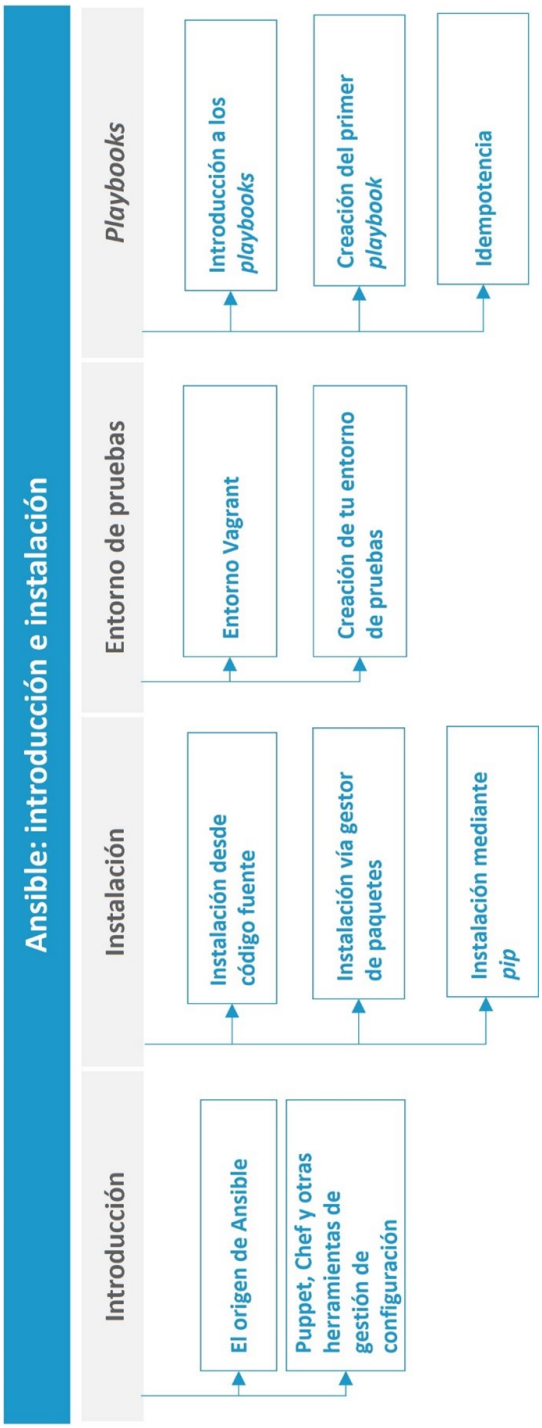
Herramientas de Automatización de Despliegues

Tema 5. Ansible. Introducción e instalación

Índice

Esquema



5.1. Introducción y objetivos

Hoy en día, cualquier negocio es un negocio digital. La tecnología es el motor de la innovación, y la entrega de las aplicaciones de una forma más rápida que la competencia ayuda a marcar la diferencia. Históricamente, se necesitaba una gran cantidad de esfuerzo manual y coordinación que complicaba el proceso. Pero, hoy en día, existen herramientas de gestión de la configuración tales como Ansible que son implementadas por miles de empresas a fin de acabar con la complejidad y acelerar sus entornos DevOps.

Los objetivos que se pretenden conseguir en este tema son los siguientes:

- Conocer la herramienta Ansible y sus principales características.
- Instalar un entorno Ansible sencillo.
- Introducir los conceptos básicos de *playbooks*.

5.2. El origen de Ansible

La primera versión de Ansible apareció en 2012 como un pequeño proyecto de apoyo de la mano de Michael DeHaan y ha tenido una ascensión meteórica en popularidad, con más de 40 000 estrellas y 5000 contribuidores únicos en GitHub.

Ansible es uno de los proyectos de código abierto con más popularidad que puedes encontrar en GitHub. No solo ha alcanzado un gran éxito, sino que gigantes como la NASA, Spotify o Apple lo han adoptado como herramienta de gestión de la configuración.

Ansible utiliza ficheros en formato **YAML,** que es un lenguaje de representación de datos utilizado habitualmente para configuración, como la fuente principal de información en tiempo de ejecución. Los ficheros YAML son muy simples de escribir y, si nunca has trabajado con ellos, una vez que hagas un par de ellos, ya estarás suficientemente familiarizado. Los ficheros YAML son muy fáciles de leer y usar, al contrario que lo que ocurre con otros formatos o lenguajes como JSON o XML, que no son tan amigables. Sin embargo, incluye algunas particularidades, siendo las principales las de ser sensible a los caracteres de tabulación (no le gustan), a los espacios en blanco y a la sangría de cada línea. Si alguna vez has escrito código fuente en lenguaje de programación Python, todo esto no será ningún problema.

Python es el lenguaje de programación en el que está escrito Ansible. El ejecutable principal y todos los módulos son compatibles con Python 2.7 o Python 3.5, lo que significa que funcionan con cualquier versión de Python2 por encima de la 2.7 o Python 3 por encima de la 3.5. DeHaan eligió Python para Ansible porque no requiere dependencias adicionales en las máquinas que se van a gestionar. En aquella época, la mayoría de las herramientas de gestión de la configuración existentes necesitaban de la instalación de Ruby como prerrequisito.

No solo no hay ninguna dependencia o requisito adicional de instalación de otros lenguajes para las máquinas que se van a gestionar, sino que no hay ningún otro requisito adicional. Ansible aprovecha el protocolo SSH para ejecutar sus comandos de forma remota en las máquinas que gestiona, por lo que no requiere la instalación de ningún otro protocolo o sistema de comunicación. Esto supone una gran ventaja, debido a:

- Las máquinas gestionadas van a ejecutar exclusivamente la aplicación o aplicaciones que le correspondan, sin ningún otro proceso ejecutándose en segundo plano que compita por la CPU y memoria de la máquina.

