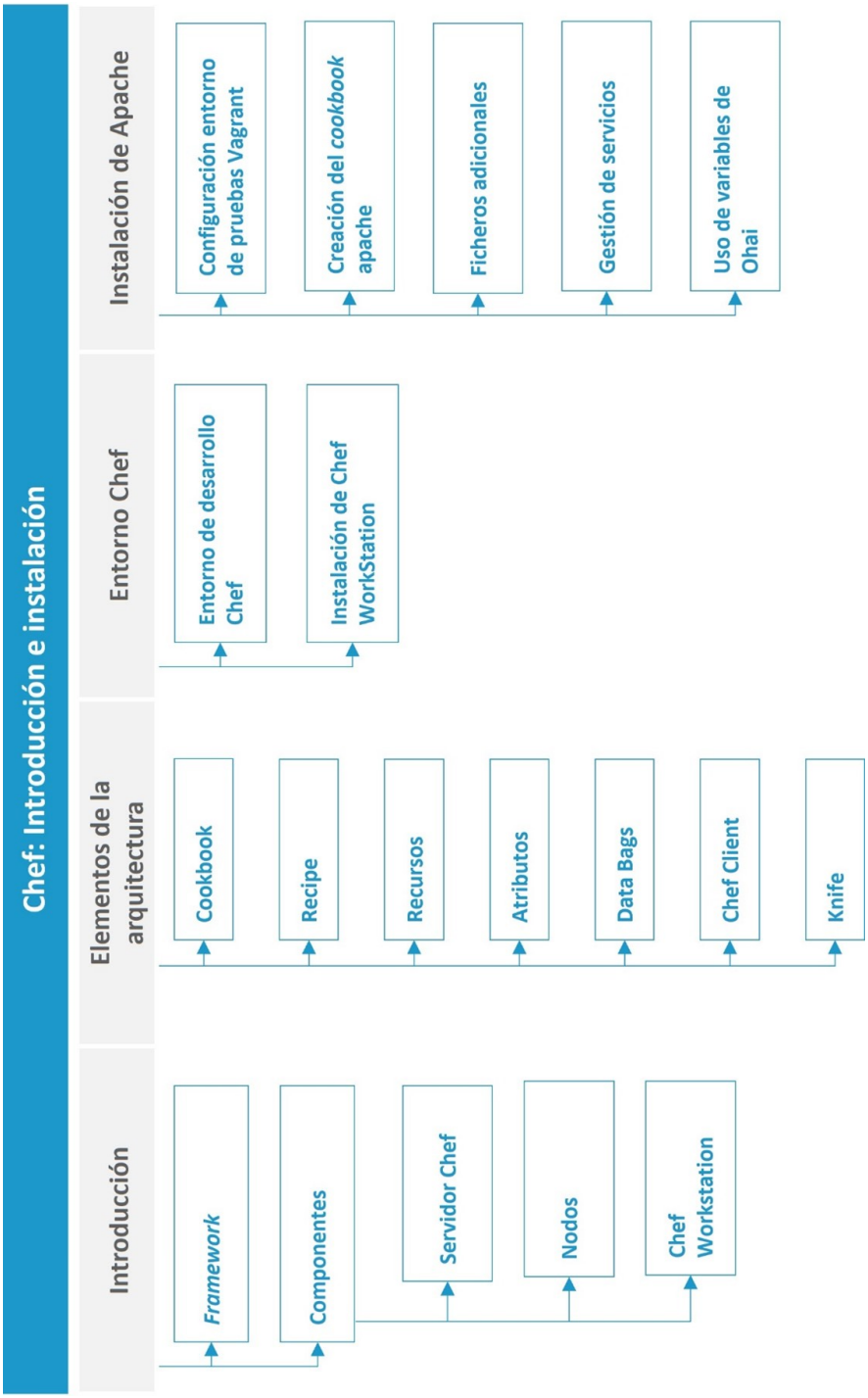
Herramientas de Automatización de Despliegues

Tema 3. Chef. Introducción e instalación

Índice



3.1. Introducción y objetivos

La gestión de la **infraestructura como código** viene a referirse tanto a la escritura de código, mediante un lenguaje descriptivo cualquiera, para el control y aprovisionamiento de servidores y su proceso de configuración, como al despliegue del código de las aplicaciones que se ejecutan en dichos servidores. Se trata además de la **adopción de pruebas automatizadas,** así como otra serie de prácticas que ya se utilizan ampliamente desde hace tiempo en el desarrollo de aplicaciones, entre las que podemos mencionar el control de versiones, las pruebas de integración, el uso de patrones de diseño, etc.

A la hora de aplicar estas mismas prácticas y normas, hemos de asegurarnos de proporcionar y gestionar la configuración de servidores mediante código, para, de esta forma, construir procesos seguros, repetibles y automatizados.

Los objetivos de este tema son los siguientes:

- Introducir la herramienta Chef.
- Explicar e instalar su entorno de desarrollo (CDK).
- Usar Chef para instalar Apache.

