
- To register for the exam go here: https://training.chef.io/basic-chef-fluency-badge
- The exam costs $75.

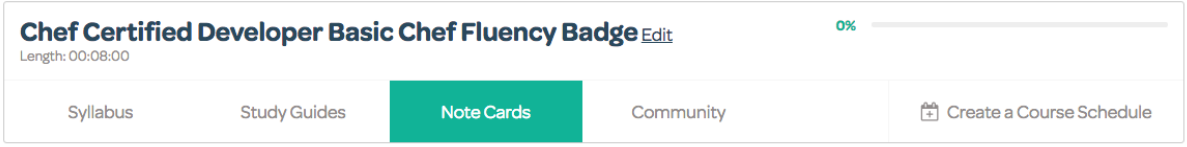
Linux Academy

The Course

We will be using with the terminal and Linux Academy lab servers to learn the content, not only conceptually, but hands-on! Please use the Linux Academy lab servers to follow along so we can assist with any issues that you might have.

Practice exam at the end of the course.

Use the instructor deck and other student decks in the Note Card section of the course to help study and prepare.



Linux Academy

The Exam

DO NOT take the Chef exam until you have passed the Linux Academy practice exam and have been issued your certificate of completion. At this point you will be ready to take the exam.

Do not skip videos or parts of videos. Certification material is in-depth and you'll want to ensure you cover everything required.

DO NOT "game" the practice exam.

- Try to answer the exam based off your knowledge; if you fail, rather than studying the answers, go back to the section of the course and relearn it. Only when you truly understand the concepts are you prepared for an exam.

Linux Academy

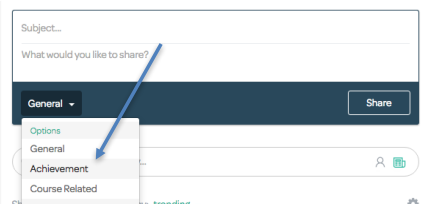
After Passing

Add your badge to the Linux Academy professional certification section on your personal Linux Academy profile.

Invite a friend to Linux Academy.

Add your certificate of completion to your LinkedIn profile.

Share with others in the Linux Academy community.



Linux Academy

Installing ChefDK

To get started with our training we will be using a single Linux Academy server with Chef installed.

To start we will be executing our code against that single node. As we progress we'll add Chef server and Chef Workstation into our workflow.

http://downloads.chef.io/chef-dk/

Note: ChefDK to be covered later in this course

Move to Terminal And Install Chef

Linux Academy

What is Chef?

Chef put simply, is a configuration management and deployment tool.

Chef at its core, is designed for configuration management of nodes within an environment.

Chef has evolved over the years to be a very robust and complete Continuous Integration, Continuous Deployment, and infrastructure compliance tool.

Chef is not just for on-premises environments, but also cloud-based environments like Amazon Web Services and Azure.

Linux Academy

Anatomy of a Chef "Convergence"

Pre-convergence - Phase before a node is configured

- Lint tests occur in this phase - Lint tests run tools to analyze source code to identify stylistic problems, foodcritic is a tool for this when using chef.

Convergence - Occurs when Chef runs on the node

- Tests the defined resources to ensure they are in the desired state
- If they are not, then the resources are put in the desired state (repaired)
- "Providers" are what do the work to enforce the desired configuration
- Chef can be run over and over again without changing configurations if configurations are already in place (Idempotency)

Post-convergence – Occurs after the Chef convergence

- Run tests that verifies a node is in the desired state of configuration also known as Unit Testing

Linux Academy

Resources

A resource is a statement of desired configuration for a given item.

A resource describes the desired state and steps for achieving the desired configuration.

Resources are managed within "recipes" (which will be covered in later videos) and are generally grouped together within cookbooks for management-specific software and tasks.

A resource statement defines the desired state of the configuration, a resource maps to a "provider" which defines the steps to achieve that desired state of configuration.

Linux Academy

Resources

Chef is idempotent, which means, during a Chef run (aka convergence) Chef determines if the desired configuration within a resource is set; if it is set, Chef does not run anything against the node. If the desired configuration does not match then Chef enforces the desired state configuration.

A resource has **four** components and is made up of a *Ruby* block of code

- Resource Type
- Resource Name
- Resource Properties
- Actions to be applied to the resource

Linux Academy

Package Resource Type

```
package 'httpd' do
      action :install
end
```

Ruby block of code

Note: In the absent of action, the default is :install

```
package 'httpd' do
end
```

What is happening here?

The httpd package is being installed ONLY if it is not already installed.

Why httpd over Nginx?

