Learning Ansible with Rocky (Spanish version)

A book from the Documentation Team

Version : 2024/04/19

Rocky Documentation Team

Table of contents

1. Licence

RockyLinux offers Linux courseware for trainers or people wishing to learn how to administer a Linux system on their own.

RockyLinux materials are published under Creative Commons-BY-SA. This means you are free to share and transform the material, while respecting the author's rights.

BY : **Attribution**. You must cite the name of the original author.

SA : **Share Alike**.

- Creative Commons-BY-SA licence : https://creativecommons.org/licenses/by-sa/4.0/

The documents and their sources are freely downloadable from:

- https://docs.rockylinux.org
- https://github.com/rocky-linux/documentation

Our media sources are hosted at github.com. You'll find the source code repository where the version of this document was created.

From these sources, you can generate your own personalized training material using mkdocs. You will find instructions for generating your document here.

How can I contribute to the documentation project?

You'll find all the information you need to join us on our git project home page.

We wish you all a pleasant reading and hope you enjoy the content.

2. Aprender Ansible con Rocky

Ansible es un sencillo, pero potente motor de automatización para Linux. Este tutorial le guiará a través de los conceptos de uso de Ansible para automatizar sus tareas de TI de una manera que divertida e informativa. El uso de los ejercicios a lo largo de estos capítulos, le ayudará a ganar un nivel de comodidad con Ansible y su aplicacion en el mundo real.

3. Conceptos básicos de Ansible

En este capítulo aprenderá cómo trabajar con Ansible.

Objetivos: En este capítulo aprenderá a:

✔ Implementar Ansible;
✔ Aplicar cambios sobre la configuración en un servidor;
✔ Crear los primeros playbooks de Ansible;

🏁 **ansible**, **modulo**, **playbook**

Conocimiento: ★ ★ ★
Complejidad: ★ ★

Tiempo de lectura: 30 minutos

Ansible se encarga de centralizar y automatizar las tareas de administración. Es una herramienta:

- **sin agentes** (no es necesario hacer despliegues específicos en los clientes),
- **idempotente** (mismo resultado cada vez que se ejecuta)

Utiliza el protocolo **SSH** para configurar remotamente los clientes Linux o el protocolo **WinRM** para trabajar con clientes Windows. Si no se dispone de ninguno de estos protocolos, siempre es posible que Ansible utilice una API, lo que lo convierte en una auténtica navaja suiza para la configuración de servidores, estaciones de trabajo, servicios Docker, equipos de red, etc. (De hecho, casi todo).

> ⚠ **Warning**
>
> La apertura de flujos SSH o WinRM a todos los clientes desde el servidor de Ansible, lo convierte en un elemento crítico de la arquitectura que debe ser cuidadosamente supervisado.

Como Ansible está basado en push, no mantendrá el estado de sus servidores entre cada una de sus ejecuciones. Todo lo contrario, Ansible realizará nuevas comprobaciones de estado cada vez que se ejecute. Se dice que no tiene estado.

Le ayudará con:

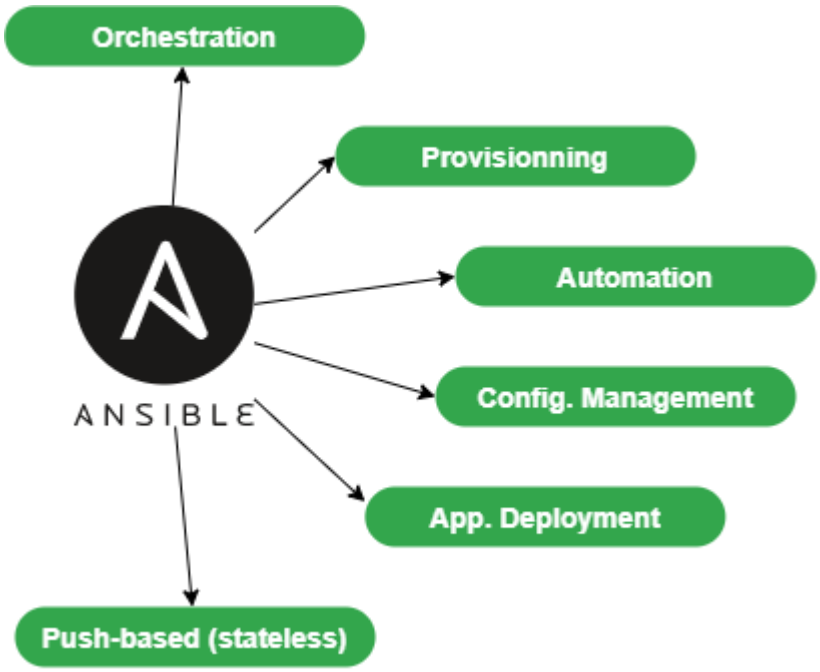
- el aprovisionamiento (despliegue de una nueva VM),
- el despliegue de aplicaciones,
- la gestión de la configuración,
- la automatización,
- la orquestación (cuando se utiliza más de un objetivo).

Note

Ansible fue escrito originalmente por Michael DeHaan, fundador de otras herramientas como Cobbler.

La primera versión fue la 0.0.1, publicada el 9 de marzo de 2012.

El 17 de octubre de 2015, AnsibleWorks (la empresa detrás de Ansible) fue adquirida por Red Hat por 150 millones de dólares.



Para ofrecer una interfaz gráfica a su uso diario de Ansible, puede instalar algunas herramientas como Ansible Tower (RedHat), que no es gratuita, su homólogo opensource Awx, también puede utilizar otros proyectos como Jenkins o el excelente Rundeck.

Abstract

Para seguir esta guia, necesitará al menos 2 servidores ejecutando Rocky8:

- la primera será la **máquina de gestión**, en ella se instalará Ansible.
- la segunda será el servidor a configurar y administrar (otro servidor con una versión de Linux distinta a Rocky Linux será igual de valida).

En los ejemplos siguientes, la máquina de gestión tendrá la dirección IP 172.16.1.10 y la estación gestionada tendrá la 172.16.1.11. Es usted quien debe adaptar los ejemplos según su plan de direccionamiento IP.

3.1 El vocabulario de Ansible

- La **máquina de gestión**: La máquina en la que está instalado Ansible. Dado que Ansible es una herramienta **sin agente**, no despliega ningún software en los servidores gestionados.
- El **inventario**: Es un archivo que contiene información sobre los servidores gestionados.
- Las **tareas**: una tarea es un bloque que define un procedimiento a ejecutar (por ejemplo, crear un usuario o un grupo, instalar un paquete de software, etc.).

- Un **módulo**: un módulo abstrae una tarea. Ansible le proporciona muchos módulos.
- Los ** libros de jugadas **: un archivo simple en formato yaml que define los servidores de destino y las tareas a realizar.
- Un **rol**: Un rol permite organizar los playbooks y todos los demás archivos necesarios (plantillas, scripts, etc.) para facilitar la compartición y reutilización del código.
- Una **colección**: Una colección es un conjunto lógico de playbooks, roles, módulos y plugins.
- Las **hechos**: Son variables globales que contienen información sobre el sistema (nombre de la máquina, versión del sistema, interfaz de red y configuración, etc.).
- Los **manejadores**: Se utilizan para hacer que un servicio se detenga o se reinicie en caso de cambio.

3.2 Instalación en el servidor de gestión

Ansible está disponible en el repositorio *EPEL* pero viene como versión 2.9.21, la cual es bastante antigua. Puede ver cómo se hace siguiendo este procedimiento, pero sáltese los pasos de instalación, ya que vamos a instalar la última versión. *EPEL* es necesario para ambas versiones, así que puedes seguir adelante e instalarlo ahora:

- Instalación de EPEL:

```
$ sudo dnf install epel-release
```

Si estuviésemos instalando Ansible desde el *EPEL* podríamos hacer lo siguiente:

```
$ sudo dnf install ansible
$ ansible --version
2.9.21
```

Como queremos utilizar una versión más reciente de Ansible, la instalaremos desde `python3-pip`:

Note

Elimine Ansible si lo ha instalado previamente desde *EPEL*.

```
$ sudo dnf install python38 python38-pip python38-wheel python3-argcomplete
rust cargo curl
```

Note

`python3-argcomplete` es proporcionado por *EPEL*. Por favor, instale epel-release si aún no lo ha hecho. Este paquete le ayudará a completar los comandos Ansible.

Antes de instalar Ansible, necesitamos indicarle a Rocky Linux que queremos utilizar la nueva versión de Python. La razón es que si continuamos con la instalación sin hacer esta configuración, la versión por defecto de Python3 (versión 3.6 en el momento de escribir este documento), se utilizará en lugar de la nueva versión 3.8. Establezca la versión que desea utilizar introduciendo el siguiente comando:

```
sudo alternatives --set python /usr/bin/python3.8
sudo alternatives --set python3 /usr/bin/python3.8
```

Ahora podemos instalar Ansible:

```
$ sudo pip3 install ansible
$ sudo activate-global-python-argcomplete
```

Compruebe su versión de Ansible:

```
$ ansible --version
ansible [core 2.11.2]
  config file = None
  configured module search path = ['/home/ansible/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/lib/python3.8/site-packages/
ansible
  ansible collection location = /home/ansible/.ansible/collections:/usr/share/
ansible/collections
  executable location = /usr/local/bin/ansible
  python version = 3.8.6 (default, Jun 29 2021, 21:14:45) [GCC 8.4.1 20200928
(Red Hat 8.4.1-1)]
```

```
  jinja version = 3.0.1
  libyaml = True
```

3.3 Archivos de configuración

La configuración del servidor se encuentra en `/etc/ansible`.

Hay dos archivos de configuración principales:

- El archivo de configuración principal `ansible.cfg` es donde residen los comandos, módulos, plugins y la configuración de ssh;
- El fichero de inventario de gestión de máquinas cliente `hosts` es el lugar donde se declaran los clientes y los grupos de clientes.

Como hemos instalado Ansible con `pip`, esos archivos no existen. Tendremos que crearlos a mano.

