Herramientas de Automatización de Despliegues

Tema 8. Roles de Ansible

Índice

Esquema

Ideas clave

8.1. Introducción y objetivos

8.2. Ansible Galaxy

8.3. Estructura de roles

8.4. Separación del playbook de WordPress

8.5. Dependencias de roles

8.6. Creación de roles para distintas plataformas

8.7. Trucos para escribir roles

8.8. Módulos en Ansible

8.9. Referencias bibliográficas

A fondo

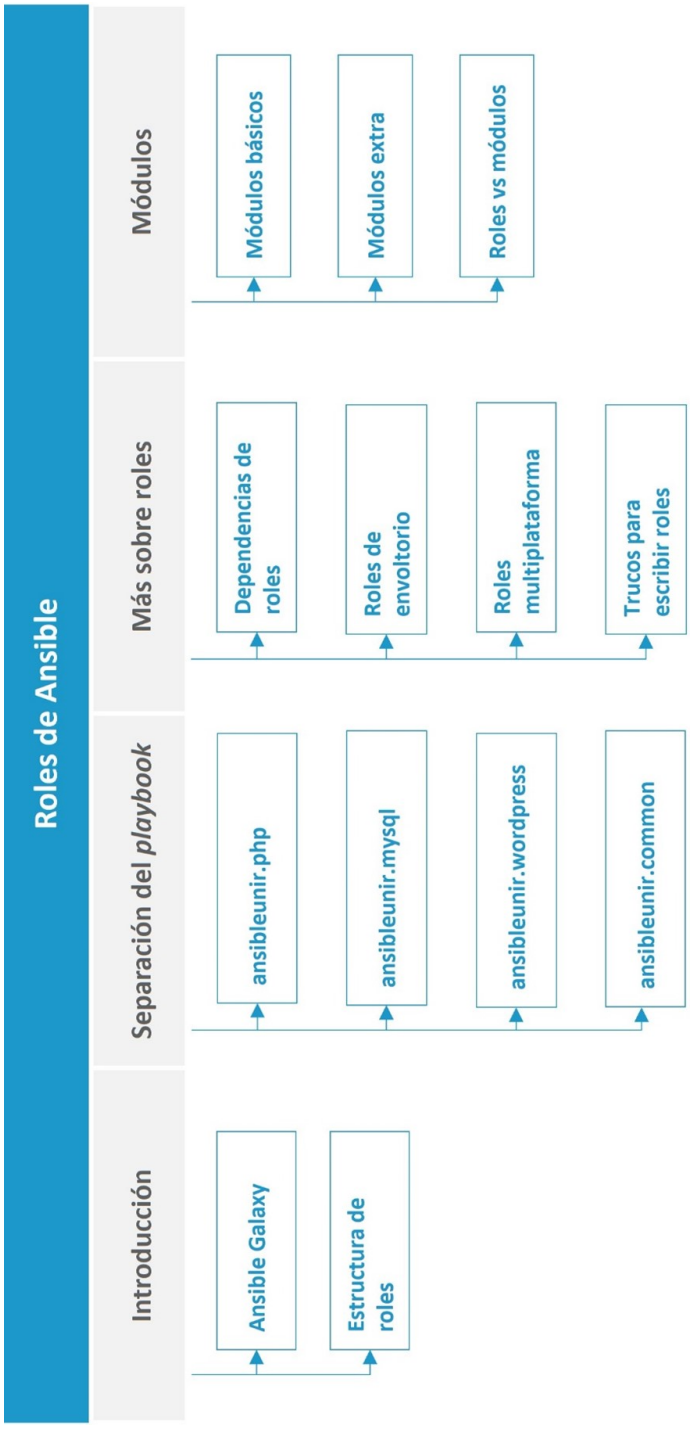
Documentación de referencia de Ansible – Roles

Ansible Galaxy

Creando roles con ansible

Test

Esquema



8.1. Introducción y objetivos

Los roles son el mecanismo proporcionado por Ansible que nos permite empaquetar las tareas, los manejadores y todos los demás archivos relacionados en componentes reutilizables que puedes compartir y reutilizar mediante su inclusión en un *playbook*.

Los *playbook*s y los roles son similares en cierto sentido, pero a la vez muy diferentes entre sí. Un *playbook* es un archivo independiente que Ansible puede ejecutar y contiene toda la información necesaria para establecer el estado de la máquina a lo que se desea. Esto quiere decir que un *playbook* puede incluir variables, tareas, manejadores, roles e incluso otros *playbooks*, todo en un mismo archivo.

Un rol podría considerarse como un *playbook* que se separa en diferentes archivos. En vez de tener un único archivo que contenga todo lo que necesitamos, vamos a tener un archivo para variables, otro para las tareas y otro para los manejadores. Sin embargo, no se puede ejecutar un rol por sí mismo; es necesario incluirlo dentro de un *playbook* junto con la información de los *hosts* donde ejecutarlo.