Linux Academy

Package Actions

:install – install a package (the default if no action is defined)

:nothing – This action does nothing UNTIL it is notified by another resource to perform an action

:purge – Removes the configuration files as well as the package (Only for Debian)

:reconfig – Reconfigures a package

:remove – Removes a package (also removes configuration files)

:upgrade – Install a package, if it is already installed, ensure it is at the latest version.

Linux Academy

Service Resource Type

```
service 'httpd' do
      action [:enable, :start]
End
```

The service httpd is enabled so it starts at boot time and then started so that it is currently running.

Linux Academy

Service Resource Type

```
service 'apache' do
      service_name 'httpd'
      action [: enable, :start]
End
```

What happened here?

Linux Academy

Service Actions

Service actions are available as a property for the service resource type.

:disable – Disable a service so it does not start at startup

:enable – Enable a service to start at boot time

:**nothing** – Does nothing to the service

:reload – Reloads the service configuration

:start – Starts the service and keeps it running until stopped or disabled

:restart – Restart a service

:stop – Stop a service

Linux Academy

Service resource type: Notifies property

What happens when a change is made that requires a restart to a service?

The notifies property allows a resource to "notify" another resource when the state changes.

Example:

```
file '/etc/httpd/vhost.conf' do
      content 'fake vhost file'
      notifies :restart, 'service[httpd]'
end
```

Linux Academy

NOT_IF & ONLY_IF Guards

NOT_IF - prevents a resource from executing when a condition returns **true**.

```
execute ‘not-if-example’ do
      command ‘/usr/bin/echo "10.0.2.1 webserver01" >> /etc/hosts’
      not_if ‘test -z $(grep "10.0.2.1 webserver01" /etc/hosts)’
end
```

ONLY_IF -Allow a resource to execute only if the condition returns **true**.

```
execute ‘only-if-example’ do
      command ‘/usr/bin/echo "10.0.2.1 webserver01" >> /etc/hosts’
      only_if ‘test -z $(grep "10.0.2.1 webserver01" /etc/hosts)’
end
```

Linux Academy

Installing Multiple Packages At One Time

```
%w{httpd vim tree emacs}.each do |pkg|
      package pkg do
            action :upgrade
      end
end
```

Linux Academy

Installing Multiple Packages At One Time

```
package ‘httpd’ do
      action :install
End

package ‘vim’ do
      action :install
End
```

package ‘emacs’ – No “do or end is required”

Linux Academy

Installing Multiple Packages At One Time

```
['mysql-server','mysql-common','mysql-client'].each do |pkg|
      package pkg do
        action :install
      end
ends
```

Linux Academy

Default Resource Actions

package ‘httpd’

In the absence of a defined “action” property the default action for the package resource type is “:install”.

Linux Academy

Default Resource Actions

service ‘httpd’

In the absence of a defined “action” property the default action for the service resource type is “:nothing”.

Linux Academy

Default Resource Actions

file ‘/etc/motd’

In the absence of a defined “action” property the default action for the file resource type is “:create”.

m3m0ryDump*

Linux Academy

Understanding Chef Extensibility: Custom Resources

Chef is designed to be extended and grow with the complexity and requirements of any environment.

Chef is also designed in a way that custom resources can be used to reduce repetitive coding and manual errors that are accompanied with such coding.

Note: Custom resource examples will be covered in later lessons

Linux Academy

Understanding Chef Extensibility: Custom Resources

Very simply, a custom resource:

- Is a simple extension of Chef
- Is implemented as part of a cookbook (in the /resources directory)
- Follows easy, repeatable syntax patterns
- Effectively leverages resources that are built into Chef
- Is reusable in the same way as resources that are built into Chef
- Can be used to take complex resource declarations and reduce the properties required for configuration that allows less experienced admins to be able to use those custom resources rather than more complex resource policies

Linux Academy

Understanding Chef Extensibility: Libraries

Libraries in Chef are used to create reusable sets of code that can be placed or used within a recipe in a clean way.

Example:

```
execute 'not-if-example' do
      command '/usr/bin/echo "10.0.2.1 webserver01" >> /etc/hosts'
      not_if 'test -z $(grep "10.0.2.1 webserver01" /etc/hosts)'
end
```

Linux Academy

Understanding Chef Extensibility: Libraries

Take the "test -z $(grep "10.0.2.1 webserver01" /etc/hosts)"' and add it to a library as a Ruby function

(We will avoid the library code at the moment until knowledge on writing libraries is required)

The execute resource type then looks liked this:

```
execute 'not-if-example' do
      command '/usr/bin/echo "10.0.2.1 webserver01" >> /etc/hosts'
      not_if { has_host_entry? }
end
```

Linux Academy

Recipes

We've been writing our resource blocks within a recipe.