Aquí se proporciona un ejemplo del archivo `ansible.cfg` y un ejemplo del archivo `hosts` aquí.

```
$ sudo mkdir /etc/ansible
$ sudo curl -o /etc/ansible/ansible.cfg https://raw.githubusercontent.com/
ansible/ansible/devel/examples/ansible.cfg
$ sudo curl -o /etc/ansible/hosts https://raw.githubusercontent.com/ansible/
ansible/devel/examples/hosts
```

También puede utilizar el comando `ansible-config` para generar un nuevo archivo de configuración:

```
usage: ansible-config [-h] [--version] [-v] {list,dump,view,init} ...

Ver la configuración de Ansible.

positional arguments:
  {list,dump,view,init}
    list                Print all config options
    dump                Dump configuration
    view                View configuration file
    init                Create initial configuration
```

Ejemplo:

```
ansible-config init --disabled > /etc/ansible/ansible.cfg
```

La opción `--disabled` permite comentar un conjunto de opciones anteponiendo un `;`.

3.3.1 El archivo de inventario `/etc/ansible/hosts`

Como Ansible tendrá que trabajar con todos sus equipos para ser configurado, es muy importante proporcionar uno (o más) archivos de inventario bien estructurados, que se ajusten perfectamente a su organización.

A veces es necesario pensar detenidamente en cómo construir este archivo.

Vaya al archivo de inventario por defecto, que se encuentra en `is/ansible/hosts`. En este fichero, se proporcionan y comentan algunos ejemplos:

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers:

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

# If you have multiple hosts following a pattern, you can specify
```

```
# them like this:

## www[001:006].example.com

# Ex 3: A collection of database servers in the 'dbservers' group:

## [dbservers]
##
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57

# Here's another example of host ranges, this time there are no
# leading 0s:

## db-[99:101]-node.example.com
```

Como puede ver, el archivo proporcionado como ejemplo utiliza el formato INI, que es bien conocido por los administradores de sistemas. Por favor, ten en cuenta que puede elegir otro formato de archivo (como yaml, por ejemplo), pero para las primeras pruebas, el formato INI se adapta bien a nuestros ejemplos.

Obviamente, en un entorno de producción, se puede generar automáticamente el inventario, especialmente si se dispone de un entorno de virtualización como VMware VSphere o un entorno de computación en la nube (Aws, Openstack u otros).

- Crear un grupo de hosts en `/etc/ansible/hosts`:

Como puede haber notado, los grupos se declaran entre corchetes. Luego vienen los elementos que pertenecen a los grupos. Puede crear, por ejemplo, un grupo `rocky8` insertando el siguiente bloque de código en este archivo:

```
[rocky8]
172.16.1.10
172.16.1.11
```

Los grupos se pueden utilizar dentro de otros grupos. En este caso, debe especificarse que el grupo principal está compuesto por subgrupos mediante el uso del atributo `:chidren` tal y como se muestra en el siguiente ejemplo:

```
[linux:children]
rocky8
debian9

[ansible:children]
ansible_management
ansible_clients

[ansible_management]
172.16.1.10

[ansible_clients]
172.16.1.10
```

Por el momento no vamos a ir más lejos en el tema del inventario, pero si está interesado, considere revisar el siguiente enlace.

Ahora que nuestro servidor de administración está instalado y nuestro inventario está listo, es hora de ejecutar nuestros primeros comandos `ansible`.

3.4 Uso de la línea de comandos de `ansible`

El comando `ansible` lanza una tarea en uno o más hosts de destino.

```
ansible <host-pattern> [-m module_name] [-a args] [options]
```

Ejemplos:

> **Warning**
>
> Como todavía no hemos configurado la autenticación en nuestros 2 servidores de prueba, es posible que alguno de los ejemplos que se muestran a continuación no funcionen. Estos se dan como ejemplos para facilitar la comprensión, y serán completamente funcionales más adelante en este mismo capítulo.

- Enumera los hosts que pertenecen al grupo rocky8:

```
ansible rocky8 --list-hosts
```

- Hacer ping a un grupo de hosts con el módulo `ping`:

```
ansible rocky8 -m setup
```

- Mostrar los datos obtenidos desde un grupo de servidores utiliznaod el módulo `setup`:

```
ansible rocky8 -m setup
```

- Ejecutar un comando en un grupo de hosts invocando el módulo `command` con argumentos:

```
ansible rocky8 -m command -a 'uptime'
```

- Ejecutar un comando con permisos de administración:

```
ansible rocky8 -m command -a 'uptime'
```

- Ejecutar un comando utilizando un archivo de inventario personalizado:

```
ansible rocky8 -i ./local-inventory -m command -a 'date'
```

Note

Al igual que en este ejemplo, a veces es más sencillo separar la declaración de los dispositivos gestionados en varios archivos (por proyecto, por ejemplo) y proporcionar a Ansible la ruta a estos archivos, en lugar de mantener un único archivo de inventario.

Opción	Información
`-a 'argumentos'`	Los argumentos que se pasan al módulo.
`-b -K`	Solicita una contraseña y ejecuta el comando con los privilegios más altos.
`--user=usuario`	Utiliza el usuario indicado para conectarse al host de destino en vez de utilizar el usuario actual.
`--become-user=usuario`	Ejecuta la operación como el usuario indicado (por defecto: `root`).
`-C`	Simulación. No realiza ningún cambio en el objetivo, sino que lo prueba para ver qué debe cambiar.
`-m modulo`	Ejecuta el módulo indicado.

3.4.1 Preparando el cliente

Tanto en la máquina de gestión como en los clientes, crearemos un usuario `ansible` dedicado a las operaciones realizadas por Ansible. Este usuario tendrá que tener permisos de sudo, por lo que tendrá que ser añadido al grupo `wheel`.

Este usuario será usado:

- En la estación de administración: para ejecutar comandos `ansible` y ssh en los clientes administrados.
- En las estaciones administradas (aquí el servidor que sirve como estación de administración también sirve como cliente, para que sea administrado por sí mismo) para ejecutar los comandos lanzados desde la estación de administración: por lo tanto debe tener permisos de sudo.

En ambas máquinas, cree un usuario `ansible`:

```
$ sudo useradd ansible
$ sudo usermod -aG wheel ansible
```

Establecer una contraseña para este usuario:

```
$ sudo passwd ansible
```

Modifique la configuración de los sudoers para permitir a los miembros del grupo `wheel` hacer sudo sin la necesidad de introducir una contraseña:

```
$ sudo visudo
```

Nuestro objetivo es comentar el valor predeterminado, y descomentar la opción NOPASSWD para que estas líneas se vean así cuando hayamos terminado:

```
## Allows people in group wheel to run all commands
# %wheel  ALL=(ALL)       ALL

## Same thing without a password
%wheel        ALL=(ALL)       NOPASSWD: ALL
```

Warning

Si recibe el siguiente mensaje de error al introducir comandos de Ansible, probablemente significa que olvidó realizar este paso en uno de sus clientes: `"msg": "Missing sudo password`

Cuando utilice la gestión a partir de este momento, comience a trabajar con este nuevo usuario:

```
$ sudo su - ansible
```

3.4.2 Probar con el módulo ping

Por defecto Ansible no permite el inicio de sesión con contraseña.

Descomente la siguiente línea situada en la sección `[defaults]` dentro del archivo de configuración `/etc/ansible/ansible.cfg` y establézcala a True:

```
ask_pass      = True
```

Ejecute el `ping` en cada servidor del grupo rocky8:

```
# ansible rocky8 -m ping
SSH password:
172.16.1.10 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
172.16.1.11 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Note

Se le pide la contraseña `ansible` de los servidores remotos, lo cual es un problema de seguridad...

Tip

Si obtiene el error `"msg": "para utilizar el tipo de conexión 'ssh' con contraseña, debe instalar el programa sshpass"`, puede instalar `sshpass` en la estación de administración:

```
$ sudo dnf install sshpass
```

Abstract

Ahora puede probar los comandos que no funcionaban previamente en este capítulo.

3.5 Autenticación mediante claves

La autenticación por contraseña será reemplazada por una autenticación de clave privada/pública mucho más segura.

3.5.1 Crear una clave SSH

La clave dual se generará con el comando `ssh-keygen` en la estación de gestión por el usuario `ansible`:

```
[ansible]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansible/.ssh/id_rsa.
Your public key has been saved in /home/ansible/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Oa1d2hYzzdO0e/K10XPad25TA1nrSVRPIuS4fnmKr9g
ansible@localhost.localdomain
The key's randomart image is:
+---[RSA 3072]----+
|           .o . +|
|           o . =.|
|          . . + +|
|         o . = =.|
|        S o = B.o|
|         = + = =+|
|        . + = o+B|
|         o + o *@|
|        . Eoo .+B|
+----[SHA256]-----+
```

La clave pública se puede copiar a los servidores:

```
# ssh-copy-id ansible@172.16.1.10
# ssh-copy-id ansible@172.16.1.11
```

Vuelva a comentar la siguiente línea de la sección `[defaults]` en el archivo de configuración `/etc/ansible/ansible.cfg` para evitar la autenticación por contraseña:

```
#ask_pass      = True
```

3.5.2 Prueba de autenticación mediante clave privada

Para la siguiente prueba, se utiliza el módulo `shell` , permitiendo la ejecución de comandos remotos:

```
# ansible rocky8 -m shell -a "uptime"
172.16.1.10 | SUCCESS | rc=0 >>
 12:36:18 up 57 min,  1 user,  load average: 0.00, 0.00, 0.00

172.16.1.11 | SUCCESS | rc=0 >>
 12:37:07 up 57 min,  1 user,  load average: 0.00, 0.00, 0.00
```

No se necesita contraseña, ¡la autenticación mediante clave privada/pública funciona!

> **Note**
>
> En el entorno de producción, ahora debe eliminar las contraseñas `ansible` establecidas anteriormente para reforzar su seguridad (ya que ahora no es necesaria una contraseña de autenticación).

3.6 Utilizar Ansible

Puede utilizar Ansible desde el intérprete de comandos o a través de libros de jugadas.

3.6.1 Los módulos

La lista de módulos clasificados por categoría puede encontrarse aquí. ¡Ansible ofrece más de 750!