- Al hacer uso de SSH, toda su funcionalidad está disponible para aprovecharla. Puedes, por ejemplo, utilizar un *host* como máquina de salto para alcanzar otro *host*. Además, no existe la necesidad de incluir un mecanismo de autenticación propio; puedes utilizar el que te proporciona SSH.

Herramientas de gestión de la configuración

La primera herramienta que se considera pionera de la gestión de la configuración fue **CFEngine.** Sin embargo, los que mayor popularidad alcanzaron fueron Puppet y Chef, y por ello todavía hoy en día se comparan habitualmente con Ansible.

Según Heap (2016), Puppet y Chef son herramientas más similares entre sí de lo que lo son con Ansible, aunque todas ellas realizan funciones similares. Tanto Puppet como Chef se basan en el uso de un servidor centralizado para gestionar todo lo relativo al estado de la configuración requerido de los *hosts* y sus metadatos relacionados. Ansible, por el contrario, no necesita utilizar un servidor centralizado al que se conecten los agentes desplegados en las máquinas a gestionar, ya que no necesita el uso de agentes; es, por tanto, *agentless*. Esta es una característica muy importante, ya que al utilizar herramientas como Puppet y Chef, cada agente que ejecuta en una máquina gestionada se conectará periódicamente con el servidor centralizado para comprobar si existen cambios de la configuración, y los aplicarán

automáticamente. Ansible, en cambio, delega completamente en el usuario final la labor de propagar los cambios de configuración cuando se requiera.

Ansible es más parecido a **SaltStack** (Salt), otra herramienta que también está escrita en Python y utiliza ficheros YAML para la configuración. Tanto Ansible como Salt están diseñadas fundamentalmente como motores de ejecución, donde la definición de la configuración del sistema no es más que una lista de comandos que ejecutar, que se abstraen mediante módulos reutilizables, los cuales se encargan de proporcionar una interfaz idempotente a los servidores.

Una característica común de todas estas herramientas, tanto Ansible y Salt como Chef y Puppet, es que son declarativas, en lugar de imperativas. La gran mayoría de los elementos de configuración que proporcionan permiten que el usuario defina el estado de configuración deseado de sus *hosts* y sea la propia herramienta la encargada de alcanzarlo, mediante la aplicación de las acciones que considere necesarias. Para lograr esto es importante el concepto de idempotencia, que permite aplicar tantas veces como queramos una definición de configuración sobre un *host* y su estado resultante será el mismo en todas las ejecuciones. La idempotencia es una característica importante en este tipo de herramientas, y la explicaremos con detalle más adelante.

Como ya hemos mencionado más arriba, Ansible no requiere de ningún agente en las máquinas que gestiona. Esto es lo que se denomina **modelo *agentless***. En este modelo, es necesario enviar los cambios de la configuración a las máquinas cuando así se requiera, a demanda (cuando haya cambios en la configuración, o en las propias máquinas, generalmente). Este modelo es diferente del de Puppet y Chef (con agente), donde se usa un servidor centralizado que almacena la copia maestra de la configuración y al que las máquinas consultan periódicamente para asegurarse de que tienen el estado de su configuración actualizado.

Este modelo tiene ventajas e inconvenientes; la ventaja principal es que, una vez que

haces cambios, puedes enviarlos inmediatamente a las máquinas, sin esperar a que un proceso demonio (el agente) compruebe si hay cambios. La desventaja es que tú eres el responsable de distribuir esos cambios a tus máquinas, mientras que con Puppet y Chef basta simplemente con guardar *(commit)* tus cambios en el servidor centralizado, teniendo la certeza de que serán distribuidos pronto. Cabe destacar que Ansible puede configurarse para utilizar este modelo de funcionamiento *pull*, pero no es lo más habitual.

Otra característica de Ansible que ya habíamos mencionado es el uso del protocolo SSH para conectarse remotamente a los *hosts* que gestiona. Esto nos proporciona una gran confianza en el mecanismo de transporte, aunque, por otro lado, puede acabar resultando lento. Por el contrario, Salt utiliza ZeroMQ, que es muy rápido cuando se trata de iniciar una conexión y enviar comandos al destinatario.

5.3. Instalación de Ansible

La instalación de Ansible puede ser una tarea sencilla, si utilizamos las versiones que suelen estar disponibles en el repositorio del gestor de paquetes del sistema operativo correspondiente, tales como en Debian y derivados o en Fedora. Sin embargo, la versión disponible en estos repositorios no suele ser la más reciente, por lo que, si deseamos estar a la última, tendremos que optar por otro método de instalación.

Dado que Ansible tiene un desarrollo muy acelerado, puedes querer utilizar la versión más actualizada posible, ya sea construyéndola tú mismo o instalándola mediante un gestor de paquetes.

Para garantizar el uso de la versión más actualizada, puedes optar por descargar directamente de GitHub el código fuente de la herramienta desde el repositorio ubicado en https://github.com/ansible/ansible e instalarlo tú mismo.

Puedes necesitar instalar algunas dependencias, si tu sistema no las tiene ya, tales como Git o la versión adecuada de Python y, una vez que las tengas, puedes descargar Ansible directamente desde el repositorio GitHub y construirlo:

```
git clone git://github.com/ansible/ansible.git –recursive
```

```
cd ansible
```

```
make
```

```
sudo make install
```

Con esto construirás e instalarás la última versión disponible de desarrollo de Ansible desde el propio código fuente.