Recipes are written using Ruby. There is no "custom" language DSL to use Chef. Not a bunch of Ruby knowledge is required, just the basics, of which we will learn in the course.

Recipes are a collection of resources, defined and written using patterns. Helper code, such as loops and if statements, can be written around those resources to help customize the configurations of specific nodes. For example, *if* or *case* statements around package names.

Linux Academy

Chef defines a recipe as the most fundamental configuration element within the organization.

A recipe:

- Is created using Ruby
- Is a collection of resources defined using patterns; helper code is added using Ruby
- Must define everything that is required to configure part of a system
- Must be stored in a cookbook
- May be included in a recipe (include_recipe)
- May use the results of a search query and read the contents of a data bag
- May have a dependency on one (or more) recipes
- May tag a node to facilitate the creation of arbitrary groupings
- Must be added to a run-list before it can be used by the chef-client
- Is always executed in the same order as listed in a run-list
- If included multiple times in a run-list, will only be executed once

Linux Academy

Chef Resource Ordering Execution

Resources are executed in the order that they are listed/created within a recipe, starting with the first recipe in a run-list.

There are "directives" that can change the order in which resources are executed

- notifies: A notification property that allows a resource to notify another resource to take action when its state changes
- subscribes: A notification property that allows a resource to listen to another resource and then take action if the state of the resource being listened to changes.

Linux Academy

Chef Resource Ordering Execution

```
service "httpd" do
end


cookbook_file "/etc/httpd/conf/httpd.conf" do
      owner 'root'
      group 'root'
      mode  '0644'
      source 'httpd.conf'
      notifies :restart, "service[httpd]"
end
```

Linux Academy

Chef Resource Ordering Execution

```
service "httpd" do
      subscribes :reload, "service[httpd]"
end


cookbook_file "/etc/httpd/conf/httpd.conf" do
      owner 'root'
      group 'root'
      mode  '0644'
      source 'httpd.conf'
end
```

Linux Academy

Chef Resource Ordering Execution

```
service "httpd" do
     action [:enable, start]
end


cookbook_file "/etc/httpd/conf/httpd.conf" do
      owner 'root'
      group 'root'
      mode  '0644'
      source 'httpd.conf'
      notifies :restart, "service[httpd]"
end
```

Linux Academy

A Brief Look at run_list

A run-list is a list of cookbooks/recipes that are to be executed on the given node.

Example:
run_list "recipe[base]","recipe[apache]","recipe[selinux_policy]"

chef-client will execute the base recipe followed by apache, selinux_policy.

run_list "recipe[base::recipe]","recipe[apache::recipe]", "recipe[selinux_policy::recipe]","recipe[base::recipe"

Note: If for any reason a recipe is assigned to a node more than once (via roles/environments/etc.) chef-client will only execute it **one** time.

Linux Academy

include_recipe

A recipe can include a recipe from an external cookbook

For example, including the mod_php recipe in an already existing recipe

include_recipe 'cookbook_name::recipe_name'

Note:
include_recipe 'cookbook_name' will, by default, translate to 'cookbook_name::default' the default recipe

Linux Academy

include_recipe

A recipe can include a recipe from an external cookbook

For example, including the mod_php recipe in an already-existing recipe

include_recipe 'cookbook_name::recipe_name'

Note:
include_recipe 'cookbook_name' will, by default, translate to 'cookbook_name::default' the default recipe

Important: If a recipe is being included from an external cookbook, then it's important to create a dependency on that cookbook in the metadata.rb file of the existing cookbook. We'll cover this in up coming lessons.

Linux Academy

Cookbooks

A cookbook is the fundamental unit of configuration and policy distribution when using Chef.

Cookbooks contain the following information:

- Recipes
- Attribute files
- File distributions
- Templates
- Any extensions to Chef such as libraries and custom resources

Linux Academy

Cookbooks

A Chef cookbook defines a scenario. For example, our Apache cookbook. The cookbook defines everything needed to install and configure Apache.

Modules and additional Apache configurations required for our application can be broken out into individual recipes within the cookbook.

Linux Academy

Cookbooks: README.md

The cookbook readme file, located inside of cookbooks/cookbookname/README.rb is a description of the cookbook's features that is written using Markdown.

Markdown is text-to-HTML conversion tool for making it easy to write structurally valid HTML.

Linux Academy

Cookbooks: metadata.rb

Cookbook metadata is located in cookbooks/cookbookname/metadata.rb. Each cookbook requires certain metadata information.