Los módulos se agrupan ahora en colecciones, cuya lista puede encontrarse aquí.

Las colecciones son un formato de distribución para contenido de Ansible que puede incluir playbooks, roles, módulos y plugins.

Un módulo se invoca con la opción `-m` del comando `ansible`:

```
ansible <host-pattern> [-m module_name] [-a args] [options]
```

Hay un módulo para casi cualquier necesidad. Por lo tanto, en lugar de utilizar el módulo shell, se recomienda buscar un módulo adaptado a esta necesidad.

Cada categoría de necesidad tiene su propio módulo. Aquí hay una lista no exhaustiva:

Tipo	Ejemplos
Administración del sistema	`user` (gestión de usuarios), `group` (gestión de grupos), etc.
Administración de software	`dnf`, `yum`, `apt`, `pip`, `npm`
Gestión de archivos	`copy`, `fetch`, `lineinfile`, `template`, `archive`
Administración de bases de datos	`mysql`, `postgresql`, `redis`
Gestión de la nube	`amazon S3`, `cloudstack`, `openstack`
Gestión del clúster	`consul`, `zookeeper`
Enviar comandos	`shell`, `script`, `expect`
Descargas	`get_url`
Gestión de código	`git`, `gitlab`

Ejemplo de instalación de software

El módulo `dnf` permite la instalación de software en los clientes de destino:

```
# ansible rocky8 --become -m dnf -a name="httpd"
172.16.1.10 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
      ...
      \n\nComplete!\n"
    ]
}
172.16.1.11 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
      ...
    \n\nComplete!\n"
    ]
}
```

Siendo el software instalado un servicio, ahora es necesario iniciarlo con el módulo `systemd`:

```
# ansible rocky8 --become  -m systemd -a "name=httpd state=started"
172.16.1.10 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
172.16.1.11 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
```

Tip

Intente ejecutar estos últimos dos comandos, dos veces. Observará que la primera vez, Ansible realizará acciones para alcanzar el estado establecido por el comando. La segunda vez, ¡no hará nada porque habrá detectado que ya se ha alcanzado el estado!

3.6.2 Ejercicios

Para ayudarle a descubrir más sobre Ansible y acostumbrarse a buscar en su documentació, aquí encontrará algunos ejercicios que puede hacer antes de continuar:

- Crear los grupos París, Tokio, NewYork
- Crear el usuario `supervisor`
- Cambia el usuario para que tenga un uid de 10000
- Cambiar el usuario para que pertenezca al grupo París
- Instalar el software tree
- Detener el servicio de crond
- Crear un archivo vacío con permisos `0644`
- Actualizar su distribución
- Reinciar su cliente

> **Warning**
>
> No utilice el módulo shell. ¡Busque en la documentación los módulos apropiados!

Modulo `setup`: Introducción a los hechos

Los datos del sistema son variables recuperadas por Ansible a través de su módulo `setup`.

Eche un vistazo a los diferentes datos de sus clientes para hacerse una idea de la cantidad de información que se puede recuperar fácilmente a través de un simple comando.

Más adelante veremos cómo utilizar los datos obtenidos desde los clientes en nuestros playbooks y cómo crear nuestros propios datos.

```
# ansible ansible_clients -m setup | less
192.168.1.11 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.1.11"
        ],
        "ansible_all_ipv6_addresses": [
            "2001:861:3dc3:fcf0:a00:27ff:fef7:28be",
            "fe80::a00:27ff:fef7:28be"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_vendor": "innotek GmbH",
        "ansible_bios_version": "VirtualBox",
        "ansible_board_asset_tag": "NA",
        "ansible_board_name": "VirtualBox",
        "ansible_board_serial": "NA",
        "ansible_board_vendor": "Oracle Corporation",
        ...
```

Ahora que hemos visto cómo configurar un servidor remoto con Ansible mediante la línea de comandos, podremos introducir el concepto de playbook. Los playbooks no son más que otra forma de utilizar Ansible, que no es mucho más compleja, pero que facilitará la reutilización del código.

3.7 Playbooks

Los playbooks de Ansible describen una política que se aplica a los sistemas remotos, para forzar su configuración. Los playbooks se escriben en un formato de texto fácilmente comprensible que agrupa un conjunto de tareas: el formato `yaml`.

> **Note**
>
> Aprenda más sobre yaml aquí

```
ansible-playbook <file.yml> ... [options]
```

Las opciones son idénticas a las del comando `ansible`.

El comando devuelve los siguientes códigos de error:

Código	Error
`0`	OK o no hay servidores coincidentes
`1`	Error
`2`	Uno o más servidores estan fallando
`3`	Uno o más servidores son inalcanzables
`4`	Error de análisis
`5`	Opciones malas o incompletas
`99`	Ejecución interrumpida por el usuario
`250`	Error inesperado

> **Note**
>
> ¡Tenga en cuenta que Ansible devolverá Ok cuando no haya ningún host que coincida con su objetivo, lo que podría inducirle a error!

3.7.1 Ejemplo de playbook de Apache y MySQL

El siguiente playbook nos permite instalar Apache y MariaDB en nuestros servidores de destino.

Cree un archivo `test.yml` con el siguiente contenido:

```
---
- hosts: rocky8 <1>
  become: true <2>
```

```
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      dnf: name=httpd,php,php-mysqli state=latest

    - name: ensure httpd is started
      systemd: name=httpd state=started

    - name: ensure mariadb is at the latest version
      dnf: name=mariadb-server state=latest

    - name: ensure mariadb is started
      systemd: name=mariadb state=started
...
```

- <1> El grupo o el servidor en cuestión debe existir en el inventario
- <2> Una vez conectado, el usuario se convierte `root` (a través de `sudo` por defecto)

La ejecución del playbook se realiza mediante el comando `ansible-playbook`:

```
$ ansible-playbook test.yml

PLAY [rocky8] ******************************************************************

TASK [setup] *******************************************************************
ok: [172.16.1.10]
ok: [172.16.1.11]

TASK [ensure apache is at the latest version] **********************************
ok: [172.16.1.10]
ok: [172.16.1.11]

TASK [ensure httpd is started] *************************************************
changed: [172.16.1.10]
changed: [172.16.1.11]

TASK [ensure mariadb is at the latest version]
*********************************
changed: [172.16.1.10]
changed: [172.16.1.11]

TASK [ensure mariadb is started]
*******************************************
```

```
changed: [172.16.1.10]
changed: [172.16.1.11]

PLAY RECAP
********************************************************************
172.16.1.10              : ok=5    changed=3    unreachable=0    failed=0
172.16.1.11              : ok=5    changed=3    unreachable=0    failed=0
```

Para una mayor legibilidad, se recomienda escribir los playbooks en formato yaml completo. En el ejemplo anterior, los argumentos se dan en la misma línea que el módulo, el valor del argumento siguiendo su nombre separado por un `=`. Mira el mismo playbook en yaml completo:

```
---
- hosts: rocky8
  become: true
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      dnf:
        name: httpd,php,php-mysqli
        state: latest

    - name: ensure httpd is started
      systemd:
        name: httpd
        state: started

    - name: ensure mariadb is at the latest version
      dnf:
        name: mariadb-server
        state: latest

    - name: ensure mariadb is started
      systemd:
        name: mariadb
        state: started
...
```

Tip

`dnf` es uno de los módulos que permiten pasarle una lista como argumento.

Nota sobre las colecciones: Ahora Ansible proporciona módulos en forma de colecciones. Algunos módulos se proporcionan por defecto dentro de la colección `ansible.builtin`, otros se deben instalar manualmente a través de la función:

```
ansible-galaxy collection install [collectionname]
```

donde [collectionname] es el nombre de la colección (aquí los corchetes se utilizan para resaltar la necesidad de reemplazar el texto contenido entre ellos por un nombre de colección real, NO son parte del comando).

El ejemplo anterior debería escribirse así:

```
- hosts: rocky8
  become: true
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      ansible.builtin.dnf:
        name: httpd,php,php-mysqli
        state: latest

    - name: ensure httpd is started
      ansible.builtin.systemd:
        name: httpd
        state: started

    - name: ensure mariadb is at the latest version
      ansible.builtin.dnf:
        name: mariadb-server
        state: latest

    - name: ensure mariadb is started
      ansible.builtin.systemd:
        name: mariadb
        state: started
...
```

Un playbook no se limita únicamente a un solo objetivo:

```
---
- hosts: webservers
  become: true
```

```
  become_user: root

  tasks:

    - name: ensure apache is at the latest version
      ansible.builtin.dnf:
        name: httpd,php,php-mysqli
        state: latest

    - name: ensure httpd is started
      ansible.builtin.systemd:
        name: httpd
        state: started

- hosts: databases
  become: true
  become_user: root

    - name: ensure mariadb is at the latest version
      ansible.builtin.dnf:
        name: mariadb-server
        state: latest

    - name: ensure mariadb is started
      ansible.builtin.systemd:
        name: mariadb
        state: started
...
```

Puede comprobar la sintaxis de su playbook:

```
$ ansible-playbook --syntax-check play.yml
```

También puedes utilizar un "linter" para el formato yaml:

```
$ dnf install -y yamllint
```

y despues comprobar la sintaxis yaml de sus playbooks:

```
$ yamllint test.yml
test.yml
  8:1       error    syntax error: could not find expected ':' (syntax)
```

3.8 Resultados de los ejercicios

- Crear los grupos París, Tokio, NewYork
- Crear el usuario `supervisor`
- Cambiar el usuario para que tenga un uid de 10000
- Cambiar el usuario para que pertenezca al grupo París
- Instalar el software tree
- Detener el servicio de crond
- Crear un archivo vacío con permisos `0644`
- Actualizar su distribución
- Reinciar su cliente

```
ansible ansible_clients --become -m group -a "name=Paris"
ansible ansible_clients --become -m group -a "name=Tokio"
ansible ansible_clients --become -m group -a "name=NewYork"
ansible ansible_clients --become -m user -a "name=Supervisor"
ansible ansible_clients --become -m user -a "name=Supervisor uid=10000"
ansible ansible_clients --become -m user -a "name=Supervisor uid=10000
groups=Paris"
ansible ansible_clients --become -m dnf -a "name=tree"
ansible ansible_clients --become -m systemd -a "name=crond state=stopped"
ansible ansible_clients --become -m copy -a "content='' dest=/tmp/test force=no
mode=0644"
ansible ansible_clients --become -m dnf -a "name=* state=latest"
ansible ansible_clients --become -m reboot
```

4. Ansible Intermedio

En este capítulo seguirá aprendiendo a trabajar con Ansible.