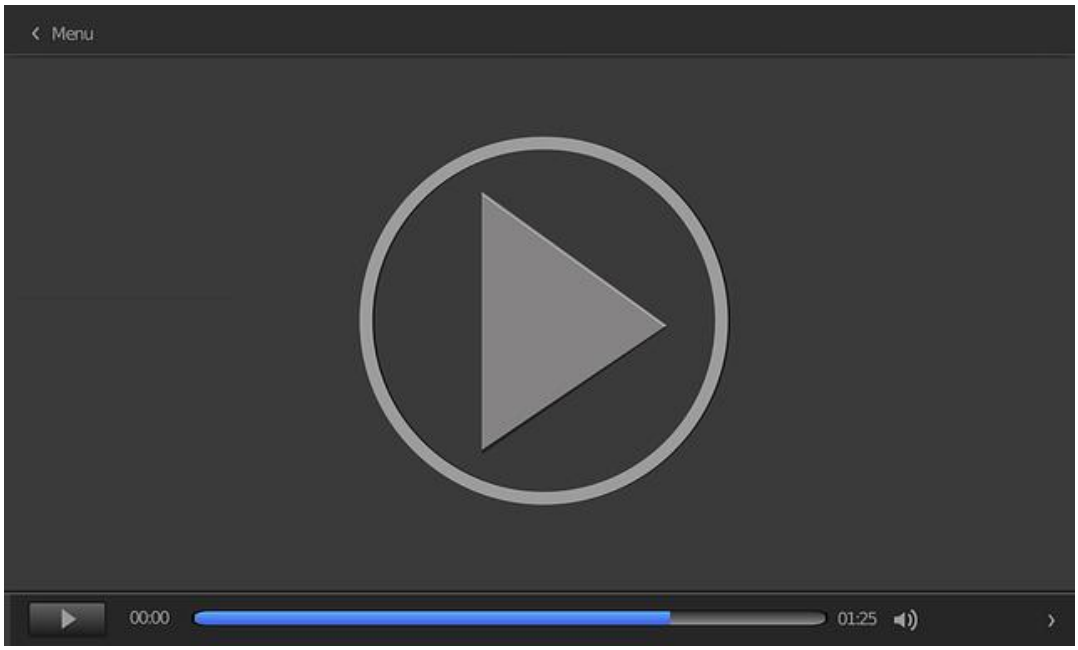
El objetivo que se pretende conseguir en este tema es el siguiente:

- Refactorizar el *playbook* del tema «Ansible: instalación de WordPress» para que se divida en diferentes roles:
  - Uno que instala PHP.
  - Otro que instala NginX.
  - Otro que instala MySQL.
  - Otro para WordPress.

Ideas clave

Esto no solo permitirá que continuemos trabajando con los *playbooks* y familiarizándonos con ellos, sino que también a su vez los convertirá en reutilizables. Tal como está nuestro ejemplo actualmente, si optásemos por instalar Drupal en lugar de WordPress, tendríamos que duplicar la configuración de toda dependencia y cambiar el final del *playbook*. Una vez que construyamos los roles a lo largo de este tema, podremos reutilizarlos en múltiples proyectos siempre que necesiten instalar un NginX, PHP o MySQL.

A continuación, puedes ver al vídeo *Sistemas de gestión de la configuración*:



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=9ee41e58-ecc0-4710-81d7-abb600fed6aa

8.2. Ansible Galaxy

Los **roles** son uno de los conceptos principales de Ansible. Tienen tanta importancia que incluso cuentan con su propio repositorio centralizado y una herramienta de línea de comandos para manejarlos. Ansible Galaxy es el repositorio web donde los usuarios de Ansible pueden subir roles que ellos mismos han desarrollado para que otros usuarios a su vez puedan descargárselos y utilizarlos.

Para acceder a este repositorio y examinar los roles que contiene, puedes acceder a https://galaxy.ansible.com donde hay más de 20 000 roles disponibles. ¡Consúltalos y utilízalos en tus proyectos propios!

Como puede ocurrir con cualquier repositorio público, podemos encontrar contribuciones tanto buenas como malas. Al realizar una búsqueda puedes ordenar los roles del resultado según el número de descargas, ya que cuanto mayor sea este número, más probable es que el rol sea de calidad. Cada rol tendrá un enlace a su código fuente, por lo que una vez que hayas profundizado lo suficiente en Ansible, deberías ser capaz de entender lo que está haciendo cualquier rol.

Al descargar un rol, puedes optar por instalarlo en tu máquina globalmente o solo localmente para un proyecto. Lo mismo que con cualquier otra dependencia, es recomendable que sea local a tu proyecto por si se da la circunstancia de que proyectos diferentes quieran usar la misma dependencia, pero con diferentes versiones. Para descargar un rol del repositorio, utiliza el comando proporcionando el parámetro para que el rol se instale en un directorio denominado roles. Debes ejecutar desde el mismo directorio donde se encuentra el archivo .

En Ansible Galaxy puedes encontrar algunos creadores de roles muy prolíficos, aunque uno que destaca significativamente es Jeff Geerling (*geerlingguy*). Sus roles son siempre de una gran calidad. Vamos a usar uno de los roles de Jeff como

ejemplo para mostrar cómo se debería descargar:

```
ansible-galaxy install geerlingguy.git –p roles
```

Esta línea creará la carpeta **roles** en caso de que no exista y descargará el rol en ella. Para usar este rol, debes incluirlo desde un *playbook*. Como ya sabemos, primero le indicamos a Ansible en qué servidores se va a ejecutar, y ahora, a continuación, también le proporcionaremos una lista de roles para que Ansible los incluya:

```
---

- hosts: all

 roles:

 - geerlingguy.git
```

Si ejecutas ahora este *playbook*, el rol va a intentar instalar en la máquina. Para probarlo, puedes incluir la sección roles en el *playbook* justo antes de la lista de tareas y a continuación ejecutar . En un *playbook* los roles se van a ejecutar antes que las tareas, por lo que la salida del rol se encontrará en la parte superior de la salida del comando .

Debería ser similar a la siguiente imagen:

Figura 1. Ejemplo de la salida de la ejecución de un rol. Fuente: elaboración propia.

El rol de Jeff se encarga de instalar tanto en RedHat como en los sistemas operativos basados en Debian. Debido a que esta máquina virtual ejecuta Ubuntu (que es derivado de Debian), se excluye la tarea RedHat y se ejecuta la tarea Debian para instalar los paquetes correspondientes.