Common Metadata Settings for Chef Fluency Badge

- chef_version: Allows you to specify which version of Chef the cookbook requires to operate correctly
- depends: Allows you to specify if there are any other cookbook dependencies, including the same cookbook but a different version (think back to include_recipe)
- version: Specifies the version of the cookbook. Chef server stores the versions differently, allowing for version control of cookbooks within Chef server.

Linux Academy

Cookbooks: metadata.rb

```
name 'mycookbook'
maintainer 'The Authors'
maintainer_email 'you@example.com'
license 'all_rights'
description 'Installs/Configures mycookbook'
long_description 'Installs/Configures mycookbook'
version '0.1.0'
depends 'mysql', '>= 1.0'
```

Linux Academy

Cookbooks: Default Cookbook Recipe

The default cookbook recipe is default.rb, which is created with the cookbook. The main part of the configuration for the cookbook generally lives here. For example, installing packages and starting services for the cookbook scenario (i.e Apache/MariaDB).

What does the default cookbook mean?

If you were to include a cookbook without a recipe in a run list, then by default it would run cookbookname::default the default recipe.

Linux Academy

Cookbooks: Default Cookbook Recipe

run_list "recipe[apache]"

This is the same as: **run_list "recipe[apache::default]"**

Use the apache cookbook and add the default recipe to the node's run-list. All other recipes in the cookbook, unless include_recipe is used, are ignored.

Linux Academy

Chef Server

The Chef server is the central location which acts as an artifact repository or "hub" that stores cookbooks, cookbook versions, facts about the node, data bags and metadata information about nodes.

All metadata of a node, when it is registered with Chef server, is stored on the Chef server.

The metadata is populated and sent to the server with chef-client, which is an application that runs on the node. (Covered in later lessons.)

Configuration enforcement is not handled by the Chef server, instead, the desired state configuration is enforced when chef-client runs and a "convergence" happens, allowing for easy scalability.

Linux Academy

Chef Server Components

Clients

- Nodes and workstations which access the Chef server for either configuration enforcement or uploading and the management of cookbooks and other Chef data.

Load balancer

- All requests to the Chef server are routed through the nginx front-end load balancer.

Chef manage

- Chef manage is the GUI-/Rails-based web application for managing Chef server.

Chef Server

- The server engine which powers Chef server.

Linux Academy

Chef Server Components

Bookshelf
- Used to store cookbook content such as files, recipes, templates, etc.

Message Queues
- Are used (using RabbitMQ) to send data to the search index and data stores. (Search indexes covered in a later lesson.)

PostgreSQL
- The data storage repository for Chef server.

Linux Academy

Chef Solo and Chef Zero is a lightweight Chef Server that runs in-memory on the local node and are now rolled into chef-client and used as “local mode” chef-client –z or chef-client --local-mode

- Allows chef-client to run against the chef-repo as if it was running against Chef Server.
- Useful for testing and validating the behavior of chef-client against our cookbooks, recipes, and run-lists before uploading those to the Chef Server for usage
- Allows Chef to run on single nodes without the need of a central server for distributing artifacts.

- Allows chef-client to run against the chef-repo as if it was running against Chef Server.
- Useful for testing and validating the behavior of chef-client against our cookbooks, recipes, and run-lists before uploading those to the Chef Server for usage

Linux Academy

Chef Server

Chef Server is a central point of management for nodes within an environment.

- Authentication is required for the nodes to communicate with the Chef Server.
- During a convergence the chef-client pulls updated cookbook, recipe, roles, and environment information from Chef Server.
- Chef Server is used for managing many different nodes and different node scenario configurations within an environment.

Linux Academy

Node Object

The node object is made up of groups of attributes and the node run-lists.

What is an attribute?

- An attribute is a specific piece of data about the node.
  - cpu information
  - ip address
  - hostname
  - memory
  - swap
  - etc

Linux Academy

Node Object

Attributes are collected by a tool called ohai, which we'll explore in our hands on lesson coming up.

Chef-client automatically executes ohai and stores the data (attributes) about a node in an object.

This node object information can be used within the recipes named node.

This information combined with the nodes run-lists is called the **node object**.

The node object is stored in a JSON (JavaScript Object Notation) file on the server.

Linux Academy

Search

Chef search allows a search from either knife or within a recipe in order to search any data that is indexed by the Chef server.

Data is stored within Chef server indexes (5 of them):

- Client
- Data bags
- Environments
- Nodes
- Roles

Linux Academy

Knife Query Syntax

Knife search INDEX “key:search_pattern”

Note: If no index is passed, then the default “node” is applied.