Objetivos : En este capítulo aprenderá a:

✔ trabajar con variables;
✔ utilizar loops;
✔ gestionar cambios de estado y reaccionar ante ellos;
✔ gestionar tareas asíncronas.

🏁 **ansible**, **modulo**, **playbook**

Conocimiento: ⭐⭐⭐
Complejidad: ⭐⭐

Tiempo de lectura: 30 minutos

En el capítulo anterior, aprendió a instalar Ansible, a utilizar en la línea de comandos o a escribir libros de jugadas para promover la reutilización de tu código.

En este capítulo, podemos empezar a descubrir algunas nociones más avanzadas sobre cómo utilizar Ansible, y descubrir algunas tareas interesantes que utilizará regularmente.

4.1 Las variables

> **Note**
>
> Puede encontrar más información aquí.

En Ansible, hay diferentes tipos de variables primitivas:

- Cadenas de texto.
- Números enteros.
- Valores booleanos.

Estas variables pueden organizarse como:

- Diccionarios.
- Listas.

Una variable se puede definit en diferentes lugares, como un libro de jugadas, un rol o desde la línea de comandos por ejemplo.

Por ejemplo, desde un libro de jugadas:

```
---
- hosts: apache1
  vars:
    port_http: 80
    service:
      debian: apache2
      rhel: httpd
```

o desde la línea de comandos:

```
$ ansible-playbook deploy-http.yml --extra-vars "service=httpd"
```

Una vez definida, una variable puede utilizarse llamándola entre llaves dobles:

- `{{ port_http }}` fpara un valor simple,
- `{{ service['rhel'] }}` or `{{ service.rhel }}` para un diccionario.

Por ejemplo:

```
- name: make sure apache is started
  ansible.builtin.systemd:
    name: "{{ service['rhel'] }}"
    state: started
```

Por supuesto, también es posible acceder a las variables globales (los **hechos**) de Ansible (tipo de SO, direcciones IP, nombre de la VM, etc.).

4.1.1 Externalización de variables

Las variables se pueden incluir en un archivo externo al libro de jugadas, en cuyo caso este archivo debe definirse en el libro de jugadas mediante la directiva `vars_files`:

```
---
- hosts: apache1
  vars_files:
    - myvariables.yml
```

El archivo `myvariables.yml`:

```
---
port_http: 80
ansible.builtin.systemd::
  debian: apache2
  rhel: httpd
```

También se pueden añadir dinámicamente mediante el uso del módulo `include_vars`:

```
- name: Include secrets.
  ansible.builtin.include_vars:
    file: vault.yml
```

4.1.2 Mostrar una variable

Para visualizar una variable, hay que activar el módulo `debug` de la siguiente manera:

```
- ansible.builtin.debug:
    var: "{{ service['debian'] }}"
```

También puede utilizar la variable dentro de un texto:

```
- ansible.builtin.debug:
    msg: "Print a variable in a message : {{ service['debian'] }}"
```

4.1.3 Guardar el retorno de una tarea

Para guardar el retorno de una tarea y poder acceder a ella posteriormente, hay que utilizar la palabra clave `register` dentro de la propia tarea.

Utilización de una variable almacenada:

```
- name: /home content
  shell: ls /home
  register: homes

- name: Print the first directory name
  ansible.builtin.debug:
    var: homes.stdout_lines[0]

- name: Print the first directory name
  ansible.builtin.debug:
    var: homes.stdout_lines[1]
```

Note

La variable `homes.stdout_lines` es una lista de variables de tipo string, una forma de organizar las variables que aún no habíamos encontrado.

Se puede acceder a las cadenas que componen la variable almacenada a través del valor `stdout` (lo que permite hacer cosas como `homes.stdout.find("core") != -1`), explotarlas mediante un bucle (ver `loop`), o simplemente por sus índices como se ha visto en el ejemplo anterior.

4.1.4 Ejercicios

- Escriba un libro de jugadas `play-vars.yml` que imprima el nombre de distribución del objetivo con su versión principal, utilizando variables globales.
- Escriba un libro de jugadas utilizando el siguiente diccionario para mostrar los servicios que se instalarán:

```
service:
  web:
    name: apache
    rpm: httpd
  db:
    name: mariadb
    rpm: mariadb-server
```

El tipo por defecto debe ser "web".

- Anular la variable `type` utilizando la línea de comandos
- Externalizar variables en un archivo `vars.yml`

4.2 Gestion de los bucles

Con la ayuda de los bucles, puede iterar una tarea sobre una lista, un hash, o diccionario, por ejemplo.

> **Note**
>
> Puede encontrar más información aquí.

Ejemplo sencillo de uso, creación de 4 usuarios:

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  loop:
     - antoine
     - patrick
     - steven
     - xavier
```

En cada iteración del bucle, el valor de la lista utilizada se almacena en la variable `item`, accesible en el código del bucle.

Por supuesto, se puede definir una lista dentro de un archivo externo:

```
users:
  - antoine
  - patrick
  - steven
  - xavier
```

y utilizarla dentro de la tarea como ésta (después de haber incluido el archivo de variables):

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  loop: "{{ users }}"
```

Podemos utilizar el ejemplo visto durante el estudio de variables almacenadas para mejorarlo. Utilización de una variable almacenada:

```
- name: /home content
  shell: ls /home
  register: homes

- name: Print the directories name
  ansible.builtin.debug:
    msg: "Directory => {{ item }}"
  loop: "{{ homes.stdout_lines }}"
```

Un diccionario también se puede utilizar en un bucle.

En este caso, tendrá que transformar el diccionario en un elemento con lo que se llama **filtro jinja** (jinja es el motor de plantillas utilizado por Ansible): `| dict2items`.

En el bucle, es posible usar `item.key` que corresponde a la clave del diccionario, y `item.value` que corresponde a los valores de la clave.

Veámoslo a través de un ejemplo concreto, mostrando la gestión de los usuarios del sistema:

```
---
- hosts: rocky8
```

```
  become: true
  become_user: root
  vars:
    users:
      antoine:
        group: users
        state: present
      steven:
        group: users
        state: absent

  tasks:

  - name: Manage users
    user:
      name: "{{ item.key }}"
      group: "{{ item.value.group }}"
      state: "{{ item.value.state }}"
    loop: "{{ users | dict2items }}"
```

Note

Se pueden hacer muchas cosas con los bucles. Descubrirá las posibilidades que ofrecen los bucles cuando su uso de Ansible le empuja a utilizarlos de una manera más compleja.

4.2.1 Ejercicios

- Mostrar el contenido de la variable `servicie` del ejercicio anterior utilizando un bucle.

Note

Tendrá que transformar su variable `service`, la cual es un diccionario, a una lista con la ayuda del filtro de jinja `list` como este:

```
{{ service.values() | list }}
```

4.3 Condicionales

Note

Puede encontrar más información aquí.

La sentencia `when` es muy útil en muchos casos: no realizar ciertas acciones en determinados tipos de servidores, si un fichero o un usuario no existe, etc.

Note

Detrás de la sentencia `when` las variables no necesitan dobles llaves (de hecho son expresiones Jinja2...).

```
- name: "Reboot only Debian servers"
  reboot:
  when: ansible_os_family == "Debian"
```

Las condiciones se pueden agrupar con paréntesis:

```
- name: "Reboot only CentOS version 6 and Debian version 7"
  reboot:
  when: (ansible_distribution == "CentOS" and
ansible_distribution_major_version == "6") or
        (ansible_distribution == "Debian" and
ansible_distribution_major_version == "7")
```

Las condiciones correspondientes a una lógica AND se pueden proporcionar como una lista:

```
- name: "Reboot only CentOS version 6"
  reboot:
  when:
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "6"
```

Puede comprobar el valor de un booleano y verificar que si es verdadero:

```
- name: check if directory exists
  stat:
    path: /home/ansible
  register: directory

- ansible.builtin.debug:
    var: directory

- ansible.builtin.debug:
    msg: The directory exists
  when:
    - directory.stat.exists
    - directory.stat.isdir
```

Tambien puede comprobar si no es verdadero:

```
when:
  - file.stat.exists
  - not file.stat.isdir
```

Probablemente tendrá que comprobar que existe una variable para evitar errores de ejecución:

```
when: myboolean is defined and myboolean
```

4.3.1 Ejercicios

- Imprima el valor de `service.web` solo cuando `type` sea igual a `web`.

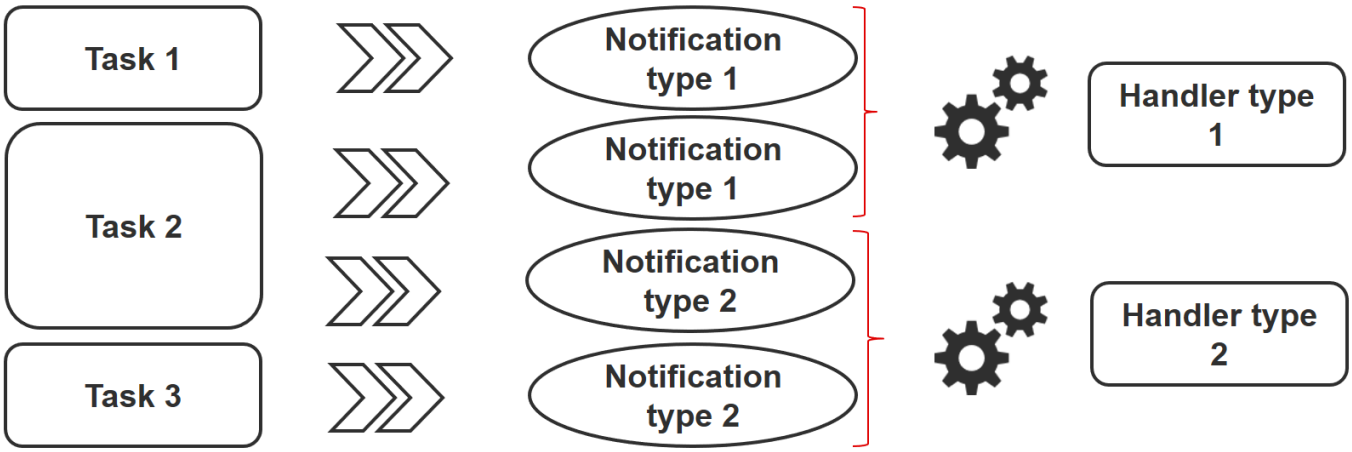
4.4 Gestionar los cambios: los `manejadores`

Note

Puede encontrar más información aquí.