Ideas clave

Si estás ejecutando desde una máquina con un sistema operativo basado en Debian o RedHat, puedes preferir utilizar o para instalarlo a través del gestor de paquetes de tu sistema, dado que proporciona una versión más estable y probada, así como una manera limpia de desinstalar Ansible.

Si estás ejecutando en un sistema que no tiene Ansible disponible a través de su gestor de paquetes, puedes instalarlo a través del Makefile. Para ello, debes ejecutar como se ha mostrado anteriormente, con lo que lograrás instalar Ansible en tu sistema.

Si tienes una máquina basada en OS X, puedes instalar Ansible a través del gestor de paquetes Homebrew (https://brew.sh/). Bastará con ejecutar y se instalará la última versión que se haya registrado en el gestor.

Otra manera de instalarlo sería utilizando PPA, para lo que necesitarás primero registrar el repositorio y actualizar la caché del gestor de paquetes. Necesitarás instalar previamente el paquete si no tienes ya instalado en tu sistema :

```
sudo apt-get install python-software-properties # if required
```

```
sudo apt-add-repository ppa:ansible/ansible
```

```
sudo apt-get update
```

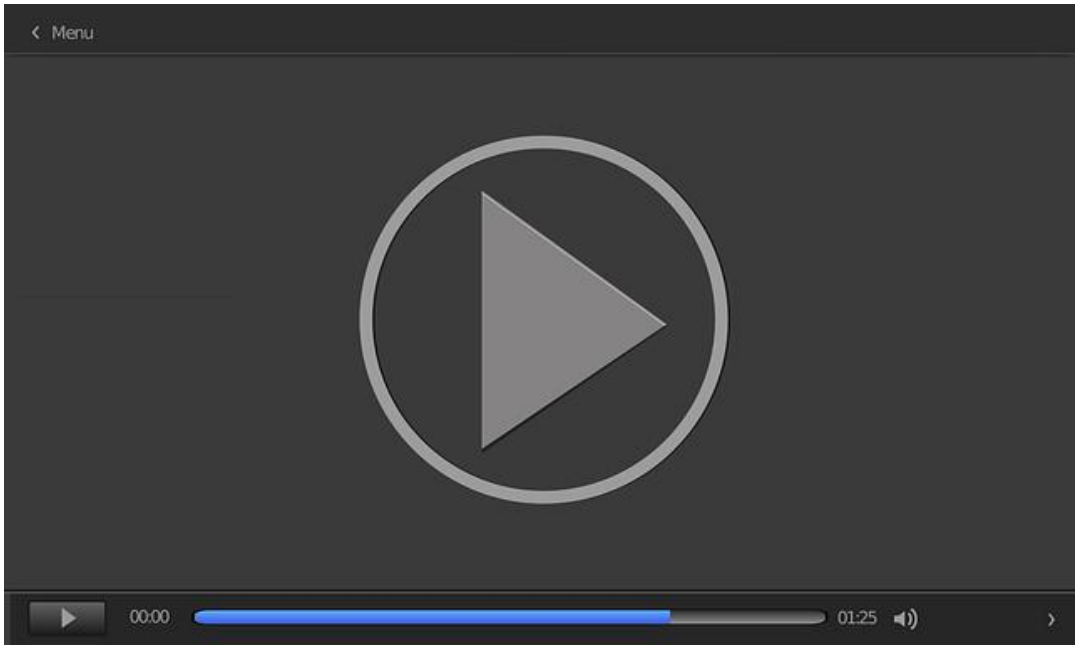
```
sudo apt-get install ansible
```

También puedes optar por la instalación utilizando , el gestor de paquetes de Python, que debes tener instalado previamente en tu sistema. Este método puede ser la alternativa si tu sistema operativo no soporta el uso de PPA:

```
sudo pip install ansible
```

Ideas clave

En el siguiente vídeo puedes ver la explicación detallada de la instalación de Ansible:



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=888c0a38-f91a-477c-83b9-abb6009c536c

5.4. Creación de un entorno de pruebas

Una vez instalado Ansible, puedes comenzar a automatizar tu infraestructura, para lo que necesitarás un entorno de pruebas y un *playbook* de Ansible. Una manera sencilla y rápida de poder desplegar un entorno de pruebas de desarrollo temporal es mediante las herramientas Vagrant y VirtualBox. Con estas herramientas generaremos el entorno donde instalar el conjunto *(stack)* PHP y MySQL mediante nuestro primer *playbook*.

Creación de un entorno de pruebas Vagrant

Antes de desarrollar el primer *playbook*, vamos a necesitar un entorno donde probarlo. Se puede desarrollar *playbooks* y ejecutarlos directamente en tu máquina local, pero es más recomendable poder probarlos en un entorno aparte, donde no tenga importancia si se han producido errores o que la configuración no funcione tal como se esperaba. Una máquina virtual (VM) de VirtualBox (que proporciona la virtualización) y la herramienta Vagrant (que automatiza la gestión de esas máquinas) proporcionan los entornos de prueba.

La herramienta VirtualBox de Oracle es gratuita y proporciona un motor de virtualización. Nos permite crear una máquina virtual e instalar en ella cualquier sistema operativo de la misma manera que se haría en una máquina física. Se puede usar VirtualBox como *software* independiente, pero es recomendable combinarlo con Vagrant, que es fundamentalmente un lenguaje de *scripting* para máquinas virtuales.

Vagrant te permite gestionar y automatizar todo esto conjuntamente. Podrías tú mismo crear y lanzar una máquina con VirtualBox y luego ejecutar Ansible sobre ella manualmente, pero Vagrant te permite definir todo eso programáticamente y poder así ejecutarlo de manera automatizada. Esta configuración puede luego ser guardada en el repositorio junto con tu código y, cuando cambie, todos los miembros del equipo tendrán la última versión la próxima vez que se actualicen contra el repositorio. Aunque lo vamos a usar con VMs, Vagrant también se podría utilizar para controlar *hardware* real.