Key is a field name found in the JSON description of an indexable object on the Chef server and search_pattern defines what will be searched for.

Index can either be a role, node, client, environment, or data bag.

The search pattern can include certain regular expressions to form a search query. This is supported in knife as well as when using search within a recipe.

Linux Academy

Data bags:

A data bag is a global variable that is stored as JSON data and is accessible from a Chef server. A data bag is indexed for searching and can be loaded by a recipe or accessed during a search.

Example use cases:
- Storing API and APP id information
- Storing users to be added to a system

Linux Academy

Using Search For Dynamic Orchestration

Scenario: Discover all nodes with a role of “web” and add them to a load balancer.

Web_nodes = search(‘role’, ‘role:web’)

Role = The index we are going to search
Role:web = The key:search_pattern

Example: Hands-on later in the course.

Linux Academy

Search

Knife search node ‘platform_family:rhel’
Knife search node ‘recipes:apache\:\:default’
Knife search node ‘platform:centos or platform:debian’

Regular Expressions in search:
Knife search node ‘platform*:ubuntu’
Knife search node ‘platfor?:centos’
Knife search ‘network_interfaces__addresses:*’

* - Replaces zero or more characters with a wildcard
? – Replaces a single character with a wildcard

Linux Academy

Search Flags:

-i will show the node ID
-a attribute_name will display the specified attribute from the search query results
-r will show the run-lists for the query results

Knife search ‘*:*’ –r

Will yield the same results as

Knife search ‘*:*’ –a run_list

Linux Academy

Roles

A role describes a run_list of recipes that are executed on a node.

How might you specify which recipes are to be run on different sets of nodes, without manually modifying each nodes run_list each time a run_list change is required?

Linux Academy

Roles

```
{
 "name": ”web",
 "description": ”Role for our web sever nodes for wordpress application",
 "json_class": "Chef::Role",
 "default_attributes": {

 },
 "override_attributes": {

 },
 "chef_type": "role",
 "run_list": [
        "recipe[apache2]",
        "recipe[apache2::websites]",
        "role[monitoring]"
 ],
 "env_run_lists": {

 }
}
```

Linux Academy

Roles

```
{
 "name": ”database",
 "description": ”Database Servers For Our Wordpress Application ",
 "json_class": "Chef::Role",
 "default_attributes": {

 },
 "override_attributes": {

 },
 "chef_type": "role",
 "run_list": [
        "recipe[postgreSQL]",
        "recipe[postgreSQL::create_databases]",
        ”role[monitoring]"
 ],
 "env_run_lists": {

 }
}
```

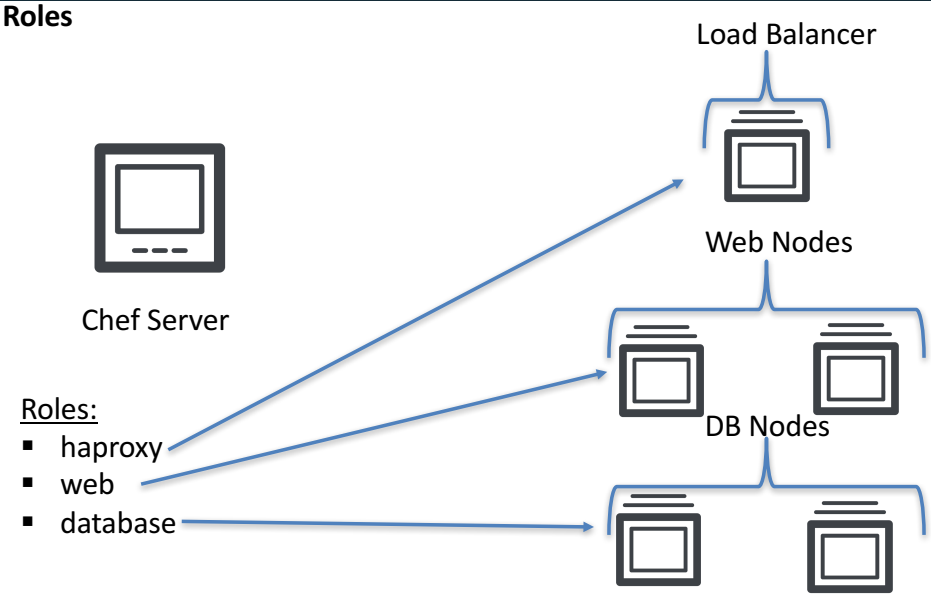
Linux Academy

Roles

```
{
 "name": ”haproxy",
 "description": ”Haproxy load balancer for webnodes ",
 "json_class": "Chef::Role",
 "default_attributes": {

 },
 "override_attributes": {

 },
 "chef_type": "role",
 "run_list": [
        ”recipe[haproxy]”,
        "role[monitoring]"
 ],
 "env_run_lists": {

 }
}
```