8.3. Estructura de roles

Ahora que has aprendido cómo ejecutar un rol desde un *playbook*, vamos a crear un rol propio. Mediante la herramienta de línea de comandos, puedes también crear un nuevo rol, y lo vamos a hacer en la carpeta de roles. Al crear roles, existe una convención de nombres que debes seguir. Los nombres de los roles suelen tener el formato ; tal como . Simplemente con el nombre de rol, ya se sabe que es un rol de Jeff Geerling que instala .

Para el proyecto de ejemplo, vamos a utilizar ansibleunir como identificador, lo que significa que un rol que instala PHP se llamará . Vamos a crear ese rol; para ello, ejecuta los siguientes comandos en la misma carpeta que el fichero :

```
mkdir -p provisioning/roles
```

```
cd provisioning/roles
```

```
ansible-galaxy init ansibleunir.php
```

El comando habrá creado un subdirectorio llamado bajo el directorio roles que contendrá la estructura y los ficheros básicos de un rol de Ansible:

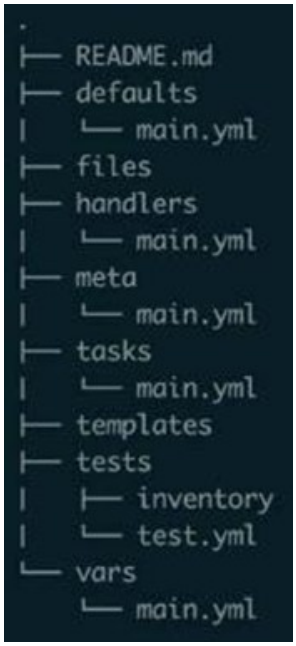


Figura 2. Estructura de directorios y ficheros de un rol. Fuente: elaboración propia.

La mayor parte de los directorios que se crean son opcionales, pero vamos a explicar para qué se utiliza cada uno y qué funcionalidad proporciona al añadir contenido en los ficheros que incorpora.

- Cada rol debe contar con un **fichero README** que describe el propósito del rol, su funcionalidad y las variables que se podrán utilizar y personalizar en el rol.

- El archivo puede contener los valores por defecto que utilizarán las variables que maneja el rol, en caso de no haber proporcionado otro valor personalizado. También es posible definir variables en otras ubicaciones tales como que actualizará cualquier variable que ya estuviera definida en , ya que tiene mayor prioridad.

- El directorio files es en el que se colocan los ficheros necesarios para la ejecución del rol. Podrá contener tanto elementos estáticos, ficheros de configuración, como cualquier otro tipo de fichero. Ten en cuenta que estos archivos no pueden ser

manipulados de ninguna manera, solo se pueden copiar.

- es donde defines manejadores como . El contar con todos los manejadores disponibles en un único lugar común es muy útil para cualquiera que utilice el módulo, para poder ver de un vistazo las acciones que están disponibles. Los manejadores se pueden invocar tanto desde el mismo rol, como desde otros roles y desde el *playbook* que incluye al rol.

- El archivo contiene los metadatos del rol. Este archivo va a incorporar los metadatos que utiliza Ansible Galaxy si publicas el rol. También se pueden definir algunos parámetros, como la versión mínima de Ansible requerida, plataformas soportadas y cualquier otra dependencia de este rol.

- El archivo es el punto de entrada del rol. Incorpora la sección de tareas que está contenida en tu *playbook*. Las acciones que están definidas en este archivo son las que procesará Ansible cuando se ejecute el rol.

- El directorio contiene todas las plantillas que necesitan ser procesadas por el procesador de plantillas para así sustituir las variables necesarias en el archivo (y cualquier otro procesamiento soportado por Jinja) antes de copiarlo en el sistema de destino.

- El directorio es donde debes incluir los *playbooks* de prueba que usan el rol. Esto se utiliza para definir pruebas automatizadas del rol, que podrán ejecutarse mediante un sistema de integración continua, tal como Jenkins o Travis CI. Este tipo de sistemas (de integración continua) son herramientas que detectan cuándo realizas modificaciones en el código fuente de un programa y desencadenan acciones relacionadas; típicamente, la compilación o construcción de ese código fuente, la ejecución de pruebas para el proyecto, empaquetado de los binarios o cualquier otra cosa que se pueda expresar en un *script*.

De todos estos directorios que hemos explicado, no todos son requeridos. El rol puede ser de gran utilidad, aunque solo incorpore un fichero . La mayor parte del

tiempo trabajaremos en este directorio tasks, con algunos ficheros de apoyo en los directorios y .

Cabe señalar que cada carpeta contiene un archivo dentro que se llama main.yml. Este es el nombre de archivo por defecto que carga Ansible cuando importa un rol. Esto es, para cargar las tareas de nuestro rol de PHP, Ansible buscará el archivo de tareas a ejecutar en la ruta . Se puede trabajar directamente en este archivo o se pueden crear archivos diferentes en el mismo directorio e incluirlos desde el fichero . La siguiente imagen muestra un ejemplo de un árbol de ficheros donde y están en el mismo directorio que .

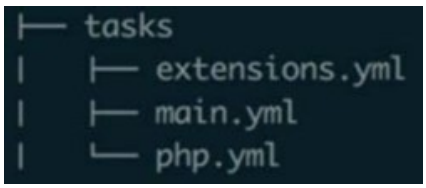


Figura 3. Directorio *tasks* de un rol con varios archivos de tareas. Fuente: elaboración propia.