Crea tu entorno

Una vez que ya tienes las dependencias instaladas, puedes crear la máquina virtual en la que vas a instalar paquetes y configurarlos mediante Ansible.

Crea un nuevo directorio con el nombre ansible-test, accede a él y ejecuta el siguiente comando:

```
 vagrant init ubuntu/bionic64
```

Esto generará el fichero en ese directorio. En el siguiente ejemplo creamos un directorio donde vamos a crear la VM con Vagrant, y vemos el resultado de esta operación:

```
$ mkdir ansible-test && cd ansible-test
```

```
$ vagrant init ubuntu/bionic64
```

```
A `Vagrantfile` has been placed in this directory. You are now
```

```
ready to `vagrant up` your first virtual environment! Please read
```

```
the comments in the Vagrantfile as well as documentation on
```

```
`vagrantup.com` for more information on using Vagrant.
```

Cuando el fichero se ha creado, el comando nos permite «levantar» la máquina virtual, que, si no está ya creada, se creará y arrancará, y en caso de estar parada, simplemente se arrancará. La primera vez que lo ejecutas, este comando realiza una serie de acciones. Primero comprobará si en tu máquina existe una caja (imagen de máquina virtual que maneja Vagrant) con el nombre . Si no existe, la descargará de Atlas, un repositorio centralizado de cajas que está mantenido por Hashicorp (la empresa responsable de Vagrant).

Si todavía no cuentas con la caja en tu *host*, el comando mostrará algo como:

```
$ vagrant up
```

```
Bringing machine 'default' up with 'virtualbox' provider...
```

```
==> default: Box 'ubuntu/bionic64' could not be found. Attempting to find and
install…
```

```
default: Box Provider: virtualbox
```

```
default: Box Version:>= 0
```

```
==> default: Loading metadata for box 'ubuntu/bionic64'
```

```
default: URL: https://atlas.hashicorp.com/ubuntu/bionic64
```

```
==> default: Adding box 'ubuntu/bionic64' (20200519.1.0) for provider:
virtualbox
```

```
default: Downloading:
https://atlas.hashicorp.com/ubuntu/boxes/bionic64/versions/20200519.1.0/providers/virtualbox.box
```

```
==> default: Successfully added box 'ubuntu/bionic64' (20200519.1.0) for
'virtualbox'!
```

Una vez que la caja ya está en tu sistema, bien porque se acabe de descargar o bien porque ya existiera, la ejecución sigue su proceso importando una copia de la caja en cuestión en el directorio de trabajo ( ), pudiendo así reutilizar la misma caja desde distintos directorios. Seguidamente, a través de VirtualBox se crea una máquina virtual basada en esta imagen y se arranca. La salida de pantalla será similar a la siguiente:

```
==> default: Importing base box 'ubuntu/bionic64'…
```

```
==> default: Matching MAC address for NAT networking…
```

```
==> default: Checking if box 'ubuntu/bionic64' is up to date…
```

```
==> default: Setting the name of the VM: ansible-test
```

```
default_1452487432943_ 66014
```

```
==> default: Clearing any previously set forwarded ports…
```

```
==> default: Clearing any previously set network interfaces…
```

```
==> default: Preparing network interfaces based on configuration…
```

```
default: Adapter 1: nat
```

```
==> default: Forwarding ports…
```

```
default: 22 => 2222 (adapter 1)
```

```
==> default: Booting VM…

==> default: Waiting for machine to boot. This may take a few minutes…

default: SSH address: 127.0.0.1:2222

default: SSH username: vagrant

default: SSH auth method: private key

default: Warning: Connection timeout. Retrying…

default:

default: Vagrant insecure key detected. Vagrant will automatically replace

default: this with a newly generated keypair for better security.

default:

default: Inserting generated public key within guest…

default: Removing insecure key from the guest if it's present…

default: Key inserted! Disconnecting and reconnecting using new SSH key…

==> default: Machine booted and ready!

==> default: Checking for host entries

==> default: Mounting shared folders…

default: /vagrant => /Users/ansible/ansible-test
```

La máquina virtual estará ya creada y ejecutándose, lo que puedes comprobar mediante el siguiente comando de Vagrant:

```
$ vagrant status

Current machine states:
```

```
default running (virtualbox)
```

También puedes entrar dentro de la máquina virtual mediante el comando . Esto hará que accedas a la máquina virtual usando una clave SSH que fue generada por Vagrant cuando la VM se estaba creando. Una vez dentro de la máquina virtual, puedes comprobar que ejecutas la máquina correcta mediante el comando :

```
vagrant@ubuntu-bionic:~$ cat /etc/issue
```

```
Ubuntu 18.04.1 LTS \n \l
```

Llegados a este punto, has logrado crear e iniciar una máquina virtual usando Vagrant. Ahora que ya has hecho esto, podrás probar por ti mismo todas las indicaciones que se darán a lo largo de las siguientes secciones. Dado que no necesitas esta máquina virtual de momento, vamos a eliminarla mediante el comando :

```
 $ vagrant destroy
```

```
default: Are you sure you want to destroy the 'default' VM? [y/N] y
```

```
==> default: Forcing shutdown of VM…
```

```
==> default: Destroying VM and associated drives…
```

```
==> default: Removing hosts
```

Cuando ejecutas el comando y respondes ‘y’ al mensaje de confirmación, provocarás que la máquina virtual se elimine, perdiendo a su vez cualquier operación o cambio que hubieras realizado. Si vuelves a ejecutar el comando , se volverá a copiar la imagen de partida nuevamente, y volverás a estar en la situación inicial.