3.2. El framework Chef

En el proceso de maduración del desarrollo de software han surgido los denominados marcos de trabajo, más conocidos por su nombre en inglés: *frameworks*. Su objetivo principal es el de simplificar y reducir el tiempo de desarrollo, permitiendo a los desarrolladores poder centrarse en el desarrollo específico que cumpla con los requisitos necesarios, sin tener que preocuparse de elementos y servicios de uso común que son proporcionados por el *framework*, y lograr así una entrega más rápida. Estos *frameworks* de software suelen ofrecernos además un conjunto común de convenciones, procedimientos y servicios que tienen como objetivo fomentar que el equipo desarrolle de forma homogénea, facilitando así el desarrollo sostenible del proyecto.

Chef es uno de los *frameworks* más extendidos para la gestión de la infraestructura como código, proporcionando una extensa biblioteca de primitivas y utilidades para la gestión de todos los recursos que se usan en los procesos de construcción y mantenimiento de la infraestructura de sistemas. Chef utiliza un lenguaje específico de dominio (DSL), que está basado en el lenguaje de programación Ruby, y proporciona una capa de abstracción sobre los recursos que permite, tanto a un administrador de sistemas como a un desarrollador, definir y aprovisionar fácilmente entornos escalables.

Componentes Chef

El *framework* Chef está compuesto por tres componentes básicos que interactúan entre sí: el **servidor Chef,** las máquinas gestionadas denominadas **nodos** y la **estación de trabajo Chef** (Chef *workstation*).

Servidor Chef

El servidor Chef constituye el **centro de los datos de configuración,** que contiene

los *cookbooks*, que son los módulos de configuración de recursos, las políticas que se aplicarán a los diferentes nodos, y los metadatos, que describen cada uno de los nodos administrados por Chef.

Nodos

Los nodos usan la herramienta **Chef Client** para conectarse con el Chef Server y solicitar los detalles de configuración que deben aplicarse sobre la máquina donde se ejecutan. A este proceso de aplicar los cambios sobre los nodos se le denomina **Chef run.**

Cabe destacar que cuando un nodo se crea y se registra en el Chef Server, el Chef Client se instala en el proceso de arranque, para iniciarse cuando la máquina se levanta.

En Chef, se llama **rol** a un conjunto de atributos y una lista de recetas para el nodo. Cada nodo puede tener uno o más roles, y estos pueden ser reutilizado por varios nodos, por lo que se puede definir un conjunto de nodos que tengan el mismo rol dentro de nuestro sistema. La **lista de ejecución** se refiere a la lista de recetas asociadas con un nodo a través del rol o las propias recetas de las que depende, y el orden de ejecución será el mismo orden en que está definida.

Chef Workstation

La estación de trabajo Chef, también denominada «repositorio de Chef (Chef-repo)», contiene la estructura del proyecto gestionado por Chef y lo maneja el desarrollador o administrador desde su *workstation*. Todos los componentes de Chef que constituyen nuestro sistema están definidos aquí: *cookbooks*, entornos, roles y pruebas. Una buena práctica que es frecuente es la de mantener el Chef-repo en un sistema de control de versiones, para así gestionarlo como si fuera código fuente de una aplicación.

La estructura de directorios de un Chef-repo puede variar, ya que algunos usuarios

prefieren un único Chef-repo para mantener todos los *cookbooks*, mientras que otros prefieren almacenar cada *cookbook* en un repositorio separado, pero, en cualquier caso, el repositorio Chef (Chef-repo) debe poder gestionar toda la infraestructura y, para ello, debe contener la información de todas sus partes.

3.3. Elementos de la arquitectura de Chef

La lista que se enumera a continuación describe los principales elementos de la arquitectura de Chef:

- **Chef Server:** servidor centralizado que contiene y gestiona la configuración de todos los nodos. Puede estar ubicado en un servidor independiente o alojado como servicio en la nube en Chef.
- **Nodo:** incluye la información relativa a las recetas y roles que deben aplicarse en la máquina durante la ejecución de Chef Client. Los atributos y la lista de ejecución del nodo son sus principales características.
- ***Cookbook:*** contiene todos los recursos e instrucciones que se necesitan para configurar nodos, y pueden ser reutilizados en varias listas de ejecución. Los *cookbooks* se componen habitualmente de varias recetas.
- ***Recipes*** (recetas): consiste en un conjunto de recursos que configuran un nodo, y es una de las partes fundamentales de Chef.
- **Recursos:** son una abstracción multiplataforma de partes configurables de un nodo, como pueden ser usuarios, paquetes, archivos o directorios.
- **Atributos:** representan la configuración del nodo, como por ejemplo el nombre del *host*, las versiones de los lenguajes de programación para instalar, servidor de base de datos, etc.
- ***Data Bags:*** contienen conjuntos de datos que están disponibles globalmente y pueden ser utilizados por los nodos y roles.
- **Chef Client** (Cliente Chef): realiza el trabajo necesario en nombre del nodo, donde ejecuta las recetas correspondientes para configurar e instalar el *software*.
- **Repositorio Chef:** componente donde se ubica el repositorio de los *cookbooks*,