Ahora cuentas con una clara separación de las tareas que instalan los paquetes PHP principales y de las tareas que instalan las extensiones, pero necesitas indicarle a Ansible la existencia de estos archivos, y lo harás desde el archivo que quedará de la siguiente manera:

```
---
```

```
- include: 'php.yml'
```

```
- include: 'extensions.yml'
```

Esto utiliza la sintaxis de YAML, que incorpora un archivo YAML dentro de otro. Cuando Ansible se ejecuta, todos estos archivos se fusionarán, pero mientras estés desarrollando el *playbook*, conseguirás una clara separación de las responsabilidades.

Ideas clave

A partir de la versión 2.0 de Ansible se introdujeron los includes dinámicos, que no se fusionan en el preprocesador (justo antes de la ejecución), sino durante la ejecución misma. Esto genera ciertos cambios de comportamiento, ya que se desconoce lo que realmente se va a incluir y ejecutar hasta que efectivamente se incluye durante la ejecución.

Para evitar que los cambios de comportamiento afectaran a *playbooks* ya existentes, en Ansible 2.1 se establecieron una serie de reglas para los dinámicos, para no romper la compatibilidad hacia atrás, aparte de añadir la opción que permite indicar explícitamente cómo queremos que se comporte el . Si queremos asegurarnos de incluir un fichero dinámicamente, debemos incluir lo siguiente:

```
- include: 'php.yml'
```

```
 static: no
```

8.4. Separación del playbook de WordPress

Vamos ahora a separar el *playbook* monolítico de WordPress en varios roles únicos. Lo primero que vamos a hacer es crear los roles que se van a necesitar, para lo cual nos situaremos en el directorio **roles,** y a continuación ejecutaremos los comandos siguientes para generar todos los roles vacíos que necesitamos:

```
ansible-galaxy init ansibleunir.nginx
```

```
ansible-galaxy init ansibleunir.mysql
```

```
ansible-galaxy init ansibleunir.wordpress
```

Una vez creados estos roles, debemos actualizar el archivo para que los incluya. La lista de roles se añade antes de la sección de tareas, con lo que el inicio del se parecerá al siguiente (si habías incluido el rol anteriormente, es el momento de suprimirlo):

```
---
- hosts: all
 become: true
 roles:
 - ansibleunir.php
 - ansibleunir.nginx
 - ansibleunir.mysql
 - ansibleunir.wordpress
```

```
tasks:
```

```
- name: Install required tools
```

```
[...]
```

Los nuevos roles están todavía vacíos, por lo que, si ejecutamos ahora Ansible, no se realizará ninguna acción adicional relacionada con ellos, ni tampoco fallará la ejecución. Los roles vacíos se pueden incluir sin peligro en *playbooks*, ya que Ansible buscará las tareas definidas en ellos, y no hará nada al no encontrar ninguna definida. Se han incluido ya los roles vacíos para que cuando comencemos en la siguiente sección a mover las tareas desde el *playbook* a su rol correspondiente, Ansible las incluya automáticamente en la lista de tareas que ejecutar de forma transparente.

Antes de continuar, deberías asegurarte de que el *playbook* está formateado y funciona correctamente, ejecutando . La ejecución debería terminar con éxito, ya que no suprimimos ninguna tarea. A continuación, vamos a ir moviendo las tareas de a los diferentes roles que hemos creado.

ansibleunir.php

Vamos a empezar rellenando el rol . Las siguientes dos tareas que aparecen en el *playbook* están relacionadas con la instalación de PHP, por lo que las vamos a mover a , que contendrá entonces lo siguiente:

```
---
```

```
- name: Install PHP
```

```
apt:
```

```
state: present
```

```
update_cache: yes

name:

- php

- php-fpm

- php-mysql

- php-xml

- name: Remove apache2

apt: name=apache2 state=absent
```

Al haber movido estas tareas, no deben ya existir en el *playbook*. Se incorpora en su lugar el rol bajo la sección de roles para que sea incluido en la lista de tareas que debe ejecutar Ansible. Una vez guardados tanto el rol como el *playbook*, puedes volver a ejecutar . Estas tareas se ejecutarán, al igual que lo hacían cuando estaban en el archivo . Sin embargo, ahora se puede ver que las tareas PHP tienen una cabecera diferente:

```
TASK [ansibleunir.php: Install PHP]
*******************************

ok: [default]

TASK [ansibleunir.php: Remove apache2]
*****************************

ok: [default]
```

Como se puede ver, ahora el nombre de la tarea es precedido por el nombre del rol,

como en el caso de . Esto hace que sea fácil identificar dónde están siendo incluidas las tareas cuando se ejecuta Ansible. Separando la parte de PHP de la instalación en un rol, hemos conseguido que la instalación de PHP sea reutilizable. Si en el futuro necesitásemos PHP en una máquina en algún otro *playbook*, podríamos añadir a la lista de roles a ejecutar y esto instalará todos los paquetes relevantes.

Seguiremos realizando la misma operación de refactorización para cada tarea en el *playbook* hasta que la sección tasks se quede solo con dos tareas: una que haga ping a tu máquina y otra que instale herramientas comunes.

ansibleunir.mysql

El siguiente grupo de tareas que hay que mover son del rol ansibleunir.mysql. Tienes por tanto que abrir el archivo r y mover todas las tareas relacionadas con MySQL del fichero playbook.yml al fichero de tareas del rol. Se trata de todas las tareas comprendidas entre las llamadas y , ambas inclusive.