Vamos ahora a modificar el que se generó, ya que por defecto las máquinas que se crean tienen asignados 489 MB de memoria RAM. Esta cantidad de memoria es escasa para nuestro cometido, por lo que vamos a aumentarla a 1024 MB de memoria para trabajar con cierta fluidez, así que edita el e incluye el siguiente fragmento antes de la última línea que contiene la palabra :

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

```
 vb.memory = "1024"
```

```
end
```

Esto le indica a Vagrant que asigne 1024 MB de memoria al crear la máquina virtual.

Para además indicarle a Vagrant que el aprovisionamiento de esta máquina virtual lo realizaremos mediante Ansible, es necesario incluir en el archivo de configuración de Vagrant el siguiente fragmento, también antes de la última línea con :

```
config.vm.provision "ansible" do |ansible|
```

```
 ansible.playbook ="provisioning/playbook.yml"
```

```
end
```

Esto le dice a Vagrant que utilice Ansible para aprovisionar la máquina virtual, ejecutando el *playbook* que se llama dentro de un subdirectorio que se encuentra en el directorio actual (ruta relativa).

Si Ansible no está instalado en tu máquina (por ejemplo, porque utilices Windows como sistema operativo), necesitas utilizar el aprovisionamiento con tal como:

```
config.vm.provision "ansible_local" do |ansible|
```

```
 ansible.playbook ="provisioning/playbook.yml"
```

```
end
```

La diferencia entre estos dos fragmentos de configuración de Vagrant es que uno utiliza ansible, mientras que el otro, . Al utilizar ansible estarás ejecutando Ansible desde tu máquina *host* para aplicar el *playbook* sobre la máquina virtual, mientras que con lo ejecutará desde dentro de la máquina virtual. Vagrant se encargará de la instalación y la configuración de Ansible dentro de la máquina virtual automáticamente para poder realizar el aprovisionamiento.

Si ejecutamos ahora , el error que se muestra nos estará indicando que el archivo al que se hace referencia no existe aún.

```
$ vagrant up
```

```
Bringing machine 'default' up with 'virtualbox' provider...

There are errors in the configuration of this machine. Please fix

the following errors and try again:

ansible provisioner:

* `playbook` does not exist on the host:

/Users/ansible/ansible- test/provisioning/playbook.yml
```

Crearemos a continuación el fichero que vamos a necesitar implementar para Ansible, evitando así el error que muestra Vagrant al provisionar la máquina virtual. Por tanto, debemos crear primero la carpeta y, dentro de esta, un archivo que se llame . Una vez que ambos están creados, ya podrás ejecutar para intentar crear la VM, aunque esta vez mostrará un error distinto debido a que el fichero que acabamos de crear al que hacemos referencia no tiene un formato YAML válido:

```
ERROR! playbooks must be a list of plays

Ansible failed to complete successfully. Any error output should be

visible above. Please fix these errors and try again.
```

Hemos logrado avanzar, en cualquier caso, ya que este error indica que Ansible se está ejecutando y trata de leer el fichero indicado, pero no es un *playbook* con un formato adecuado.

Seguramente verás errores similares en otras ocasiones en el futuro, debido a que Ansible es bastante estricto con respecto a cómo están formateados sus ficheros *playbook*. El significado del error y el cómo arreglarlo lo explicaremos en el apartado siguiente.

5.5. Introducción a los playbooks

Como ya se ha mencionado anteriormente, Ansible utiliza YAML como formato de ficheros para definir el estado de la configuración. Estos ficheros son los denominados *playbooks*, en los que, a través de una terminología propia, se define el estado deseado mediante un listado de tareas, compuestas por un conjunto de comandos y argumentos, y cualquier otro dato de configuración que pudiera requerirse. Estas tareas se pueden ejecutar síncrona o asíncronamente, dependiendo de la naturaleza de la tarea y lo que el *playbook* requiera. Estas definiciones requieren una sintaxis muy reducida que lo hacen fácilmente legible y entendible, tanto para las máquinas como para las personas.

Los *playbooks* se parecen más a un modelo de sistemas que a un lenguaje de programación o *script*. Salvo unas contadas excepciones (que veremos más adelante), se trata de definir el estado deseado de un sistema y dejar que Ansible se asegure de que las máquinas estén en ese estado definido.

Un ejemplo de esto sería el desarrollo de un *playbook* que especifique que el programa PHP se debe instalar en el *host* destino. Al ejecutar el *playbook*, Ansible lo instalará por ti si no está ya instalado, usando el gestor de paquetes que le indiques que debe utilizar. Ansible es capaz de detectar si ya está instalado y, en tal caso, no ejecutará nada. En la siguiente sección, vamos a desarrollar este *playbook*.

Tu primer *playbook*

Ya hemos visto que un *playbook* de Ansible no es más que un archivo YAML con una sintaxis específica. No se trata de un nuevo lenguaje que debes aprender (como en el caso de Puppet) o de código ejecutable (como en el caso de Chef), sino que se trata de formato YAML estándar.

Los archivos YAML comienzan por definir una sección de metadatos (habitualmente

conocida como asunto frontal – *front matter*). Dado que no es necesario añadir ningún metadato, no escribiremos nada en esta sección e iniciaremos nuestro *playbook* con tres guiones seguidos en una sola línea (que indican el final de la sección de metadatos, que está vacía en nuestro caso).

Debido a que los *playbooks* son archivos YAML estándar, deben seguir las mismas reglas que cualquier otro archivo YAML. La mayoría de las veces tropezarás con el hecho de que los ficheros YAML son sensibles a los espacios en blanco, por lo que los espacios y tabuladores en tus ficheros realmente significan algo.