Los manejadores permiten iniciar operaciones, como reiniciar un servicio, cuando ocurren cambios.

Un módulo, al ser idempotente, un libro de jugadas puede detectar que se ha producido un cambio significativo en un sistema remoto, y así desencadenar una operación en reacción a este cambio. Se envía una notificación al final de un bloque de tareas de un libro de jugadas, y la operación de reacción se desencadenará una sola vez aunque varias tareas envíen la misma notificación.



Por ejemplo, varias tareas pueden indicar que el servicio `httpd` necesita reiniciarse debido a un cambio en sus archivos de configuración. Pero el servicio sólo se reiniciará una vez para evitar múltiples inicios innecesarios.

```
- name: template configuration file
  template:
    src: template-site.j2
    dest: /etc/httpd/sites-availables/test-site.conf
  notify:
     - restart memcached
     - restart httpd
```

Un manejador es un tipo de tarea referenciada por un nombre global único:

- Se activa por una o más notificaciones.
- No se inicia inmediatamente, pero espera hasta que todas las tareas estén completas para ejecutarse.

Ejemplo de manejadores:

```
handlers:

  - name: restart memcached
    systemd:
      name: memcached
      state: restarted

  - name: restart httpd
    systemd:
      name: httpd
      state: restarted
```

Desde la versión 2.2 de Ansible, los manejadores también pueden escuchar directamente:

```
handlers:

  - name: restart memcached
    systemd:
      name: memcached
      state: restarted
    listen: "web services restart"

  - name: restart apache
    systemd:
      name: apache
      state: restarted
    listen: "web services restart"

tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "web services restart"
```

4.5 Tareas asíncronas

Note

Puede encontrar más información aquí.

Por defecto, las conexiones SSH a los hosts permanecen abiertas durante la ejecución de varias tareas en todos los nodos.

Esto puede provocar algunos problemas, especialmente:

- Si el tiempo de ejecución de la tarea es más largo que el tiempo de espera de la conexión SSH.
- Si la conexión es interrumpida durante la acción (por ejemplo por el reinicio de un servidor).

En este caso, tendrá que cambiar al modo asíncrono y especificar un tiempo máximo de ejecución así como la frecuencia (por defecto 10s) con la que comprobará el estado del host.

Al especificar un valor de 0, Ansible ejecutará la tarea y continuará sin preocuparse por el resultado.

Aquí hay un ejemplo que utiliza tareas asincrónicas, que permite reiniciar un servidor y esperar a que el puerto 22 esté accesible de nuevo:

```
# Wait 2s and launch the reboot
- name: Reboot system
  shell: sleep 2 && shutdown -r now "Ansible reboot triggered"
  async: 1
  poll: 0
  ignore_errors: true
  become: true
  changed_when: False

  # Wait the server is available
  - name: Waiting for server to restart (10 mins max)
    wait_for:
      host: "{{ inventory_hostname }}"
      port: 22
      delay: 30
      state: started
      timeout: 600
    delegate_to: localhost
```

También puede decidir si lanza una tarea de larga ejecución y olvidarse de ella (fire and forget) porque la ejecución no es relevante en el playbook.

4.6 Resultados de los ejercicios

- Escriba un libro de jugadas `play-vars.yml` que imprima el nombre de la distribución de los objetivos con su versión principal, utilizando variables globales.

```
---
- hosts: ansible_clients

  tasks:

    - name: Print globales variables
      debug:
```

```
        msg: "The distribution is {{ ansible_distribution }} version
{{ ansible_distribution_major_version }}"
```

```
$ ansible-playbook play-vars.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print globales variables]
*********************************************************************
ok: [192.168.1.11] => {
    "msg": "The distribution is Rocky version 8"
}

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Escriba un libro de jugadas utilizando el siguiente diccionario para mostrar los servicios que se instalarán:

```
service:
  web:
    name: apache
    rpm: httpd
  db:
    name: mariadb
    rpm: mariadb-server
```

El tipo por defecto debe ser "web".

```
---
- hosts: ansible_clients
  vars:
    type: web
    service:
      web:
        name: apache
        rpm: httpd
      db:
```

```
        name: mariadb
        rpm: mariadb-server

  tasks:

    - name: Print a specific entry of a dictionary
      debug:
        msg: "The {{ service[type]['name'] }} will be installed with the
packages {{ service[type].rpm }}"
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print a specific entry of a dictionnaire]
*******************************************************
ok: [192.168.1.11] => {
    "msg": "The apache will be installed with the packages httpd"
}

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Anular la variable `type` utilizando la línea de comandos:

```
ansible-playbook --extra-vars "type=db" display-dict.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print a specific entry of a dictionary]
*******************************************************
ok: [192.168.1.11] => {
    "msg": "The mariadb will be installed with the packages mariadb-server"
}
```

```
PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Externalizar variables en un archivo `vars.yml`

```
type: web
service:
  web:
    name: apache
    rpm: httpd
  db:
    name: mariadb
    rpm: mariadb-server
```

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:

    - name: Print a specific entry of a dictionary
      debug:
        msg: "The {{ service[type]['name'] }} will be installed with the
packages {{ service[type].rpm }}"
```

- Mostrar el contenido de la variable `service` del ejercicio anterior utilizando un bucle.

Note

Tendrá que transformar su variable `service`, que es un diccionario, en un elemento o una lista con la ayuda de los filtros jinja `dict2items` o `list` como este:

```
{{ service | dict2items }}
```

```
{{ service.values() | list }}
```

Con `dict2items`:

```
---
- hosts: ansible_clients
```

```
  vars_files:
    - vars.yml

  tasks:

    - name: Print a dictionary variable with a loop
      debug:
        msg: "{{item.key }} | The {{ item.value.name }} will be installed with
the packages {{ item.value.rpm }}"
      loop: "{{ service | dict2items }}"
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print a dictionary variable with a loop]
*************************************************
ok: [192.168.1.11] => (item={'key': 'web', 'value': {'name': 'apache', 'rpm':
'httpd'}}) => {
    "msg": "web | The apache will be installed with the packages httpd"
}
ok: [192.168.1.11] => (item={'key': 'db', 'value': {'name': 'mariadb', 'rpm':
'mariadb-server'}}) => {
    "msg": "db | The mariadb will be installed with the packages mariadb-
server"
}

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Con `list`:

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:
```

```
    - name: Print a dictionary variable with a loop
      debug:
        msg: "The {{ item.name }} will be installed with the packages
{{ item.rpm }}"
      loop: "{{ service.values() | list}}"
~
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print a dictionary variable with a loop]
*************************************************
ok: [192.168.1.11] => (item={'name': 'apache', 'rpm': 'httpd'}) => {
    "msg": "The apache will be installed with the packages httpd"
}
ok: [192.168.1.11] => (item={'name': 'mariadb', 'rpm': 'mariadb-server'}) => {
    "msg": "The mariadb will be installed with the packages mariadb-server"
}

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

- Imprime el valor de `service.web` solo cuando `type` sea igual a `web`.

```
---
- hosts: ansible_clients
  vars_files:
    - vars.yml

  tasks:

    - name: Print a dictionary variable
      debug:
        msg: "The {{ service.web.name }} will be installed with the packages
{{ service.web.rpm }}"
      when: type == "web"
```

```
    - name: Print a dictionary variable
      debug:
        msg: "The {{ service.db.name }} will be installed with the packages
{{ service.db.rpm }}"
      when: type == "db"
```

```
$ ansible-playbook display-dict.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print a dictionary variable]
*********************************************************************
ok: [192.168.1.11] => {
    "msg": "The apache will be installed with the packages httpd"
}

TASK [Print a dictionary variable]
*********************************************************************
skipping: [192.168.1.11]

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0

$ ansible-playbook --extra-vars "type=db" display-dict.yml

PLAY [ansible_clients]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Print a dictionary variable]
*********************************************************************
skipping: [192.168.1.11]

TASK [Print a dictionary variable]
*********************************************************************
ok: [192.168.1.11] => {
    "msg": "The mariadb will be installed with the packages mariadb-server"
}
```

```
PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

5. Ansible - Gestión de archivos

En este capítulo aprenderás a gestionar archivos con Ansible.

Objetivos : En este capítulo aprenderá a:

✔ modificar el contenido de un archivo;
✔ subir archivos a los servidores de destino;
✔ descargar archivos desde los servidores de destino.

🏁 **ansible**, **module**, **files**

Conocimiento: ⭐ ⭐
Complejidad: ⭐

Tiempo de lectura: 20 minutos

Dependiendo de sus necesidades, tendrá que utilizar módulos de Ansible diferentes para modificar los archivos de configuración del sistema.

5.1 Módulo `ini_file`

Cuando quiera modificar un archivo INI (la sección entre `[]` y los pares `clave=valor`), la forma más fácil es utilizar el módulo `ini_file`.

> **Note**
>
> Puede encontrar más información aquí.