Recordemos en este punto que las tareas MySQL usaban una plantilla para rellenar , que tendremos asimismo que mover para que también forme parte del nuevo rol. Debes por tanto mover el archivo que se encuentra en la ruta a . Esto puede hacerse mediante:

```
Mv provisioning/templates/mysql/my.cnf
```

```
provisioning/roles/ansibleunir.mysql/templates
```

Para finalizar la operación, tienes que hacer un cambio en la tarea de plantilla del rol . El parámetro de la tarea de plantilla es actualmente , pero dado que ahora la plantilla forma parte de un rol, esta ruta no es válida. Ansible automáticamente buscará en el directorio templates cuando se utiliza el módulo template dentro de un rol, por tanto, simplemente tenemos que cambiar el parámetro del módulo de plantilla al valor Una vez que hemos hecho este cambio, la tarea resultante en debería quedar como sigue:

```
- name: Create my.cnf
```

```
 template: src=my.cnf dest=/root/.my.cnf
```

```
 when: mysql_new_root_pass.changed
```

Ahora puedes ejecutar de nuevo para asegurar que el nuevo rol funciona adecuadamente. Todas las tareas de MySQL deberían estar prefijadas ahora con en la salida.

Ya estamos a mitad de camino de la refactorización tras haber migrado PHP y MySQL, y ya hemos visto el patrón de cambios que se deben hacer para mover tareas de un *playbook* a un rol independiente. La migración de los roles restantes que instalan NginX y WordPress siguen los mismos pasos, tal como veremos a continuación.

ansibleunir.nginx

Solo tenemos tres tareas para NginX: la de instalación de paquetes, la encargada de asegurar que NginX se ejecuta y la que copia la plantilla de configuración. Mueve ahora estas tareas de a . Una de estas tareas utiliza el módulo de plantilla (*template*). Como hicimos con el rol MySQL, tendremos que mover esta plantilla de modo que se aloje dentro del nuevo rol ejecutando:

```
mv provisioning/templates/nginx/default
```

```
provisioning/roles/ansibleunir.nginx/templates
```

Ahora corrige la tarea de modo que el campo no empiece por la ruta del directorio. Es decir, debería quedar solo el default:

```
- name: Create nginx config
```

```
 template: src=default dest=/etc/nginx/sites-available/default
```

```
notify: restart nginx
```

Esta tarea tenía también un manejador ( ) asociado, lo cual no habíamos encontrado todavía trabajando con roles. Así como se pueden definir tareas en un rol, pueden también definirse manejadores en el rol editando el fichero . Abre ahora y mueve a este archivo el manejador que se encuentra en . No se necesita la cabecera , solo la definición del manejador propiamente dicha. Una vez movido, se verá así:

```
---
```

```
# handlers file for ansibleunir.nginx
```

```
- name: restart nginx
```

```
 service: name=nginx state=restarted
```

Ya puedes eliminar la cabecera `del`, ya que debería estar ya vacía, y ejecuta nuevamente para probar el *playbook*. La ejecución debería completarse sin errores, quedando únicamente por migrar las tareas de WordPress.

ansibleunir.wordpress

Este es el rol más complejo al contar con diez tareas diferentes, pero no es complicado de migrar. Se puede utilizar la misma estrategia que con los tres roles anteriores, migrándolo paso a paso hasta que se haya completado. Las tareas que debes mover van desde la que se llama , hasta la de Import WordPress DB. Mueve todas estas tareas, incluyendo estas dos de ambos extremos, del fichero a .

Solo tenemos una tarea que usa el módulo de plantilla en este grupo, así que vamos a migrarla en primer lugar. Esto ya debería resultarte familiar. Primero tienes que mover el archivo para alojarlo dentro de su rol. Para ello, ejecuta:

```
mv provisioning/templates/wordpress/wp-config.php
```

```
provisioning/roles/ansibleunir.wordpress/templates
```

Posteriormente, debes actualizar la tarea de plantilla; ahora el parámetro solo debe contener , sin referencia al directorio:

```
- name: Create wp-config
```

```
template: src=wp-config.php dest=/var/www/book.example.com/wp-
```

```
config.php
```

Este rol no utiliza ningún manejador, por lo que no hay nada que migrar a este respecto. Sin embargo, hay otra cosa que tenemos que migrar. Tal como hemos hecho al mover los ficheros del módulo de plantilla, también tendremos que hacer lo propio con los archivos del módulo de copia. Los archivos utilizados por el módulo de plantilla se ubican en el directorio , y los archivos para el módulo de copia están situados en el directorio . Hay dos tareas en que utilizan el módulo : una para copiar el archivo ZIP de WordPress y otra para copiar la copia de seguridad de la base de datos en el sistema. Lo primero es mover estos archivos para situarlos dentro de la

ubicación correspondiente en el rol:

```
mv provisioning/files/wordpress.zip
 provisioning/roles/ansibleunir.wordpress/files
mv provisioning/files/wp-database.sql
 provisioning/roles/ansibleunir.wordpress/files
```

También necesitas actualizar cada tarea copy para eliminar la referencia al directorio del parámetro . Al igual que ocurre con las plantillas, Ansible busca los ficheros del módulo de copia directamente en su directorio correspondiente.

Una vez que hayas finalizado con estas operaciones, las tareas serán como se muestra a continuación:

```
- name: Copy wordpress.zip into tmp
 copy: src=wordpress.zip dest=/tmp/wordpress.zip
- name: Copy WordPress DB
 copy: src=wp-database.sql dest=/tmp/wp-database.sql
 when: db_exist.rc> 0
ansibleunir.common
```

Llegados hasta aquí, podemos eliminar la tarea de ping, ya que únicamente servía para comprobar que Ansible lograba conectarse al *host*. En cuanto a la tarea que instala paquetes comunes, pueden guardarse tales tareas en otro rol. Vamos a crear este nuevo rol ahora ejecutando el comando desde el directorio roles. Mueve a del nuevo rol la tarea que instala los paquetes comunes, y añade como primer rol de la

lista de roles en . Dado que ahora la sección de tareas estará vacía, se puede suprimir perfectamente. ¡El *playbook* es mucho más pequeño ahora!

```
---

- hosts: all

 become: true

 roles:

 - ansibleunir.common

 - ansibleunir.php

 - ansibleunir.nginx

 - ansibleunir.mysql

 - ansibleunir.wordpress
```

Asegúrate de ejecutar una vez más para asegurarte de que todo sigue funcionando. Una vez hecho esto, ¡enhorabuena! ya has terminado de refactorizar el *playbook* en cinco roles reutilizables.