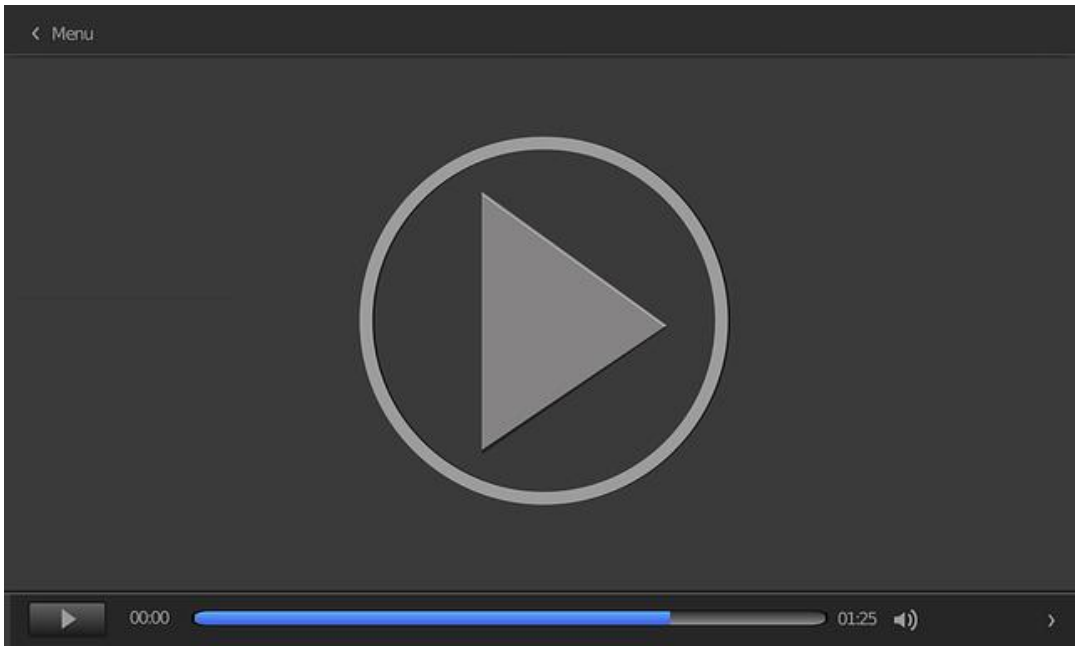
roles, archivos de configuración y otros artefactos que maneja el usuario.

- **Chef Solo:** herramienta de línea de comandos que permite ejecutar *cookbooks* Chef independientemente, sin conectarse a un servidor Chef. Es una versión de código abierto del Chef Client.

- ***Knife*** (cuchillo): herramienta de línea de comandos que utilizan los usuarios para cargar los ficheros de configuración en el servidor Chef.

3.4. Entorno de desarrollo Chef

A continuación, puedes ver el vídeo *Instalación de Chef*:



Accede al vídeo:https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=d97c5b3a-664d-40cd-b557-abb600a58e65

El repositorio Chef es el componente con el que se suele trabajar habitualmente, realizando las siguientes tareas:

- Desarrollo de *cookbooks* y *recipes* (recetas).
- Carga de información en el servidor Chef.
- Sincronización del código fuente del repositorio Chef con el control de versiones.
- Administración de las opciones de configuración y de los entornos.
- Interacción y aprovisionamiento de los nodos nuevos y los ya existentes.

Ideas clave

El cometido fundamental de una estación de trabajo en una instalación de Chef es el de sincronizar los datos del repositorio Chef con el servidor Chef, así como con todos los nodos con los que interactúa.

Dentro del conjunto estándar de herramientas que incorpora Chef se incluyen dos herramientas fundamentales de línea de comandos: `knife`(cuchillo), que es la herramienta de línea de comandos que interactúa con el servidor Chef y la sincronización de los nodos, y el propio comando chef, que trabaja con los componentes del repositorio Chef y la gestión del entorno de Ruby embebido utilizado por Chef.

Además de las herramientas estándar que proporciona Chef, existe una serie de herramientas específicas adicionales que han sido ampliamente adoptadas por la comunidad, tales como:

- **Berkshelf:** gestor de dependencia de *cookbooks*.
- **Test Kitchen:** *framework* de pruebas de integración.
- **ChefSpec:** *framework* de pruebas unitarias.
- **Foodcritic:** herramienta para la realización de análisis de código estático en *cookbooks*.

Instalación de Chef Workstation

Anteriormente, de cara a que la instalación de un entorno de desarrollo Chef fuera lo más sencilla posible, contábamos con el **kit de desarrollo de Chef** (ChefDK), elaborado por la propia comunidad Chef. Este proyecto tenía como objetivo recopilar las herramientas Chef más ampliamente adoptadas en un solo paquete de fácil instalación, y con soporte para múltiples plataformas.

Actualmente contamos con Chef Workstation, desarrollada esta vez por la propia organización Chef Software, que incluye todas las funcionalidades de ChefDK

además de otras nuevas, y que es el producto que seguirá recibiendo actualizaciones y mejoras en el futuro. Esta nueva herramienta sigue siendo gratuita para uso no comercial, por lo que para nuestro caso es recomendable instalarla, aunque ya contásemos con ChefDK, para obtener las últimas versiones soportadas de las herramientas incorporadas, así como nuevas funcionalidades.

Para instalarlo, basta con descargar la versión correspondiente a nuestro sistema operativo desde la página de descargas e instalar el paquete siguiendo los pasos de instalación.