Después de esta primera línea de tres guiones que cierra el asunto frontal, lo primero que debes hacer es decirle a Ansible contra qué máquinas debe ejecutarse este *playbook*, especificando el *host* o grupo de *hosts* en donde ejecutar. Por el momento le diremos a Ansible que ejecute en todos los *hosts* que estén disponibles mediante la línea - en nuestro archivo *playbook*. Ahora, el *playbook* (ubicado en) contendrá lo siguiente:

```
---
```

```
- hosts: all
```

Con esto, Ansible ya sabe «dónde» ejecutar, y ahora puedes decirle «qué» es lo que quieres que ejecute. La sección de tareas ( ) es la que debemos utilizar para ello. Dentro de esta sección, le vamos a indicar a Ansible que haga un ping de las máquinas para asegurarnos de que puede conectarse a ellas:

```
---
```

```
- hosts: all
```

```
 tasks:
```

```
 - ping:
```

A continuación, si la máquina ya estaba creada, puedes ejecutar Ansible mediante , o si no, crea y aprovisiona la máquina todo de una vez con . Verás una salida similar a la siguiente:

```
$ vagrant provision
```

```
==> default: Running provisioner: ansible…
```

```
PLAY [all] *********************************************************
```

```
TASK [setup] *******************************************************
```

```
ok: [default]
```

```
TASK: [ping] *******************************************************
```

```
ok: [default]
```

```
PLAY RECAP *********************************************************
```

```
default: ok=2 changed=0 unreachable=0 failed=0
```

El bloque que dice te permite saber que tu acción se ha ejecutado correctamente. Esto nos indica que Ansible es capaz de conectarse a la máquina, por lo que podrá gestionarla. Dado que esta ejecución es bastante sencilla, es muy fácil de seguir la salida de la ejecución y entender lo que está pasando; a medida que la complejidad del *playbook* se incrementa, puedes imaginar que la salida se va complicando cada vez más y, por lo tanto, es más difícil de entender. Es por esto muy recomendable utilizar la posibilidad que nos brinda Ansible de añadir un nombre a cada tarea para explicar su propósito. En el siguiente fragmento añadimos el atributo nombre a la tarea:

```
---
```

```
- hosts: all
```

```
 tasks:
```

```
 - name: Make sure that we can connect to the machine
```

```
 ping:
```

En esta ocasión, la salida de la ejecución de Ansible va a mostrar el nombre de tarea establecido, en lugar del mensaje genérico: que mostraba anteriormente:

```
TASK: [Make sure that we can connect to the machine]
***************
```

```
ok: [default]
```

Hemos verificado que Ansible es capaz de conectarse al *host*. Podrías haber conectado también con ella usando directamente, pero una vez ahí te encontrarías en una máquina vacía sin nada configurado todavía. Vamos a añadir una tarea adicional para decirle a Ansible que instale algunos paquetes en tu máquina virtual.

Vamos a instalar el *software* de código abierto PHP de desarrollo. Lo vamos a añadir mediante otra tarea en nuestro fichero , que ahora se parecerá al siguiente:

```
---
```

```
- hosts: all
```

```
 tasks:
```

```
 - name: Make sure that we can connect to the machine
```

```
ping:
```

```
- name: Install PHP
```

```
apt: name=php state=present update_cache=yes
```

El módulo ping lo habíamos utilizado para verificar la conexión con el *host*. Esta vez, el módulo que estamos utilizando es . El módulo que se quiere utilizar se especifica antes de los dos puntos (:), mientras que los atributos se especifican detrás. El módulo ping no requiere de ningún atributo, mientras que el módulo tiene varios de ellos, algunos necesarios para que el módulo sepa lo que debe hacer. En nuestro caso, solo vamos a usar tres de ellos: nombre ( ) para identificar el paquete, estado ( ) para indicar el estado deseado y actualizar caché ( ) para refrescar la caché de .

Puedes encontrar un listado completo de todos los argumentos que soporta cada módulo en la documentación oficial de Ansible. Para el módulo : https://docs.ansible.com/ansible/latest/modules/apt_module.html

Al haber utilizado el módulo , le estamos indicando a Ansible que queremos que el paquete de nombre esté instalado. El atributo admite cualquier valor de una lista de posibles valores, incluyendo y .

El atributo indica a Ansible que actualice la caché del gestor de paquetes primero. Si ejecutas nuevamente , Ansible debería intentar instalar el paquete . Desgraciadamente, esto fallará, y mostrará el siguiente mensaje de error:

```
TASK: [Install PHP]
**********************************************
```

```
fatal:
```

```
[default]: FAILED! => {"changed": false, "cmd": "apt- get update",
```

```
"msg": "E: Could not open lock file /var/lib/apt/lists/lock - open
(13: Permission denied)
```

```
[…]
```

La ejecución ha devuelto un error debido a que Ansible está conectando con el usuario , que es el que se configura por defecto para acceso a la máquina virtual, y este usuario no cuenta con permisos suficientes para instalar paquetes. Aquí es donde la opción (que controla con qué usuario se ejecutan los comandos) es útil. se puede añadir en dos lugares distintos de tu *playbook*: o bien se añade en la tarea que requiere más permisos, o bien puede añadirse a nivel *playbook*, lo que provocará que todo comando del *playbook* se ejecute con permisos de administrador.