A partir de ahora, si necesitas PHP en algún otro proyecto, simplemente tienes que incluir en tu *playbook*. Si necesitas una base de datos, puedes incluir . Los roles son un mecanismo muy potente para encapsular tu lógica y, al mismo tiempo, acceder fácilmente si lo necesitas.

Como hemos visto, la creación de un rol de un *playbook* no es un procedimiento complicado. En esencia, es completamente mecánico. Solo has de seguir estos pasos:

Ideas clave

- Mueve las tareas al archivo .
- Mueve los manejadores al archivo .
- Si alguna tarea utiliza el módulo de plantilla, mueve los archivos necesarios al directorio de plantillas y cambia el atributo del módulo para que sea relativo al directorio templates; por ejemplo, buscaría la plantilla en:
- Si alguna tarea utiliza el módulo de copia, mueve los archivos necesarios al directorio y cambia el atributo del módulo para que sea ahora relativo al directorio files; por ejemplo, buscaría el archivo en: .
- Mueve cualquier variable utilizada en el rol (en tareas o en plantillas) a (esto no fue necesario en nuestro caso).

8.5. Dependencias de roles

Tal como ha quedado, el *playbook* ejecuta e instala todas las dependencias necesarias para configurar la aplicación WordPress. Esto es debido a que hemos especificado todos los roles que instalan los prerrequisitos en la lista de roles que deben ejecutarse. Esto es adecuado solo si somos nosotros mismos los que lo utilizamos, pero si lo utiliza alguien que no conoce las dependencias necesarias, puede omitir alguno de los roles necesarios cuando trate de usarlo.

Para evitar esto, existe la opción de dependencias en que fue mencionada al explicar la estructura de directorios de un rol. Pues bien, puedes usarla para especificar las dependencias de un rol y hacer que Ansible las incluya de forma automática, en lugar de tenerlo que incluir tú en el *playbook*. Abre el archivo y borra todo el contenido (toda la información que contiene es opcional, así que lo podemos suprimir sin problemas) para introducir solo la información que necesitamos. Añade el siguiente contenido:

```
dependencies:
```

```
- ansibleunir.common
```

```
- ansibleunir.php
```

```
- ansibleunir.mysql
```

```
- ansibleunir.nginx
```

Esto es la lista de roles requeridos para ejecutar este rol. Después, modifica para que solo esté en la lista de roles. Si ejecutas nuevamente , podrás apreciar que todas las dependencias se ejecutan antes de que se ejecute el rol . Ansible analiza los metadatos de cada rol que encuentra y se asegura de que cualquier dependencia

puesta en la lista se ejecute antes que el rol. Esto permite a quien lo use no tener que saber cuáles son las dependencias de nuestro rol; basta con que lo incluya en la lista de roles que desea ejecutar y Ansible ejecutará las dependencias de forma recursiva automáticamente antes que el propio rol que se incluyó.

Roles de envoltorio

Cuando un rol se implementa para poder ser utilizado en diferentes situaciones, tiende a ser muy configurable. Pero, en ocasiones, las posibilidades de configuración de un rol hacen que sea más difícil de gestionar y utilizar que lo que cabría esperar. **Un rol de envoltorio** es un patrón que puede usarse para abstraer parte de esta configuración.

Al envolver un rol dentro de otro rol, se puede entender la intención de cómo un rol debería ser utilizado desde otro rol. Imagina que tienes un rol para saludar. Este rol se llama y contiene los siguientes dos archivos:

```
# defaults/main.yml

---

your_name: World

# tasks/main.yml:

---

- name: Say hello

 debug: msg="Hello {{your_name}}"
```

Al ejecutar esto, se obtendrá la siguiente salida:

```
TASK [ansibleunir.hello: Say hello]
*********************************
```

```
ok: [default] => {
"msg":"Hello World"
}
```

Esto es debido a que tiene un valor por defecto Mundo para la variable . Si quisieras cambiar el nombre utilizado, podrías poner una variable que lo sobrescribiera en tu *playbook*. Sin embargo, si siempre quisieras que dijera «Hola Unir», por ejemplo, el tener que proporcionar el nombre en cada *playbook* que incluyera este sería algo engorroso.

La alternativa más adecuada es la de envolver este rol en otro que contenga únicamente las variables que quieres poner. Para hacer esto, crea un rol llamado que no contenga ninguna tarea, sino que solo especifique como una dependencia:

```
# meta/main.yml
dependencies:
- role: ansibleunir.hola
 vars:
 your_name: Unir
```

Si ejecutas ahora :

```
TASK [ansibleunir.hello: Say hello]
**********************************
ok: [default] => {
```

```
"msg":"Hello Unir"
```

```
}
```

Esta es una manera muy conveniente de utilizar una configuración personalizada para un rol y codificarlo de forma que pueda guardarse en el control de versiones y usarse tantas veces quieras. Hemos visto este patrón mediante un ejemplo simple, pero se puede extrapolar a un rol MySQL, por ejemplo, para especificar los nombres de usuario y contraseñas de la base de datos que utilizamos habitualmente. Cuando necesites una base de datos específica en una máquina, solo tendrías que incluir el rol .