Puedes acceder a la descarga de Chef Workstation a través del enlace:

https://downloads.chef.io/chef-workstation/

Para revisar todos los componentes instalados por la instalación, podemos ejecutar el comando chef con el argumento `-v` para que liste las versiones de los componentes:

```
$ chef -v
```

```
Chef Workstation version: 20.6.62
```

```
Chef Infra Client version: 16.1.16
```

```
Chef InSpec version: 4.19.0
```

```
Chef CLI version: 3.0.4
```

```
Test Kitchen version: 2.5.1
```

```
Cookstyle version: 6.7.3
```

Para obtener instrucciones más detalladas sobre este proceso, puedes consultar la documentación oficial.

Ideas clave

Una vez realizada la instalación satisfactoriamente, ya estamos preparados para comenzar a usar Chef para gestionar la infraestructura. Ahora tenemos todas las herramientas necesarias para desarrollar y probar recetas, cargarlas en el servidor y aprovisionar nodos.

3.5. Instalación de Apache mediante Chef

De cara a poder demostrar las características fundamentales de Chef para el manejo de la infraestructura como código, vamos a ir desarrollando un ejemplo que se encarga de instalar un servidor HTTP Apache en una máquina virtual aprovisionada con Vagrant, de manera semejante a lo que hemos hecho en temas anteriores.

Lo primero será preparar un entorno de pruebas mediante Vagrant, que nos servirá para generar una máquina virtual que será la que aprovisionemos con Chef, y sobre la que iremos incorporando diferentes recursos hasta completar una instalación de Apache con una configuración personalizada y una página de inicio propia.

Configuración de un entorno de pruebas con Vagrant

Nuevamente vamos a utilizar Vagrant para preparar nuestro entorno de pruebas con una máquina virtual que utilizaremos para desarrollar el ejemplo, aprovisionándolo fácilmente con Chef a través de los comandos Vagrant. Ejecuta los siguientes comandos desde un terminal:

```
mkdir chef-apache && cd chef-apache
```

```
vagrant init ubuntu/bionic64
```

Vamos a configurar la red con una dirección IP privada que establezcamos nosotros mismos, y a asignar mayor cantidad de memoria a la máquina virtual, para que la ejecución de Chef y las diferentes operaciones de configuración sean más fluidas. Para ello, incluimos el siguiente fragmento en el `Vagrantfile`:

```
config.vm.network "private_network", ip: "192.168.33.40"
```

```
config.vm.provider "virtualbox" do |vb|
```

```
vb.memory = "1024"

 end
```

A continuación, incluiremos la configuración necesaria para aprovisionar nuestra máquina virtual con Chef. A diferencia de lo que pasaba con Puppet, Vagrant es capaz de instalar automáticamente Chef en la máquina virtual si detecta que no está instalado y hemos especificado `chef_solo` como el aprovisionador a utilizar. Este aprovisionador permite aprovisionar una máquina con Chef, sin la necesidad de contar con un servidor al que conectar el nodo, por lo que es el más apropiado para nuestro caso.

Incluimos el siguiente fragmento en el `Vagrantfile`:

```
config.vm.provision :chef_solo do |chef|

 chef.<em>cookbooks</em>_path = "chef-repo/<em>cookbooks</em>"

 chef.add_recipe "chef_apache"

 chef.arguments = "--chef-license accept"

 end
```

Aquí estamos indicando que el aprovisionador a usar será `chef_solo`, indicándole la ruta donde almacenamos las recetas y la receta a ejecutar. El último parámetro sirve para especificar argumentos en la línea de comandos al ejecutar Chef, y en este caso nos permite añadir el argumento para aceptar la licencia de Chef, que es necesario realizar al ser una nueva instalación de Chef la que se realiza en la máquina virtual. Esta licencia nos permite usar Chef para aprendizaje y experimentación, y cualquier uso comercial debe contar con un acuerdo de licencia comercial con Chef.

En definitiva, el fichero de configuración Vagrant (`Vagrantfile`) para nuestra máquina virtual quedará así:

```
Vagrant.configure("2") do |config|

 config.vm.box = "ubuntu/bionic64"

 config.vm.network "private_network", ip: "192.168.33.40"

 config.vm.provider "virtualbox" do |vb|

 vb.memory = "1024"

 end

 config.vm.provision :chef_solo do |chef|

 chef.<em>cookbooks</em>_path = "chef-repo/<em>cookbooks</em>"

 chef.add_recipe "chef_apache"

 chef.arguments = "--chef-license accept"

 end

end
```

Una vez creado el fichero de configuración Vagrant, ya podemos preparar el *cookbook* Chef necesario para el aprovisionamiento, que describiremos a lo largo de la siguiente sección.

Creación del *cookbook* apache

Lo primero que vamos a hacer es crear las rutas y ficheros que hemos especificado en el `Vagrantfile` para que cuando ejecute Vagrant se encuentre todos los recursos

que necesita para aprovisionar.

```
mkdir -p chef-repo/cookbooks/ && cd chef-repo/cookbooks/
```

Una vez creado el directorio de *cookbooks*, para generar nuestro *cookbook* utilizaremos el comando chef correspondiente, que creará toda la estructura de directorios y ficheros que se requiere.

```
chef generate cookbook apache
```

Si es la primera vez que ejecutamos un comando chef, nos pedirá que aceptemos la licencia de los productos asociados, para poder comenzar a utilizar sus funcionalidades.