Para añadirlo a una acción individual:

```
- name: Install PHP
```

```
 apt: name=php5-cli state=present update_cache=yes
```

```
 become: true
```

En su lugar, puedes añadirlo a nivel general de todo el *playbook*, dado que varios de los comandos que vas a ejecutar requieren estos permisos. Después de añadirlo, tu *playbook* quedará de la siguiente manera:

```
---
```

```
- hosts: all
```

```
 become: true
```

```
 tasks:
```

```
 - name: Make sure that we can connect to the machine
```

```
ping:

- name: Install PHP

apt: name=php state=present update_cache=yes
```

Al guardar los cambios y volver a ejecutar , la salida de Ansible debería indicar que PHP se ha instalado correctamente:

```
TASK [Install PHP] *****************************************

changed: [default]
```

Ahora puedes añadir más pasos para instalar y , agregando más tareas con el módulo e indicando que quieres que estén presentes y como muestra el siguiente fragmento:

```
---

- hosts: all

become: true

tasks:

- name: Make sure that we can connect to the machine

ping:

- name: Install PHP

apt: name=php state=present update_cache=yes

- name: Install nginx
```

```
 apt: name=nginx state=present
```

```
 - name: Install mySQL
```

```
 apt: name=mysql-server state=present
```

Lo mismo que ocurrió al ejecutar con el paquete , cuando ejecutes otra vez , la salida obtenida será:

```
TASK: [Install nginx] ****************************************
```

```
changed: [default]
```

```
TASK: [Install mySQL] ****************************************
```

```
changed: [default]
```

Tras estas operaciones, vamos a acceder dentro de la máquina virtual mediante el comando para comprobar que todo se ha instalado correctamente. Luego, con los comandos que se muestran a continuación, podrás comprobar que los programas en cuestión están instalados:

```
vagrant@vagrant-ubuntu-bionic:~$ which php
```

```
/usr/bin/php
```

```
vagrant@vagrant-ubuntu-bionic:~$ which nginx
```

```
/usr/sbin/nginx
```

```
vagrant@vagrant-ubuntu-bionic:~$ which mysqld
```

```
/usr/sbin/mysqld
```

Llegados hasta este punto, ya tenemos automatizada la instalación de estas

herramientas en la máquina virtual mediante Ansible, por lo que, si ahora eliminas la máquina virtual y la vuelves a crear de nuevo, el proceso de aprovisionamiento configurado volverá a instalar automáticamente estos paquetes. Por tanto, puedes ejecutar , y seguidamente , para comprobarlo.

Ya hemos terminado la labor de crear nuestro primer *playbook*, mediante el que instalamos los paquetes que hemos incluido en la lista de tareas. Ahora lo que queda por hacer es limpiar un poco, es decir, eliminar las tareas que no son requeridas y los duplicados.

Lo primero que vamos a hacer es borrar la tarea de ping. Dado que ya hemos comprobado que Ansible se puede conectar a la máquina, no es necesario realizar la misma tarea cada vez.

Otra mejora que podemos realizar es la de combinar todas las tareas del módulo apt en una sola y utilizar como parámetro una lista de elementos. Para entender cómo funciona esto basta decir que vamos a proporcionar una lista de elementos dentro del argumento name en vez de un único valor. Ansible entonces llamará a tu tarea con todos los elementos de la lista. Vamos a desglosar la tarea en múltiples líneas para que se vea más claro:

```
---

- hosts: all

 become: true

 tasks:

 - name: Install required packages

 apt:
```

```
 state: present
```

```
 update_cache: yes
```

```
 name:
```

```
 - php
```

```
 - nginx
```

```
 - mysql-server
```

Si ejecutas otra vez , debería contraer toda la salida para esa tarea dentro de un bloque, lo que reduce significativamente la salida:

```
$ vagrant provision
```

```
==> default: Running provisioner: ansible…
```

```
default: Running ansible playbook…
```

```
PLAY [all]
**********************************************************
```

```
TASK [setup]
**********************************************************
```

```
ok: [default]
```

```
TASK [Install required packages]
*********************************
```

```
ok: [default]
```

```
PLAY RECAP
**********************************************************
```

```
default: ok=2 changed=0 unreachable=0 failed=0
```

Llegados hasta aquí, tienes una máquina con todas las dependencias necesarias instaladas, y no importa cuántas veces ejecutes el *playbook* ahora que el estado resultante de la configuración va a ser siempre el mismo.

Playbooks e idempotencia

Idempotencia se refiere a la posibilidad de hacer algo muchas veces sin que el resultado varíe en cualquiera de las veces. Para Ansible, un *playbook* es idempotente si el resultado de ejecutarlo sucesivas veces no varía el estado de configuración obtenido tras la primera ejecución, con lo que ese estado de la configuración es estable e invariante, independientemente del número de veces que se ejecute el *playbook*.

Considerando el *playbook* del ejemplo anterior, la primera vez que lo ejecutas evalúa su contenido y aplica las transformaciones necesarias para asegurar que PHP, NGinX y MySQL se instalan. Puedes comprobar en la salida de Ansible que indica que el estado ha cambiado:

```
PLAY [all]
*****************************************************
```

```
GATHERING FACTS
*****************************************************
```

```
ok: [default]
```

```
TASK: [Install required packages]
*********************************
```

```
changed: [default]
```

```
PLAY RECAP
```

```
**********************************************************
```

```
default: ok=2 changed=1 unreachable=0 failed=0
```

Si ejecutas el *playbook* nuevamente, en lugar de indicar que ha habido cambios, dirá que está ok. Esto se debe a que el módulo comprobará antes que nada si el paquete que se desea instalar ya estuviera instalado en el sistema. Dado que en este caso los paquetes ya se encuentran instalados, no realizará ningún cambio. Por tanto, se puede decir que el *playbook* es idempotente:

```
PLAY [all]
**********************************************************
```

```
GATHERING FACTS
**********************************************************
```

```
ok: [default]
```

```
TASK: [Install required packages]
*********************************
```

```
ok: [default]
```

```
PLAY RECAP
**********************************************************
```

```
default: ok=2 changed=0 unreachable=0 failed=0
```