8.6. Creación de roles para distintas plataformas

No todas las plataformas disponen de los mismos comandos y utilidades. Utilizar el módulo para instalar cualquier paquete ha hecho nuestro trabajo más fácil, pero si tratamos de usar nuestro rol en otra plataforma, esto no funcionará. Siempre que sea posible, es recomendable evitar la mezcla de distintas plataformas en un mismo despliegue.

En esta situación hay que hacer algo más de trabajo. Apache2 es un ejemplo muy adecuado que podremos usar. Mientras que el paquete para Apache se llama en sistemas basados en Debian, su nombre es en sistemas basados en Redhat. En tal situación, se pueden crear tres archivos de tareas diferentes: , , e . El fichero será el encargado de incluir el fichero de variables adecuado y luego delegar la instalación en el *script* que corresponda.

```
# main.yml
```

```
---
```

```
- include_vars:"{{ansible_os_family}}.yml"
```

```
- include: install-debian.yml
```

```
 when: ansible_os_family =='Debian'
```

```
- include: install-redhat.yml
```

```
 when: ansible_os_family =='RedHat'
```

En vez de condicionar las tareas con , se podía haber simplificado:

```
include:"install-{{ansible_os_family}}.yml"
```

Sin embargo, es preferible usar en vez de incluir el fichero basado en la variable para así ver claramente qué familias de sistemas operativos están soportados por el rol. También se incluye el fichero de variables correcto automáticamente. El fichero de variables define las mismas variables, pero asigna en cada caso diferentes valores.

```
# vars/Debian.yml

---

apache2_package_name: apache2

# vars/RedHat.yml

---

apache2_package_name: httpd
```

Posteriormente, en el *script* de instalación, se utiliza el módulo Ansible para el gestor de paquetes correspondiente según sea la familia de SO.

```
# tasks/install-debian.yml

---

- name: Install Apache

 apt: name={{apache2_package_name}} state=present

# tasks/install-redhat.yml

---

- name: Install Apache
```

```
yum: name={{apache2_package_name}} state=present
```

Este patrón es muy común, y es la forma aceptada generalmente para desarrollar un rol que funcione en múltiples familias de sistemas operativos.

Cabe también señalar que para estos casos existe en Ansible un módulo (https://docs.ansible.com/ansible/latest/collections/ansible/builtin/package_module.html) que delega al gestor de paquetes correspondiente para cada sistema operativo. No se ha querido utilizar este módulo en el ejemplo para poder demostrar cómo utilizar módulos distintos según la familia del sistema operativo. Si lo escribieras tú mismo, lo deberías escribir como aparece debajo, y Ansible delegaría la labor al gestor de paquetes correspondiente para la familia de sistema operativo sobre el que se está ejecutando:

```
---
```

```
- name: Install Apache
```

```
package: name={{apache2_package_name}} state=present
```

8.7. Trucos para escribir roles

Según Heap (2016), estas son las recomendaciones que hay que tener en cuenta a la hora de escribir roles:

- Cuando crees un rol, trata de asegurarte de que es utilizable sin necesidad de aportar nada más. Si tu rol instala un paquete específico de *software*, haz que se instale y configure una instalación básica sin intervención de usuario. Proporciona puntos de extensión para los usuarios que quieran personalizar cosas más tarde, pero no les hagas proporcionar información de entrada. Utiliza para esto, ya que puede ser fácilmente sobrescrito.

- Escribe un rol sencillo que haga exactamente lo que debe hacer (¡y nada más!). Nos puede ocurrir que tratemos de hacer un rol tan genérico que lo compliquemos en exceso. La mayor ventaja que conseguirás de los roles es hacer justamente lo que hemos hecho en este tema: extraer la funcionalidad común en roles que pueden ser incluidos en otros *playbooks*.

- Hay dos clases de roles: rígidos y flexibles. Generalmente encontrarás roles flexibles en Ansible Galaxy, que han sido diseñados para ser reutilizados. Soportarán muchas variables diferentes y te dejarán usarlos como necesites.

- Cuando escribas roles tú mismo, o los encuentres en un escenario concreto (de un cliente, por ejemplo), verás que son completamente rígidos. Estos roles son específicos para un cliente, una aplicación particular, o un caso de uso de una aplicación. No necesitan puntos de extensión, por lo que, en vez de usar variables, se tiende a establecer los valores directamente en el rol. Se busca principalmente codificar los requisitos, y no tanto el hacer un rol reutilizable.

8.8. Módulos en Ansible

Los módulos, también denominados *plugins* de tareas (o *plugins* de biblioteca), son fragmentos de código que se pueden usar desde la línea de comandos o desde una tarea de un *playbook*. Ansible ejecuta cada módulo habitualmente en el nodo remoto destino de la ejecución y recolecta los valores de retorno.

Existen módulos disponibles en Ansible para las tareas más comunes de administración del sistema. Hay actualmente más de 500 módulos que se suministran con Ansible, que es una cifra considerable si la comparamos con los 141 módulos que se suministraban con la versión de octubre de 2013.

No obstante, podemos escribir nuestro propio módulo (en o Python) y extender así la funcionalidad existente. Todos los módulos básicos de Ansible están desarrollados en Python. Hay dos grupos de módulos: y .

ansible-modules-core

Contiene todos los módulos básicos que se proporcionan con Ansible, tales como y , y es mantenido por el equipo principal de Ansible. Estos módulos son robustos y sufren importantes revisiones antes de incorporar cualquier nuevo cambio.

ansible-modules-extras

Los módulos restantes se almacenan en este grupo, y aunque se distribuyen en la instalación estándar de Ansible, en realidad son mantenidos por la comunidad. El repositorio de módulos extra contiene algunos como y . Al contribuir con Ansible, si su cambio afecta a un módulo extra, el responsable de ese módulo es el responsable de revisar sus cambios, y no el equipo de Ansible.