El módulo requiere:

- El valor de la sección
- El nombre de la opción
- El nuevo valor

Ejemplo de uso:

```
- name: change value on inifile
  community.general.ini_file:
    dest: /path/to/file.ini
    section: SECTIONNAME
    option: OPTIONNAME
    value: NEWVALUE
```

5.2 Módulo `lineinfile`

Para asegurarse de que una línea está presente en un archivo, o cuando se necesita añadir o modificar una sola línea en un archivo, utilice el módulo `linefile`.

Note

Puede encontrar más información aquí.

En este caso, la línea a modificar en un archivo se encontrará mediante una regexp.

Por ejemplo, para garantizar que la línea que comienza con `SELINUX=` en el archivo `/etc/selinux/config` contiene el valor `enforcing`:

```
- ansible.builtin.lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=enforcing'
```

5.3 Módulo `copy`

Cuando hay que copiar un archivo desde el servidor Ansible a uno o más hosts, es mejor utilizar el módulo `copy`.

Note

Puede encontrar más información aquí.

Aquí estamos copiando el archivo `myflile.conf` de una ubicación a otra:

```
- ansible.builtin.copy:
    src: /data/ansible/sources/myfile.conf
    dest: /etc/myfile.conf
```

```
    owner: root
    group: root
    mode: 0644
```

5.4 Módulo `fetch`

Cuando hay que copiar un archivo de un servidor remoto a un servidor local, lo mejor es utilizar el módulo `fetch`.

> **Note**
>
> Puede encontrar más información aquí.

Este módulo hace lo contrario que el módulo `copy`:

```
- ansible.builtin.fetch:
    src: /etc/myfile.conf
    dest: /data/ansible/backup/myfile-{{ inventory_hostname }}.conf
    flat: yes
```

5.5 Módulo `template`

Ansible y su módulo `template` utilizan el sistema de plantillas **Jinja2** (http://jinja.pocoo.org/docs/) para generar archivos en los hosts de destino.

> **Note**
>
> Puede encontrar más información aquí.

Por ejemplo:

```
- ansible.builtin.template:
    src: /data/ansible/templates/monfichier.j2
    dest: /etc/myfile.conf
    owner: root
    group: root
    mode: 0644
```

Es posible añadir un paso de validación si el servicio de destino lo permite (por ejemplo Apache con el comando `apachectl -t`):

```
- template:
    src: /data/ansible/templates/vhost.j2
    dest: /etc/httpd/sites-available/vhost.conf
    owner: root
    group: root
    mode: 0644
    validate: '/usr/sbin/apachectl -t'
```

5.6 Módulo `get_url`

Para subir archivos desde un sitio web o ftp a uno o más hosts, utilice el módulo `get_url`:

```
- get_url:
    url: http://site.com/archive.zip
    dest: /tmp/archive.zip
    mode: 0640
    checksum:
sha256:f772bd36185515581aa9a2e4b38fb97940ff28764900ba708e68286121770e9a
```

Al proporcionar una suma de comprobación del archivo, éste no se volverá a descargar si ya está presente en la ubicación de destino y su suma de comprobación coincide con el valor proporcionado.

6. Ansible Galaxy: Colecciones y roles

En este capítulo aprenderá a utilizar, instalar y gestionar los roles y colecciones de Ansible.

Objetivos : En este capítulo aprenderá a:

✔ instalar y gestionar colecciones;
✔ instalar y gestionar roles;.

🏁 **ansible**, **ansible-galaxy**, **roles**, **colecciones**

Conocimiento: ⭐ ⭐
Complejidad: ⭐ ⭐ ⭐

Tiempo de lectura: 40 minutos

Ansible Galaxy proporciona roles y colecciones de Ansible desde la comunidad Ansible.

Los elementos proporcionados se pueden referenciar en los playbooks y ser utilizados tal y como están

6.1 Comando `ansible-galaxy`

El comando `ansible-galaxy` gestiona los roles y las colecciones utilizando galaxy.ansible.com.

- Para gestionar los roles:

```
ansible-galaxy role [import|init|install|login|remove|...]
```

Sub-comandos	Observaciones
`install`	instalar un rol.
`remove`	elimina uno o más roles.
`lista`	muestra el nombre y la versión de los roles instalados.
`info`	muestra la información sobre un rol.
`init`	genera la estructura básica de un nuevo rol.
`import`	importa un rol desde el sitio web de Ansible Galaxy. Requiere inicio de sesión.

- Para gestionar las colecciones:

```
ansible-galaxy collection [import|init|install|login|remove|...]
```

Sub-comandos	Observaciones
`init`	genera la estructura básica de una nueva colección.
`install`	instala una colección.
`list`	muestra el nombre y la versión de las colecciones instaladas.

6.2 Roles de Ansible

Un rol de Ansible es una unidad que promueve la reutilización de los playbooks.

Note

Puede encontrar más información aquí

6.2.1 Instalación de roles útiles

Para resaltar el interés del uso de roles, le sugiero que utilice el rol `alemorvan/patchmanagement`, que le permitirá realizar un montón de tareas (preactualización o postactualización, por ejemplo) durante su proceso de actualización, en sólo unas pocas líneas de código.

Puedes consultar el código en el repo de github del rol [aquí](https://github. com/alemorvan/patchmanagement).

- Instalar el rol. Para ello, sólo es necesario un comando:

```
ansible-galaxy role install alemorvan.patchmanagement
```

- Cree un playbook para incluir el rol:

```
- name: Start a Patch Management
  hosts: ansible_clients
  vars:
    pm_before_update_tasks_file: custom_tasks/pm_before_update_tasks_file.yml
    pm_after_update_tasks_file: custom_tasks/pm_after_update_tasks_file.yml

  tasks:
    - name: "Include patchmanagement"
      include_role:
        name: "alemorvan.patchmanagement"
```

Con este rol, puede añadir sus propias tareas para todo su inventario o sólo para su nodo objetivo.

Vamos a crear tareas que se ejecutarán antes y después del proceso de actualización:

- Cree la carpeta `custom_tasks`:

```
mkdir custom_tasks
```

- Cree el archivo `custom_tasks/pm_before_update_tasks_file.yml` (siéntase libre de cambiar el nombre y el contenido de este archivo)

```
---
- name: sample task before the update process
  debug:
    msg: "This is a sample tasks, feel free to add your own test task"
```

- Cree el archivo `custom_tasks/pm_after_update_tasks_file.yml` (siéntase libre de cambiar el nombre y el contenido de este archivo)

```
---
- name: sample task after the update process
  debug:
    msg: "This is a sample tasks, feel free to add your own test task"
```

Y lance su primera gestión de parches:

```
ansible-playbook patchmanagement.yml

PLAY [Start a Patch Management]
*********************************************************************

TASK [Gathering Facts]
*****************************************************************************************
ok: [192.168.1.11]

TASK [Include patchmanagement]
*********************************************************************

TASK [alemorvan.patchmanagement : MAIN | Linux Patch Management Job]
***********************************
ok: [192.168.1.11] => {
    "msg": "Start 192 patch management"
}

...

TASK [alemorvan.patchmanagement : sample task before the update process]
*******************************
ok: [192.168.1.11] => {
    "msg": "This is a sample tasks, feel free to add your own test task"
}

...

TASK [alemorvan.patchmanagement : MAIN | We can now patch]
*******************************************
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
patch.yml for 192.168.1.11

TASK [alemorvan.patchmanagement : PATCH | Tasks depends on distribution]
*******************************
ok: [192.168.1.11] => {
    "ansible_distribution": "Rocky"
}

TASK [alemorvan.patchmanagement : PATCH | Include tasks for CentOS & RedHat
```

```
tasks] **********************
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
linux_tasks/redhat_centos.yml for 192.168.1.11

TASK [alemorvan.patchmanagement : RHEL CENTOS | yum clean all]
****************************************
changed: [192.168.1.11]

TASK [alemorvan.patchmanagement : RHEL CENTOS | Ensure yum-utils is installed]
*************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : RHEL CENTOS | Remove old kernels]
***********************************
skipping: [192.168.1.11]

TASK [alemorvan.patchmanagement : RHEL CENTOS | Update rpm package with yum]
*****************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : PATCH | Inlude tasks for Debian & Ubuntu
tasks] ***********************
skipping: [192.168.1.11]

TASK [alemorvan.patchmanagement : MAIN | We can now reboot]
*****************************************
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
reboot.yml for 192.168.1.11

TASK [alemorvan.patchmanagement : REBOOT | Reboot triggered]
*****************************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : REBOOT | Ensure we are not in rescue mode]
*****************************
ok: [192.168.1.11]

...

TASK [alemorvan.patchmanagement : FACTS | Insert fact file]
*******************************************
ok: [192.168.1.11]

TASK [alemorvan.patchmanagement : FACTS | Save date of last PM]
***************************************
ok: [192.168.1.11]

...
```

```
TASK [alemorvan.patchmanagement : sample task after the update process]
*******************************
ok: [192.168.1.11] => {
    "msg": "This is a sample tasks, feel free to add your own test task"
}

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=31   changed=1    unreachable=0    failed=0
skipped=4    rescued=0    ignored=0
```

Bastante fácil para un proceso tan complejo, ¿no?

Este es sólo un ejemplo de lo que se puede hacer utilizando los roles puestos a disposición por la comunidad. Eche un vistazo a galaxy.ansible.com para descubrir las funciones que podrían serle útiles.

También puede crear sus propios roles para cubrir sus propias necesidades y publicarlos en Internet si le apetece. Lo que trataremos brevemente en el próximo capítulo.