La mayoría de los módulos de Ansible que puedes utilizar en tus *playbooks* son idempotentes, pero también hay algunos módulos que, dependiendo de cómo los uses, pueden romper la idempotencia, como son los módulos o , que siempre se ejecutarán al procesar el *playbook*, dado que Ansible no puede determinar por sí solo si la acción que va a ejecutar el comando se ha realizado ya anteriormente o no. Por ejemplo, si borras un archivo a través de un comando, la siguiente vez que ejecutes

ya no existirá ese archivo, por lo que podrá fallar la ejecución. Para hacer que tus comandos personalizados sean también idempotentes, puedes añadir condiciones a tareas para indicar cuándo ejecutarlas.

5.6. 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

Red Hat Inc. (2020). *Ansible Documentation*. https://docs.ansible.com/ansible/latest/index.html

En el sitio oficial de documentación Ansible es donde podrás encontrar la documentación de referencia más completa y actualizada de la herramienta, así como la versión de documentación correspondiente con la propia versión de la herramienta que estés utilizando. Es el primer recurso al que acceder en busca que cualquier concepto Ansible, para asegurarnos de su veracidad y actualidad.

Ansible Tips and Tricks

DePorter, M. (2018, Agosto 2). *Ansible Tips and Tricks*. GitHub. https://github.com/nfaction/ansible-tips-and-tricks/wiki

Esta wiki proporciona una guía adicional a la documentación oficial de referencia de Ansible, mencionada anteriormente. Proporciona ejemplos adicionales más complejos o completos a los que se encuentran en la documentación oficial.

Repositorio de GitHub de Ansible

Open Source community. (2020). *Ansible*. GitHub. https://github.com/ansible/ansible

El repositorio GitHub de Ansible es donde se desarrolla el código fuente de la herramienta y donde se puede encontrar la versión más actualizada, ya que no para de recibir contribuciones y actualizar sus funcionalidades por parte de la comunidad *open source*.

Referencia del módulo apt

Williams, M. (2021, junio 14). *ansible.builtin.apt – Manages apt-packages*. Ansible. https://docs.ansible.com/ansible/latest/modules/apt_module.html

Uno de los módulos más utilizados en este tema ha sido el módulo apt. En este enlace del sitio oficial de documentación de Ansible podrás encontrar la referencia completa y actualizada de este módulo, junto con todos los posibles argumentos y modos de uso que soporta.

Test

1. ¿En qué lenguaje está escrito Ansible?

   A. YAML.

   B. Ruby.

   C. Python.

   D. Ninguno de los anteriores.

2. ¿Qué lenguaje emplean los ficheros *playbook*?

   A. XML.

   B. YAML.

   C. HTML.

   D. Properties.

3. ¿Qué diferencia principal existe entre Ansible y Puppet/Chef?

   A. Puppet y Chef usan un servidor centralizado cada uno, Ansible no tiene ningún servidor centralizado (funciona sin agente, *agentless*).

   B. Las máquinas gestionadas por Ansible preguntan periódicamente para saber si hay cambios de configuración, mientras que Puppet y Chef, no.

   C. No existen diferencias a nivel funcional entre las tres herramientas.

   D. Ninguna de las anteriores.

4. ¿Qué diferencia la distribución de cambios entre Ansible, Puppet y Chef?

A. La distribución de cambios usa el mismo modelo en los tres.

B. Puppet y Chef requieren que el usuario distribuya explícitamente los cambios cuando necesite a los nodos requeridos, mientras que Ansible los distribuye automáticamente.

C. La diferencia es que el usuario es el responsable de distribuir las actualizaciones a las máquinas, mientras que con Puppet y Chef se pueden guardar (*commit*) los cambios a fin de ser distribuidos.

D. Ninguna de las anteriores.

5. ¿Cómo puedes utilizar Vagrant para provisionar Ansible, si no lo tienes instalado en tu máquina?

A. No es posible. Debes instalar Ansible en tu máquina.

B. Vagrant se encarga de instalarlo en tu máquina automáticamente.

C. Utilizando el aprovisionador “ ".

D. Utilizando el aprovisionador “ ".

6. ¿Cómo suelen empezar los *playbooks* de Ansible?

A. Con una línea en blanco.

B. Con tres guiones.

C. Con tres espacios.

D. Con .

7. ¿Cómo se llaman las operaciones individuales de configuración con las que dices a Ansible "qué" ejecutar?

A. Acciones.

B. Comandos.

C. Funciones.

D. Tareas.

8. ¿Cómo es posible identificar adecuadamente una tarea que ejecuta Ansible?

A. Añadiendo el atributo name con una descripción adecuada a la tarea.

B. Poniendo un comentario en el código.

C. Añadiendo una descripción a la tarea.

D. No es posible proporcionar una identificación personalizada a una tarea. Ansible muestra en la salida de la ejecución el tipo de tarea para que la identifiques.

9. ¿Cómo puedes ejecutar en Ansible una tarea con privilegios de administrador?

A. Usando el módulo

B. Usando el atributo

C. Usando el módulo

D. Usando la opción

10. ¿Qué significa el término idempotencia?

A. Que Ansible es igual de potente que Puppet o Chef.

B. Que puedes hacer algo muchas veces y el resultado será siempre el mismo.

C. Que lo que ejecutes con Ansible será igual de potente que si lo ejecutas manualmente.

D. Ninguna de las anteriores.