Una vez aceptadas las licencias, o si ya se habían aceptado anteriormente, el comando creará un directorio apache con la siguiente estructura:

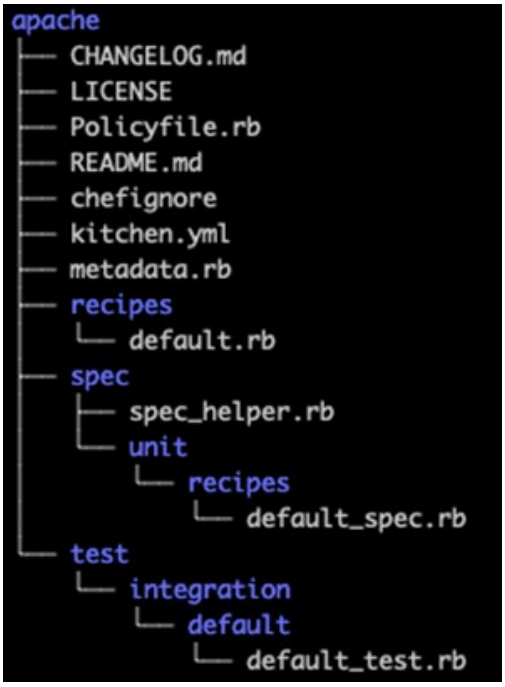


Figura 1. Estructura de directorios de un rol. Fuente: elaboración propia.

En esta estructura se pueden encontrar varios ficheros que se generan por defecto y

que nos sirven de plantilla o versión inicial para incorporar funcionalidad a nuestro *cookbook*. El fichero principal donde se especifica la configuración a obtener es la receta por defecto, ubicado en: `recipes/default.rb`, al que vamos a incluir el primer fragmento de código:

```
apt_update 'Update the apt cache daily' do
 frequency 86400
 action :periodic
end
package 'apache2'
```

Aquí hemos definido un recurso de tipo `apt_update` que se encargará de refrescar la caché del gestor de paquetes, y como argumentos le indicamos que sea una acción periódica con una frecuencia de 86 400 segundos (1 día, que es el valor por defecto, pero se ha incluido aquí por claridad). Esto indica a Chef que, si al aplicar la configuración, esta acción no se ha realizado hace menos de 1 día, se volverá a ejecutar. El siguiente recurso es de tipo `package` y, si no especificamos ningún argumento, le indica a Chef que el paquete apache2 debe estar instalado, que es la acción por defecto.

Esto hará que cuando se aprovisione nuestra máquina virtual se instalará Apache con la configuración por defecto, por lo que si ejecutamos ahora vagrant up desde la ruta del fichero Vagrantfile comprobaremos que, una vez creada la máquina, el aprovisionador Chef instalará Apache. Una vez que finalice el aprovisionamiento, podremos acceder desde un navegador a la dirección http://192.168.33.40 y comprobar cómo se muestra la página por defecto de Apache.

A continuación, vamos a sustituir esa página por defecto, para lo cual tendremos que

cambiar la configuración del servidor Apache e indicarle qué pagina utilizar, además de proporcionarle dicha página. Añadimos lo siguiente a nuestra receta:

```
file '/etc/apache2/sites-enabled/000-default.conf' do

 action :delete

end

template '/etc/apache2/sites-available/vagrant.conf' do

source 'virtual-hosts.conf.erb'

end

link '/etc/apache2/sites-enabled/vagrant.conf' do

to '/etc/apache2/sites-available/vagrant.conf'

end

cookbook_file “/vagrant/index.html" do

source 'index.html'

only_if do

File.exist?('/etc/apache2/sites-enabled/vagrant.conf')

end

end
```

El primero de los recursos añadidos es de tipo `file` y nos permite gestionar los ficheros del nodo. En este caso, el argumento `action` nos indica que deseamos

borrar el fichero, por lo que, si el fichero especificado en la ruta se encuentra disponible, se eliminará.

El segundo de los recursos es de tipo `template` que nos permite procesar una plantilla del servidor y guardar el fichero generado en la ruta de recurso especificada. El argumento que se utiliza en este caso es `source`, que proporciona la ruta de la plantilla a utilizar. La plantilla tiene la extensión `erb`, que indica el formato de plantillas de Ruby, y se buscará en el subdirectorio `templates` dentro del *cookbook*.

El tercero de los recursos es de tipo link y se utiliza para generar un enlace simbólico en la máquina destino, indicando en el recurso la ruta del enlace a crear, y en el argumento `to` el fichero a enlazar.

El cuarto recurso añadido arriba es de tipo `cookbook_file` y permite copiar un fichero desde el directorio del *cookbook* al nodo, especificando en el recurso la ruta destino y en el argumento `source` el fichero a utilizar, que se buscará partiendo del subdirectorio files dentro del *cookbook*. También podemos apreciar que se incluye una sentencia `only_if`, que sirve para condicionar la ejecución del recurso a que se cumpla la condición especificada en la sentencia, que en este caso consiste en preguntar por la existencia del fichero que debería haber creado el paso anterior, por lo que, si todo ha ido bien, la condición será cierta y se ejecutará el recurso. Este tipo de sentencia condicional se llama en Chef guarda (`guard`), y también contamos con la inversa `not_if`, que no ejecutará el recurso si se cumple la condición.

En este último recurso hemos utilizado la ruta `/vagrant` como destino del fichero `index.html` que hemos copiado. Esta ruta también será necesario utilizarla en el fichero de configuración de Apache para indicarle dónde están los ficheros de la web. Sería bueno poder definir una variable que nos permitiera establecer este valor una única vez y poderlo referenciar desde ambos sitios.

