CHEF OVERVIEW

CHEF

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

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.

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

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?

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.

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.

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

What happened here?

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

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
```

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
```

Installing Multiple Packages At One Time

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

Installing Multiple Packages At One Time

```
package 'httpd'
      action :install
End

package 'vim'
      action :install
End
```

package 'emacs' – No "do or end is required"

Installing Multiple Packages At One Time

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

DEFAULT RESOURCE ACTIONS

Default Resource Actions

package ‘httpd’

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

service ‘httpd’

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

file ‘/etc/motd’

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

CHEF EXTENSIBILITY

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

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

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
```

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
```