Linux Academy

Roles



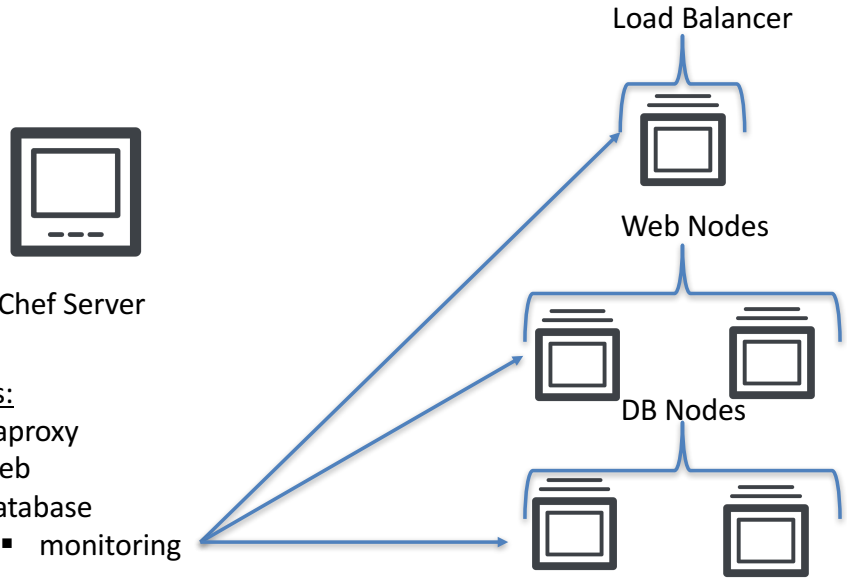
Linux Academy

What about role[monitoring] included in the db, haproxy, and web roles we created?

```
{
 "name": ”monitoring",
 "description": ”Recipes that make up the monitoring stack required for all nodes",
 "json_class": "Chef::Role",
 "default_attributes": {

 },
 "override_attributes": {

 },
 "chef_type": "role",
 "run_list": [
        ”recipe[nagios]”,
        "recipe[collectd]"
 ],
 "env_run_lists": {

 }
}
```

Linux Academy

Roles



Roles:

- haproxy
- web
- database
  - monitoring

Linux Academy

When you assign a role to a node you do so in its run list.

$ Knife node run_list set nodename “role[web]”

All recipes and roles assigned to the web role run list will be executed on this node.

Linux Academy

After making a change to the roles, how can we force the new run list to execute on nodes with a given role assigned to them?

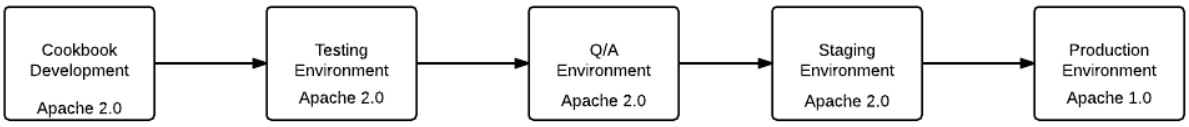
1. Wait for chef-client on the nodes to execute if chef-client is set to run at intervals
2. Execute knife ssh

Knife ssh ”role:web” “sudo chef-client” –x user –P password

Linux Academy

Implementing Web Node Apache Cookbook Changes

Scenario: You've been tasked with making changes to the Apache cookbook from 1.0 to 2.0 versions. These changes need to go through the Testing, Q/A, and Staging environments before production. How can we use the same roles for all environments, including production, but limit the cookbook version only to specific environments?



Linux Academy

Environments: Understanding Chef Environments

Every node belongs to the _default environment.

The _default environment cannot be modified.

Environments allow you to assign roles that contain a specific cookbook version to a group of nodes, while also allowing a certain cookbook version to execute given the environment “configurations”.