Por tanto, necesitamos definir una variable, a la que Chef llama atributo, para poder especificar el valor que deseamos utilizar como raíz de nuestra página, para lo cual

vamos a crear el fichero `attributes/default.rb` en nuestro *cookbook* con el siguiente contenido:

```
default['apache']['document_root'] = "/vagrant"
```

Esto define un atributo de tipo default (su valor se inicializa cada vez que el Chef Client ejecuta la configuración de su nodo) en la colección apache, denominado `document_root` y con el valor "`/vagrant`". El recurso `cookbook_file` ahora sería:

```
cookbook_file "#{node['apache']['document_root']}/index.html" do
 source 'index.html'
 only_if do
 File.exist?('/etc/apache2/sites-enabled/vagrant.conf')
 end
end
```

Tal como se muestra, la referencia al atributo se hace con el símbolo de la almohadilla (#) y, entre llaves ({}), el nombre del atributo dentro del objeto nodo.

Ficheros adicionales

En los recursos anteriores hemos referenciado un par de ficheros que deberían estar contenidos en el *cookbook*; la plantilla que genera el fichero de configuración de Apache y el fichero `index.html` que se utilizará como página por defecto. Estos ficheros deben estar ubicados en las rutas en las que Chef espera encontrarlos, para que la ejecución de la configuración no dé errores y pueda ejecutar todos los pasos.

Para generar dichos ficheros, nos situamos en el directorio de nuestro *cookbook* apache y ejecutamos los siguientes comandos:

```
mkdir files && mkdir templates
```

```
touch files/index.html && touch templates/virtual-hosts.conf.erb
```

El fichero de plantilla (`virtual-hosts.conf.erb`) tendrá el siguiente contenido:

```
<VirtualHost *:80>
 ServerAdmin webmaster@localhost
 DocumentRoot <%= node['apache']['document_root']%>
 <Directory <%= @node['apache']['document_root']%>>
 Require all granted
 </Directory>
 ErrorLog ${APACHE_LOG_DIR}/error.log
 CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Como podemos apreciar, aparece un par de veces la referencia mediante la sintaxis de expresiones Ruby al atributo `node[:apache][:document_root]` que habíamos definido en el fichero de atributos, que sigue el formato: `<%= expresión %>`

Esto le indica a Chef que debe sustituir esa referencia por el valor del atributo definido. Cabe señalar que, en este caso, también podemos ver otro tipo de referencia de variables, tal como `${APACHE_LOG_DIR}`, pero esta sintaxis está

referenciando una variable de entorno. En este caso, Chef no lo interpretará y lo copiará tal cual al fichero destino, y quien lo interprete será Apache cuando lea el fichero de configuración resultante.

El otro fichero al que hacemos referencia es `index.html`, que se copiará tal cual desde el origen a la ruta destino. El contenido de ejemplo es el siguiente:

```
<html>

 <head>

 <title>Ejemplo UNIR</title>

 </head>

 <body>

 <h1>Hola hola, alumnos de UNIR!</h1>

 </body>

</html>
```

Gestión de servicios

Si ahora ejecutamos `vagrant provision`, veremos que aparece en la salida la ejecución de los nuevos recursos de ficheros, pero si accedemos a la IP de la máquina virtual desde un navegador, no veremos cambios y se seguirá mostrando la página por defecto de Apache. Esto es debido a que el servicio de Apache no ha recargado la configuración y, por tanto, no se ha enterado de los cambios. Vamos a incluir un servicio recurso detrás de la instalación del paquete `apache2` en nuestra receta:

```
service 'apache2' do
```

```
 supports :status => true

 action :nothing

end
```

En este recurso indicamos el nombre del servicio, en nuestro caso, `'apache2'`, y como argumentos estamos proporcionando supports para indicar que el estado del servicio se puede determinar mediante la opción `status` del sistema operativo, y action con el valor `:nothing` para especificar que no queremos hacer nada cuando Chef lea el recurso, ya que vamos a incluir notificaciones en los recursos que requieren que la configuración se recargue, y en ese momento es cuando el recurso ejecutará la acción que corresponda. Modificamos los siguientes recursos para añadir la notificación:

```
template '/etc/apache2/sites-available/vagrant.conf' do

 source 'virtual-hosts.conf.erb'

notifies :restart, resources(:service => "apache2")

end

link '/etc/apache2/sites-enabled/vagrant.conf' do

to '/etc/apache2/sites-available/vagrant.conf'

notifies :restart, resources(:service => "apache2")

end
```

Hemos incluido la notificación tanto en el fichero de configuración, como en el enlace que lo apunta, ya que así cualquier cambio en alguno de los dos recursos provocará

la recarga de la configuración de Apache. Por defecto, aunque se disparen varias notificaciones en una ejecución, solo se recargará el servicio una vez al final de la ejecución de la configuración *(Chef Client run),* aunque se puede especificar un temporizador distinto a la notificación para que se ejecute inmediatamente (`:immediate`), o incluso antes de la ejecución del recurso que lo incluye (`:before`). Podríamos haber añadido la notificación al recurso que cambia el fichero `index.html`, pero esto no es necesario, ya que los cambios en este fichero los detectará Apache, y siempre servirá la última versión disponible.

Si has ejecutado vagrant provision tras la inclusión de los ficheros de configuración, y dado que estos no han variado, si lo vuelves a ejecutar ahora no se disparará el reinicio del servicio. Es un buen momento para realizar todo el proceso desde cero y validar que todo funciona correctamente, así que procedemos a ejecutar `vagrant destroy` y confirmar la acción, para posteriormente ejecutar `vagrant up` y crear una nueva máquina virtual. A continuación, se muestra el contenido completo de la receta principal de nuestro *cookbook* apache, una vez realizadas todas las modificaciones:

```
apt_update 'Update the apt cache daily' do
 frequency 86400
 action :periodic
end
package 'apache2'
service 'apache2' do
 supports :status => true
 action :nothing
```

```
end

file '/etc/apache2/sites-enabled/000-default.conf' do

 action :delete

end

template '/etc/apache2/sites-available/vagrant.conf' do

 source 'virtual-hosts.conf.erb'

 notifies :restart, resources(:service => "apache2")

end

link '/etc/apache2/sites-enabled/vagrant.conf' do

 to '/etc/apache2/sites-available/vagrant.conf'

 notifies :restart, resources(:service => "apache2")

end

cookbook_file "#{node['apache']['document_root']}/index.html" do

 source 'index.html'

 only_if do

 File.exist?('/etc/apache2/sites-enabled/vagrant.conf')

 end

end
```

Uso de variables de Ohai

Al igual que otras herramientas de gestión de la configuración, Chef cuenta con un mecanismo para recopilar datos de los nodos y hacerlos disponibles a través de atributos, a los que habitualmente se conoce como *facts*. La herramienta de Chef que nos facilita estas variables es `Ohai`, y vamos ahora a incluir otra pequeña receta que demuestra cómo se utilizan algunos de los atributos que `Ohai`pone a nuestra disposición. Para ello, creamos en el mismo directorio de recetas el fichero `facts.rb` con el siguiente contenido:

```
log 'Showing machine Ohai attributes' do
```

```
 message "Machine with #{node['memory']['total']} of memory and #
{node['cpu']['total']} processor/s. \
```

```
 \nPlease check access to http://#{node[:network][:interfaces]
[:enp0s8][:addresses].detect{|k,v| v[:family] == 'inet'}.first}"
```

```
end
```

Este recurso log mostrará un mensaje que incluye la cantidad de memoria disponible en la máquina virtual (`node['memory']['total']`y el número total de procesadores `(node['cpu']['total'])`. Para obtener el atributo que almacena la dirección IP —que, como podemos ver, es un atributo cuya ubicación no es nada evidente dentro del objeto—, hemos accedido primero a la máquina virtual mediante `vagrant ssh` y ejecutado el comando `ohai` directamente, que mostrará por la salida estándar el contenido del objeto nodo que construye con todos sus atributos. Una vez analizada la salida, hemos construido la expresión Ruby que da acceso al atributo correspondiente cuya familia es de tipo `inet` dentro del objeto. Hemos utilizado esta fórmula porque la IP por defecto de la máquina virtual (disponible directamente en el atributo `'ipaddress'` es privada y no accesible desde el *host*).

Ahora nos faltaría referenciar esta receta para que se ejecute, para lo cual, al final de nuestra receta por defecto, que es la que se ejecuta el referenciar al *cookbook*, incluimos lo siguiente:

```
include_recipe '::facts'
```

Esta sentencia le indica a Chef que queremos incluir el contenido de la receta indicada en este punto de la ejecución, como si incorporásemos todo su contenido en el punto donde se incluye. Para referenciar a una receta se utiliza el nombre de *cookbook* y, si la receta es diferente a la de por defecto, el separador `::` y el nombre de la receta. Como en nuestro caso es el mismo *cookbook*, no hace falta especificar su nombre, pero sí debemos incluir el separador y, a continuación, el nombre de nuestra receta.

Si volvemos a ejecutar `vagrant provision` ahora, comprobaremos que la configuración de nuestra máquina no varía (ya se había realizado previamente), pero ahora la salida muestra la ejecución de la nueva receta, y su mensaje con el valor de los *facts* del sistema.

3.6. Chef Supermarket

Chef, como otras herramientas de gestión de la configuración similares, cuenta con un repositorio compartido de *cookbooks* donde cualquier usuario de la comunidad puede publicar sus *cookbooks* o consultar los que hay disponibles, alrededor de 4000, para hacer uso de ellos. Este repositorio es el Chef Supermarket.

Cada *cookbook* publicado consta de información relativa a su descripción, requisitos, atributos, versión, fecha de publicación, dependencias, etc., con lo que tendremos una información muy valiosa para determinar si el *cookbook* en cuestión cumple en todo o en parte la función que andamos buscando.

Un aspecto interesante que incluyen los *cookbooks* publicados en el Chef Supermarket es una medida de la calidad del *cookbook*, calculada a través de una serie de métricas automáticas que comprueban si contiene indicaciones para pruebas, diferentes colaboradores, incorpora guía para contribuciones, plataformas soportadas, pasa el análisis de código estático, etc. Aunque esta medida no puede sustituir una prueba real de las capacidades y uso del *cookbook*, puede proporcionar una idea del grado de calidad con el que ha sido desarrollado.

Cabe señalar que el Chef Supermarket no solo consta de *cookbooks* publicados, sino que además incorpora una sección de herramientas y *plugins*, donde podemos encontrar otra serie de utilidades y elementos adicionales que enriquecen el ecosistema de Chef.

3.7. Referencias bibliográficas

Chef Software. (2020). *Chef Documentation*. https://docs.chef.io/

Marschall, M. (2015). *Chef Infrastructure Automation Cookbook*, 2nd Edition. Packt Publishing.

Waud, E. (2016). *Mastering Chef Provisioning*. Packt Publishing.

Documentación oficial de referencia de Chef

Chef Software. (2020). *Chef Documentation*. https://docs.chef.io/

Este es el sitio de la documentación oficial de referencia de Chef, donde encontrarás la información más actualizada y precisa sobre la herramienta, mantenida por la propia empresa que desarrolla Chef.

Referencia sobre recursos en Chef

Chef Software. (2020). *All Infra Resources*. https://docs.chef.io/resources/

Enlace a la documentación oficial sobre todos los recursos disponibles en Chef, su funcionalidad y características comunes, además de su sintaxis y propiedades específicas.

Referencia sobre atributos en Chef

Chef Software. (2020). *About attributes*. https://docs.chef.io/attributes/

Enlace a la documentación oficial sobre atributos (variables) en Chef, donde puedes encontrar sus tipos, fuentes, precedencia, etc.

A Beginner's Guide to Chef

Krout, E. (2021, mayo 21). *A Beginner's Guide to Chef*. Linode. https://www.linode.com/docs/applications/configuration-management/beginners-guide-chef/

Aquí podemos encontrar un tutorial para principiantes en Chef bastante sencillo y asequible, que puede servir como punto de partida o complemento a esta documentación.

Chef Supermarket

Chef Software. (2020). *Chef Supermarket*. https://supermarket.chef.io/

Repositorio compartido de *cookbooks* de Chef, donde cualquier usuario de la comunidad puede consultar los *cookbooks* publicados y descargarse los que desee probar o utilizar, así como también publicar sus propios *cookbooks*.

Test

1. ¿Cuáles son los componentes básicos del framework Chef?

A. Servidor Chef, cliente y nodos.

B. Servidor Chef, nodos y *workstation* Chef.

C. Cliente Chef, nodos y *workstation* Chef.

D. Servidor Chef y nodos.

2. ¿Con qué otro nombre se denomina al Chef Workstation?

A. Repositorio Chef o Chef-repo.

B. Servidor Chef.

C. Cliente Chef.

D. Plataforma Chef o *Chef-platform*.

3. ¿Cuál de las siguientes afirmaciones es más correcta con respecto al servidor Chef?

A. Servidor centralizado que contiene y gestiona la configuración de todos los nodos.

B. Repositorio de los *cookbooks*, roles, archivos de configuración y otros artefactos que maneja el usuario.

C. Puede estar ubicado en un servidor independiente o alojado como servicio en la nube en Chef.

D. A y C son correctas.

4. ¿Qué es un *cookbook* en Chef?

A. Una abstracción multiplataforma de partes configurables de un nodo, como pueden ser usuarios, paquetes, archivos o directorios.

B. Contiene conjuntos de datos que están disponibles globalmente, y puede ser utilizado por los nodos y roles.

C. Colección reutilizable de recursos e instrucciones que se necesitan para configurar nodos. Se componen habitualmente de varias recetas.

D. A y B son correctas.

5. ¿Cómo podemos procesar un fichero de plantilla en Chef?

A. Mediante un recurso de tipo `template` y el argumento `source` para especificar la plantilla a procesar.

B. Mediante un recurso de tipo `file` y el argumento `template` para especificar la plantilla a procesar.

C. Mediante un recurso de tipo `template` y el argumento `file` para especificar la plantilla a procesar.

D. Mediante un recurso de tipo `file` y la función `template` para especificar la plantilla a procesar.

6. ¿Cómo podemos condicionar la ejecución de un recurso en Chef?

A. Mediante una guarda `notif` en el recurso.

B. Mediante una guarda `onlyif` en el recurso.

C. Mediante una condición `unless` en el recurso.

D. A y B son correctas.

7. ¿Dónde es más adecuado declarar una variable o atributo en un *cookbook*?

A. En el fichero `vars/default.rb`

B. En el fichero `default/vars.rb`

C. En el fichero `default/attributes.rb`

D. En el fichero `attributes/default.rb`

8. ¿Cómo se notifica a un servicio para su reinicio desde un recurso?

A. Mediante la propiedad `service` con la acción `:restart`

B. Mediante la propiedad `notifies` con la acción `:restart`

C. Mediante la propiedad `service` con la acción `:reload`

D. Mediante la propiedad `restart` con la acción `:service`

9. ¿Cómo podemos acceder a los *facts* de un nodo proporcionados por Ohai?

A. Como a cualquier otro atributo, a través del objeto `node`

B. Mediante el objeto `ohai`

C. Mediante el objeto *facts*

D. Como a cualquier otro atributo, a través del objeto `node`, bajo la clave *facts*

10. ¿Cuál es el repositorio público de *cookbooks* compartidos de la comunidad de Chef?

A. Chef Knife.

B. Chef Kitchen.

C. Chef Supermarket.

D. ChefSpec.