¿Cuándo implementar un módulo?

En ocasiones, no es una tarea fácil decidir si crear un rol o escribir un módulo. Si

necesitas realizar varios pasos relacionados pero que no requieren realizar peticiones complejas a servicios externos, entonces un rol es la decisión adecuada. Un buen ejemplo de esto sería el de crear una base de datos, configurar los usuarios e importar un archivo SQL de ejemplo.

Si necesitas interactuar con una fuente de datos externa, como una API, entonces un módulo es la mejor opción, ya que tienes a tu disposición toda la potencia de un lenguaje de programación, en lugar de utilizar el módulo de comandos y .

Los módulos suelen estar diseñados para un cometido más concreto, aceptando una serie de variables que pueden utilizarse para configurar su comportamiento y permitir flexibilidad y reutilización.

8.9. Referencias bibliográficas

Heap, M. (2016). *Ansible: from Beginner to Pro*. Apress.

Hochstein, L. y Moser, R. (2014). *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way.* O'Reilly Media.

Documentación de referencia de Ansible – Roles

Red Hat, Inc. (2020). *Roles*. Ansible Documentation. https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

Documentación de referencia más actualizada de Ansible relativa a roles. Esta sección de la documentación oficial te explica en detalle todo lo relativo a los roles.

Ansible Galaxy

Red Hat, Inc. (2018). *Ansible Galaxy*. https://galaxy.ansible.com/

Ansible Galaxy es el repositorio oficial de Ansible para compartir contenidos. Proporciona el mayor catálogo de roles Ansible disponibles para su consulta y descarga. Asimismo, cualquier rol reutilizable que se realice se puede publicar en Galaxy y dejarlo disponible para toda la comunidad Ansible.

Creando roles con ansible

Monells, G. (2016). *Creando roles con Ansible*. Linux Sysadmin. https://www.linuxsysadmin.ml/2016/08/creando-roles-con-ansible.html

Tutorial en castellano que explica la creación de roles de una manera muy didáctica mediante un ejemplo que crea varios roles distintos.

Test

1. ¿Cómo se llama el repositorio principal donde encontrar roles compartidos de Ansible?

A. Ansible Community.

B. Ansible Hub.

C. Ansible Galaxy.

D. Ansible Forge.

2. ¿Para qué sirve el fichero en un rol?

A. Son las tareas que se procesarán al ejecutar el rol.

B. Sirve para referenciar las tareas dependientes.

C. Si no existe otro fichero, será el fichero por defecto que se procese al ejecutar el rol.

D. Es el fichero principal de tareas, se ejecutará tras los ficheros secundarios de tareas.

3. ¿Qué pasa cuando se incluye de manera estática un fichero en un *playbook*?

A. Todo el contenido del fichero incluido se verá fusionado en tiempo de desarrollo.

B. El contenido del fichero incluido se fusionará justo antes de la ejecución.

C. Nada, es simplemente una referencia informativa que indica que se va a usar ese fichero posteriormente.

D. El fichero incluido se copia al *host* que se está gestionando.

4. ¿Cómo se puede referenciar un rol desde un *playbook*?

A. Incluyéndolo en la sección roles antes de *hosts*.

B. Incluyéndolo en la sección roles antes de *tasks*.

C. Incluyéndolo en la sección *tasks* como primera tarea.

D. Incluyéndolo en la sección roles después de *tasks*.

5. ¿Qué diferencia la invocación de una plantilla (*template*) desde un rol a hacerlo desde un *playbook* independiente?

A. No se diferencian en nada.

B. En un *playbook* independiente se referencia la ruta relativa al directorio de la plantilla, y en el rol es la ruta relativa a la raíz del rol.

C. En un *playbook* independiente se referencia la ruta absoluta o relativa de la plantilla, y en el rol es la ruta relativa al directorio del rol.

D. En un *playbook* independiente se referencia la ruta absoluta de la plantilla, y en el rol es la ruta relativa al directorio del rol.

6. ¿Dónde se deben ubicar los manejadores en un rol?

A. En el fichero

B. En el fichero

C. En el fichero

D. En el fichero

7. ¿Cómo se pueden declarar las dependencias en un rol?

A. En el fichero

B. En el fichero

C. En el fichero

D. En el fichero

8. ¿Para qué sirve un rol de envoltorio?

A. Para complicar el uso del rol que se envuelve.

B. Para simplificar el uso del rol que se envuelve.

C. Para proporcionar valores concretos a variables de configuración del rol que se envuelve.

D. Las opciones B y C son correctas.

9. ¿Cuál es la mejor manera de configurar un rol para permitir distintas familias de sistemas operativos (SO) Linux?

A. Crear distintos ficheros de variables para cada familia de SO, con los mismos nombres de variables y añadir tantos al fichero de tareas como ficheros de variables, condicionados a la variable .

B. Crear distintos ficheros de variables para cada familia de SO, con los mismos nombres de variables y añadir tantos al fichero de tareas como ficheros de variables, condicionados a la variable .

C. Crear un único fichero de variables, y añadir tantos ficheros de tareas como familias de SO, y Ansible ejecutará el que corresponda.

D. Crear un único fichero de variables, y añadir tantos al fichero de tareas como familias de SO.

10. ¿Qué son los módulos en Ansible?

A. Agrupación de tareas reutilizables.

B. Fragmentos de código que se usan en una tarea.

C. Una encapsulación de roles.

D. Una biblioteca de funciones matemáticas.