A more simple way of thinking about it is “generally every environment is associated with one or more cookbooks or cookbook verisons”

Linux Academy

Environments: Understanding Chef Environments

```
name ' Production'
description 'Production Environment'
cookbook 'apache', '= 1.0'
```

```
name ' Staging'
description ' Staging Environment Before Production'
cookbook 'apache', ’ = 2.0'
```

Linux Academy

Advantages and reasons of defining your infrastructure as code

- Flexibility
- Version control of infrastructure
- Human-readable infrastructure – the code is the documentation!
- Create testable infrastructures just like testable code!
- Easily scalable to thousands of systems, multiple clouds, and on-premises
- Use existing cookbooks created on Chef Supermarket as well as automate deployments and compliance

Linux Academy

Rolling back vs Rolling forward

Rolling backward: Roll back the environment or code to a previous state (Restore / Blue-Green)
Con: A roll back must succeed for ALL components in your environment or it fails. Generally a riskier method of fixing issues.

Rolling forward: Rolling forward means, understanding the current issue and implementing a permanent fix applied going forward. In Chef it is generally best practice to roll forward rather than roll backwards.

Linux Academy

Imperative vs Declarative

Chef is considered an imperative configuration management tool.

Ansible and Puppet are both considered declarative configuration management tools.

But what is the difference between Imperative and Declarative languages?

Imperative describes “how” you’re going to implement the code (in our case, the Chef configuration on a node).

Declarative describes “the end result” or the “desired result” but does not describe how to implement that result. That is left up to the executor.

Linux Academy

Imperative vs Declarative: Still Confused?

Think about SQL: SQL allows you to pull data from a database but does not specify “how” to pull the data. It only specifies “what” data to pull.

Puppet and Ansible: Specify the desired end result.

Chef: Specify how you want the configuration to be put in place (How to get to the end result).

Example: With Chef you can use “for each loops,” but with Puppet, it is not allowed.

Linux Academy

Imperative vs Declarative: Getting Clearer….

Chef Code (Ruby Syntax)	The How		
Chef DSL	The What	Puppet DSL	The What
Chef	The Executor	Puppet	The Executor

Linux Academy

Imperative vs Declarative: Getting Clearer….

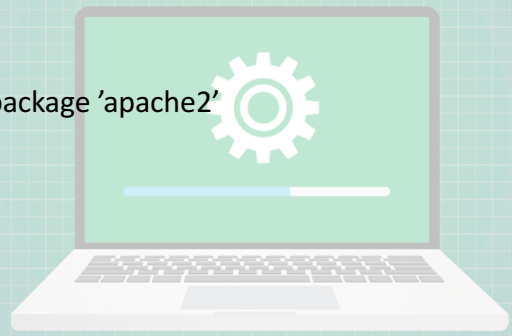
Imperative

```
If node['os_family'] == "debian"
   execute "install apache2" do
      command "apt-get install apache2 -y"
   end
end
```

Imperative approach provides more flexibility

Declarative

```
package 'apache2'
```



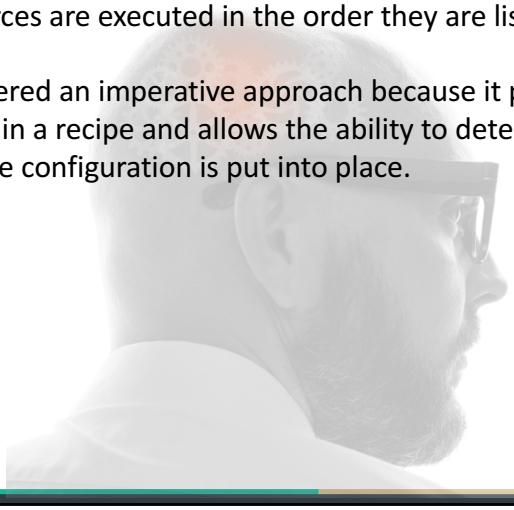
Declarative approach is easier to implement

Linux Academy

Imperative vs Declarative: Getting Clearer….

With Chef, resources are executed in the order they are listed in the page.

This is considered an imperative approach because it places resources correctly within a recipe and allows the ability to determine execution order and “how” the configuration is put into place.



Linux Academy

Certified Chef Developer

Push vs. Pull

Linux Academy

Push vs. Pull

Push deployment occurs when the deployment of infrastructure changes are pushed from a central location (server) to the nodes

Pull deployment is when each server polls/queries a central location for changes and applies those changes locally on the node

Chef does both! Up until this point we've been working with "pull deployments" because chef-client runs in intervals and pulls cookbook changes from the server.

Linux Academy

Push vs. Pull: Chef Push Jobs

Chef push jobs is a Chef server extension that allows jobs to be executed against a node, without the dependency of a chef-client run.

A separate agent is running on the node listening to the Chef push jobs server for any jobs that might be sent.

A job is an action or command that is to be executed against a set of nodes.

A search query made to the Chef server is what determines "which" nodes receive the push job.

Linux Academy

Push vs. Pull: Chef Push Jobs

Knife SSH

- Requires SSH keys or sudo permissions
- No corrective action if a job fails
- Hard to scale to large amounts of nodes
- Requires scripting if you want to schedule a command

Push Jobs

- An installed agent much like chef-client is listening to the chef server and requires no additional authentication
- Is a resource type that is managed with knife and recipes
- Can be used within recipes to orchestrate actions between nodes

Linux Academy

Windows DSC

Windows DSC is a PowerShell task-based command-line shell and scripting language that was developed by Microsoft.

DSC (Desired State Configuration) is a PowerShell feature which provides a set of language extensions, cmdlets, and resources.

Essentially, it's a tool for configuration management on Windows machines using PowerShell. The DSC is exposed as configuration data from within the Windows PowerShell but Chef uses Ruby.

Linux Academy

Windows DSC

The dsc_resource type allows the DSC to be used in Chef recipes along with any custom resources that have been added to the PowerShell environment.

Linux Academy

Removing Resources From A Recipe

Chef recipes are used to describe a desired state of configuration. If there a resource type is removed from within a recipe, then the resource it manages is no longer evaluated by the chef-client during a chef-client run.

The underlying resource will very simply stay in the state of the last chef-client run (unless it's changed manually)

Linux Academy

Certified Chef Developer

Chef Supermarket

Linux Academy

Chef Supermarket

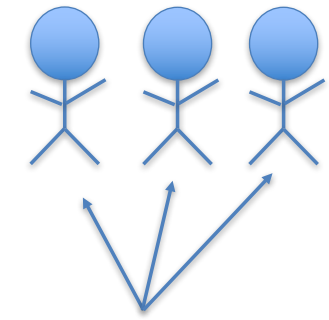
Meet Jeff!

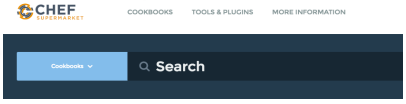
Jeff needs a cookbook to manage his Apache configuration

Jeff writes a cookbook to manage his Apache nodes

Jeff knows that someone else will need an Apache configuration, after all, the internet runs on webservers

After building the cookbook Jeff uploads it to Chef Supermarket, a site for community cookbooks





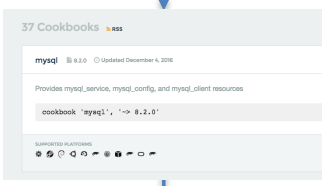
Linux Academy

Chef Supermarket

Jeff needs a cookbook to manage his MySQL nodes

Having contributed to the Chef Supermarket this is the first place Jeff looks

Jeff opens https://supermarket.chef.io to search for a MySQL cookbook



Jeff finds a MySQL cookbook, downloads it, and makes minor changes for his Nodes

Jeff was able to do all of this without writing a cookbook from scratch!

Be Like Jeff!

Linux Academy

Chef Supermarket

Chef Supermarket is 100% free, everything uploaded is available to any Chef user to use within any of their environments at no cost.

There is no shopping cart functionality.. Supermarket is free.

Chef Supermarket can be run on a machine, on your network, within your own environment, and at no additional cost.

The job of public Chef Supermarket is to host community built cookbooks.

Linux Academy

Chef Supermarket: Private Super Markets

A private Supermarket, located behind your firewall, can be installed on a local machine in your environment.

Using Berkshelf multiple Supermarket installations can be specified, for example, a local one, and the public Chef supermarket. In the event that a cookbook book is not available on one, it pulls from the other.

How can we modify cookbooks from Chef Supermarket without forking?!?

Cookbook Wrappers! (Covered later in this course)

Linux Academy

Chef Supermarket: More than just a cookbook repository!

Chef Supermarket is a site for uploading and downloading community built (and a lot of Chef built) cookbooks.

Cookbooks available on the Chef Supermarket available and accessible to all Chef users

Don't skip the video!, there is a lot more to Chef Supermarket than just hosting cookbooks!

- Chef Supermarket can be part of your deployment process
- Using Berkshelf, cookbooks can be deployed directly from the Chef Supermarket and use multiple installations of Chef Supermarket to help resolve dependency issues

Linux Academy

Chef Supermarket: Private Super Markets

A private Supermarket, located behind your firewall, can be installed on a local machine in your environment.

Using Berkshelf multiple Supermarket installations can be specified, for example, a local one, and the public Chef supermarket. In the event that a cookbook book is not available on one, it pulls from the other.

How can we modify cookbooks from Chef Supermarket without forking?!?

Cookbook Wrappers! (Covered later in this course)