6.2.2 Introducción al desarrollo de roles

La estructura básica de un rol, sirve como punto de partida para el desarrollo de roles personalizados, puede ser generada mediante el comando `ansible-galaxy`:

```
$ ansible-galaxy role init rocky8
- Role rocky8 was created successfully
```

El comando generará la siguiente estructura de árbol para contener el rol `rocky8`:

```
tree rocky8/
rocky8/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
```

```
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

Los roles le permiten prescindir de la necesidad de incluir archivos. No es necesario especificar rutas de archivos o directivas `include` en los playbooks. Sólo tiene que especificar una tarea, y Ansible se encarga de las inclusiones.

La estructura de un rol es bastante obvia de entender.

Simplemente las variables se almacenan en `vars/main.yml` si las variables no van a ser sobreescribir, o en `default/main.yml` si quiere dejar la posibilidad de sobreescribir el contenido de las variables desde fuera de su rol.

Los handlers, archivos y plantillas necesarios para su código se almacenan en `handlers/main.yml`, `files` y `templates` respectivamente.

Sólo queda definir el código de las tareas de su rol en `tasks/main.yml`.

Una vez que todo esto funciona correctamente, puede utilizar este rol en sus playbooks. Podrá utilizar su rol sin preocuparse por el aspecto técnico de sus tareas, mientras personaliza su funcionamiento con variables.

6.2.3 Trabajo práctico: crear un primer rol sencillo

Implementemos esto con un rol que creará un usuario por defecto e instalará paquetes de software. Este rol puede aplicarse sistemáticamente a todos sus servidores.

Variables

Crearemos un usuario `rockstar` en todos nuestros servidores. Como no queremos que este usuario sea reemplazado, vamos a definirlo en el `vars/main.yml`:

```
---
rocky8_default_group:
```

```
  name: rockstar
  gid: 1100
rocky8_default_user:
  name: rockstar
  uid: 1100
  group: rockstar
```

Ahora podemos usar esas variables dentro de nuestro archivo `tasks/main.yml` sin ninguna inclusión.

```
---
- name: Create default group
  group:
    name: "{{ rocky8_default_group.name }}"
    gid: "{{ rocky8_default_group.gid }}"

- name: Create default user
  user:
    name: "{{ rocky8_default_user.name }}"
    uid: "{{ rocky8_default_user.uid }}"
    group: "{{ rocky8_default_user.group }}"
```

Para probar su nuevo rol, vamos a crear un playbook `test-role.yml` en el mismo directorio que su rol:

```
---
- name: Test my role
  hosts: localhost

  roles:

    - role: rocky8
      become: true
      become_user: root
```

y ejecútelo:

```
ansible-playbook test-role.yml

PLAY [Test my role]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
```

```
ok: [localhost]

TASK [rocky8 : Create default group]
*********************************************************************
changed: [localhost]

TASK [rocky8 : Create default user]
*********************************************************************
changed: [localhost]

PLAY RECAP
*********************************************************************************************
localhost                  : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

¡Felicidades! Ahora es capaz de hacer grandes cosas con un playbook de unas pocas líneas.

Veamos el uso de las variables por defecto.

Cree una lista de paquetes para instalar por defecto en sus servidores y una lista vacía de paquetes para desinstalar. Edite los archivos `defaults/main.yml` y añada esas dos listas:

```
rocky8_default_packages:
  - tree
  - vim
rocky8_remove_packages: []
```

y utilícelos en tu archivo `tasks/main.yml`:

```
- name: Install default packages (can be overridden)
  package:
    name: "{{ rocky8_default_packages }}"
    state: present

- name: "Uninstall default packages (can be overridden)
{{ rocky8_remove_packages }}"
  package:
    name: "{{ rocky8_remove_packages }}"
    state: absent
```

Pruebe su rol con la ayuda del playbook creado anteriormente:

```
ansible-playbook test-role.yml

PLAY [Test my role]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [localhost]

TASK [rocky8 : Create default group]
*****************************************************************
ok: [localhost]

TASK [rocky8 : Create default user]
*****************************************************************
ok: [localhost]

TASK [rocky8 : Install default packages (can be overridden)]
*****************************************
ok: [localhost]

TASK [rocky8 : Uninstall default packages (can be overridden) []]
**************************************
ok: [localhost]

PLAY RECAP
*********************************************************************************
localhost                  : ok=5    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Ahora puede anular el valor de la variable `rocky8_remove_packages` en su playbook y desinstalar por ejemplo `cockpit`:

```
---
- name: Test my role
  hosts: localhost
  vars:
    rocky8_remove_packages:
      - cockpit

  roles:

    - role: rocky8
      become: true
      become_user: root
```

```
ansible-playbook test-role.yml

PLAY [Test my role]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [localhost]

TASK [rocky8 : Create default group]
*****************************************************************
ok: [localhost]

TASK [rocky8 : Create default user]
*****************************************************************
ok: [localhost]

TASK [rocky8 : Install default packages (can be overridden)]
*****************************************
ok: [localhost]

TASK [rocky8 : Uninstall default packages (can be overridden) ['cockpit']]
*****************************
changed: [localhost]

PLAY RECAP
*********************************************************************************
localhost                  : ok=5    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Evidentemente, no hay límite a la hora de mejorar su rol. Imagine que para uno de sus servidores, necesita un paquete que está en la lista de los que hay que desinstalar. A continuación, podría, por ejemplo, crear una nueva lista que puede ser anulada y luego eliminar de la lista de paquetes a desinstalar los de la lista de paquetes específicos a instalar mediante el filtro de jinja `difference()`.

```
- name: "Uninstall default packages (can be overridden)
{{ rocky8_remove_packages }}"
  package:
    name: "{{ rocky8_remove_packages |
difference(rocky8_specifics_packages) }}"
    state: absent
```

6.3 Colecciones de Ansible

Las colecciones son un formato de distribución para contenido de Ansible que puede incluir playbooks, roles, módulos y plugins.

> **Note**
>
> Puede encontrar más información aquí

Para instalar o actualizar una colección:

```
ansible-galaxy collection install namespace.collection [--upgrade]
```

A continuación, puede utilizar la colección recién instalada utilizando su espacio de nombres y su nombre antes del nombre del módulo o del rol:

```
- import_role:
    name: namespace.collection.rolename

- namespace.collection.modulename:
    option1: value
```

Puede encontrar un índice de la colección aquí.

Vamos a instalar la colección `community.general`:

```
ansible-galaxy collection install community.general
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/community-general-3.3.2.tar.gz
to /home/ansible/.ansible/tmp/ansible-local-51384hsuhf3t5/tmpr_c9qrt1/
community-general-3.3.2-f4q9u4dg
Installing 'community.general:3.3.2' to '/home/ansible/.ansible/collections/
ansible_collections/community/general'
community.general:3.3.2 was installed successfully
```

Ahora podemos utilizar el nuevo módulo disponible `yum_versionlock`:

```
- name: Start a Patch Management
  hosts: ansible_clients
  become: true
  become_user: root
```

```
  tasks:

    - name: Ensure yum-versionlock is installed
      package:
        name: python3-dnf-plugin-versionlock
        state: present

    - name: Prevent kernel from being updated
      community.general.yum_versionlock:
        state: present
        name: kernel
      register: locks

    - name: Display locks
      debug:
        var: locks.meta.packages
```

```
ansible-playbook versionlock.yml

PLAY [Start a Patch Management]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [192.168.1.11]

TASK [Ensure yum-versionlock is installed]
*************************************************************
changed: [192.168.1.11]

TASK [Prevent kernel from being updated]
*************************************************************
changed: [192.168.1.11]

TASK [Display locks]
*********************************************************************************
ok: [192.168.1.11] => {
    "locks.meta.packages": [
        "kernel"
    ]
}

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

6.3.1 Crea su propia colección

Al igual que con los roles, puede crear su propia colección con la ayuda del comando `ansible-galaxy`:

```
ansible-galaxy collection init rocky8.rockstarcollection
- Collection rocky8.rockstarcollection was created successfully
```

```
tree rocky8/rockstarcollection/
rocky8/rockstarcollection/
├── docs
├── galaxy.yml
├── plugins
│ └── README.md
├── README.md
└── roles
```

A continuación, puede almacenar sus propios plugins o roles dentro de esta nueva colección.

7. Despliegues de Ansible con Ansistrano

En este capítulo aprenderá a desplegar aplicaciones con el rol de Ansible Ansistrano.

Objetivos : En este capítulo aprenderá a:

✔ Implementar Ansistrano;
✔ Configurar Ansistrano;
✔ Utilizar carpetas compartidas y ficheros entre las versiones deplgadas;
✔ Desplegar versiones diferentes de un sitio web desde Git;
✔ Reaccionar entre los pasos de los despliegues.

🏁 **ansible**, **ansistrano**, **roles**, **despliegues**

Conocimiento: ⭐ ⭐
Complejidad: ⭐ ⭐ ⭐

Tiempo de lectura: 40 minutos

Ansistrano es un rol de Ansible para desplegar fácilmente aplicaciones PHP, Python, etc. Se basa en la funcionalidad de Capistrano.

7.1 Introducción

Ansistrano requiere las siguientes piezas para funcionar correctamente:

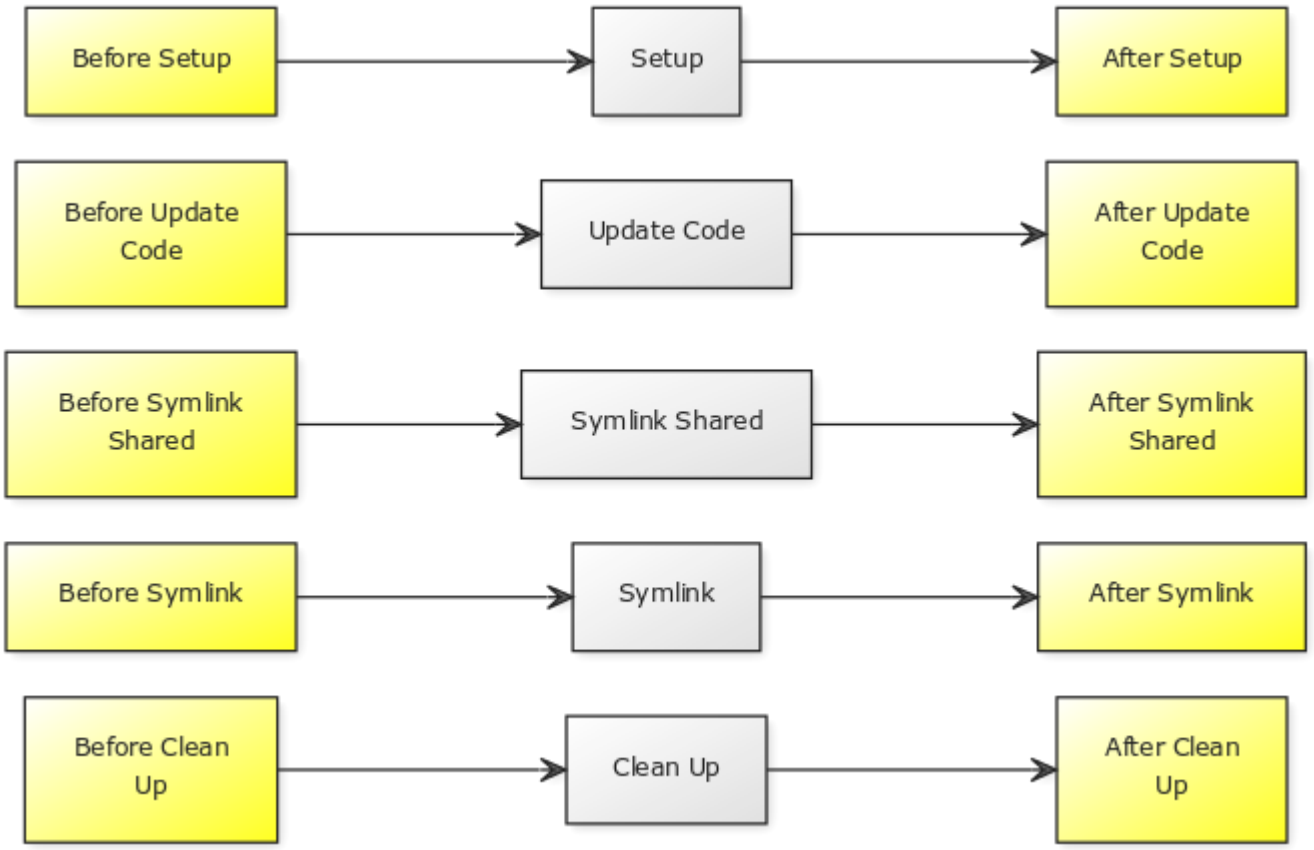
- Ansible en la máquina de despliegue,
- `rsync` o `git` la máquina cilente.

Puede descargar el código fuente desde `rsync`, `git`, `scp`, `http`, `S3`, ...

> **Note**
>
> Para nuestro despliegue de ejemplo, utilizaremos el protocolo `git`.

Ansistrano despliega las aplicaciones mediante los siguientes 5 pasos:

- **Setup**: crea la estructura de directorios para alojar los despliegues;
- **Update Code**: descargar la nueva versión a los equipos de destino;
- **Symlink Shared** y **Symlink**: después de desplegar la nueva versión, el enlace simbólico `current` se modifica para apuntar a esta nueva versión;
- **Clean Up**: para hacer algo de limpieza (eliminando versiones antiguas).



La estructura básica de un despliegue con Ansistrano tiene el siguiente aspecto:

```
/var/www/site/
├── current -> ./releases/20210718100000Z
├── releases
│   └── 20210718100000Z
│       ├── css -> ../../shared/css/
│       ├── img -> ../../shared/img/
│       └── REVISION
├── repo
└── shared
    ├── css/
    └── img/
```

Puede encontrar toda la documentación de Ansistrano en su repositorio de Github.

7.2 Laboratorios

Seguirá trabajando en sus 2 servidores:

El servidor de gestión:

• Ansible ya está instalado. Tendrá que instalar el rol `ansistrano.deploy`.

El servidor administrado:

• Tendrá que instalar Apache y desplegar el sitio del cliente.

7.2.1 Desplegar el servidor web

Para una mayor eficiencia, utilizaremos el rol `geerlingguy.apache` para configurar el servidor:

```
$ ansible-galaxy role install geerlingguy.apache
Starting galaxy role install process
- downloading role 'apache', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-apache/
archive/3.1.4.tar.gz
- extracting geerlingguy.apache to /home/ansible/.ansible/roles/
geerlingguy.apache
- geerlingguy.apache (3.1.4) was installed successfully
```

Probablemente necesitemos abrir algunas reglas de firewall, por lo que también instalaremos la colección `ansible.posix` para trabajr con su módulo `firewalld`:

```
$ ansible-galaxy collection install ansible.posix
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ansible-posix-1.2.0.tar.gz to /
home/ansible/.ansible/tmp/ansible-local-519039bp65pwn/tmpsvuj1fw5/ansible-
posix-1.2.0-bhjbfdpw
Installing 'ansible.posix:1.2.0' to '/home/ansible/.ansible/collections/
ansible_collections/ansible/posix'
ansible.posix:1.2.0 was installed successfully
```

Una vez instalados el rol y la colección, podemos crear la primera parte de nuestro playbook, que será:

- Instalar Apache,
- Crear una carpeta de destino para nuestro `vhost`,
- Crear el `vhost` por defecto,
- Abrir el cortafuegos,
- Iniciar o reiniciar Apache.

Consideraciones técnicas:

- Desplegaremos nuestro sitio en la carpeta `/var/www/site/`.
- Como veremos más adelante, `ansistrano` creará un enlace simbólico `current` a la carpeta con la versión actual.
- El código fuente a desplegar contiene una carpeta `html` a la que debe apuntar el vhost. Su `DirectoryIndex` apunta al fichero `index.htm`.
- El despliegue se realiza mediante `git`, el paquete se instalará.

Note

Por lo tanto, el objetivo de nuestro vhost será `/var/www/site/current/html`.

Nuestro playbook para configurar el servidor: `playbook-config-server.yml`

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
      DirectoryIndex index.php index.htm
    apache_vhosts:
      - servername: "website"
        documentroot: "{{ dest }}current/html"

  tasks:

    - name: create directory for website
      file:
```

```
        path: /var/www/site/
        state: directory
        mode: 0755

    - name: install git
      package:
        name: git
        state: latest

    - name: permit traffic in default zone for http service
      ansible.posix.firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: yes

  roles:
    - { role: geerlingguy.apache }
```

El playbook se puede aplicar al servidor:

```
$ ansible-playbook playbook-config-server.yml
```

Observe la ejecución de las siguientes tareas:

```
TASK [geerlingguy.apache : Ensure Apache is installed on RHEL.]
****************
TASK [geerlingguy.apache : Configure Apache.]
*********************************
TASK [geerlingguy.apache : Add apache vhosts configuration.]
******************
TASK [geerlingguy.apache : Ensure Apache has selected state and enabled on
boot.] ***
TASK [permit traffic in default zone for http service]
************************
RUNNING HANDLER [geerlingguy.apache : restart apache]
************************
```

El rol `geerlingguy.apache` nos facilita mucho el trabajo al encargarse de la instalación y configuración de Apache.

Puede comprobar que todo funciona utilizando el comando `curl`:

```
$ curl -I http://192.168.1.11
HTTP/1.1 404 Not Found
```

```
Date: Mon, 05 Jul 2021 23:30:02 GMT
Server: Apache/2.4.37 (rocky) OpenSSL/1.1.1g
Content-Type: text/html; charset=iso-8859-1
```

Note

Todavía no hemos desplegado ningún código, por lo que es normal que la ejecución del comando `curl` devuelva un código HTTP `404`. Pero ya podemos confirmar que el servicio `httpd` está funcionando y que el firewall está abierto.

7.2.2 Desplegar el software

Ahora que tenemos nuestro servidor configurado, podemos desplegar la aplicación.

Para ello, utilizaremos el rol `ansistrano.deploy` en un segundo playbook dedicado al despliegue de la aplicación (para mayor legibilidad).

```
$ ansible-galaxy role install ansistrano.deploy
Starting galaxy role install process
- downloading role 'deploy', owned by ansistrano
- downloading role from https://github.com/ansistrano/deploy/archive/
3.10.0.tar.gz
- extracting ansistrano.deploy to /home/ansible/.ansible/roles/
ansistrano.deploy
- ansistrano.deploy (3.10.0) was installed successfully
```

El código fuente del software se puede encontrar en el repositorio de github.

Crearemos un playbook `playbook-deploy.yml` para gestionar nuestro despliegue:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"

  roles:
     - { role: ansistrano.deploy }
```

```
$ ansible-playbook playbook-deploy.yml

PLAY [ansible_clients]
*******************************************************

TASK [ansistrano.deploy : ANSISTRANO | Ensure deployment base path exists]
*****
TASK [ansistrano.deploy : ANSISTRANO | Ensure releases folder exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared elements folder exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure basedir shared files exists]
TASK [ansistrano.deploy : ANSISTRANO | Get release version]
*******************
TASK [ansistrano.deploy : ANSISTRANO | Get release path]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Register ansistrano_git_result
variable]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Set git_real_repo_tree]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Create release folder]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Sync repo subtree[""] to release
path]
TASK [ansistrano.deploy : ANSISTRANO | Copy git released version into REVISION
file]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths targets are absent]
TASK [ansistrano.deploy : ANSISTRANO | Create softlinks for shared paths and
files]
TASK [ansistrano.deploy : ANSISTRANO | Ensure .rsync-filter is absent]
TASK [ansistrano.deploy : ANSISTRANO | Setup .rsync-filter with shared-folders]
TASK [ansistrano.deploy : ANSISTRANO | Get current folder]
TASK [ansistrano.deploy : ANSISTRANO | Remove current folder if it's a
directory]
TASK [ansistrano.deploy : ANSISTRANO | Change softlink to new release]
TASK [ansistrano.deploy : ANSISTRANO | Clean up releases]

PLAY RECAP
*********************************************************************************
192.168.1.11               : ok=25   changed=8    unreachable=0    failed=0
skipped=14   rescued=0    ignored=0
```

¡Se pueden hacer muchas cosas con sólo 11 líneas de código!

```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version Master</h1>
```

```
</body>
<html>
```

7.2.3 Comprobaciones en el servidor

Ahora puede conectarse vía ssh a su máquina cliente.

- Ejecute el comando `tree` en el directorio `/var/www/site/`:

```
$ tree /var/www/site/
/var/www/site
├── current -> ./releases/20210722155312Z
├── releases
│   └── 20210722155312Z
│       ├── REVISION
│       └── html
│           └── index.htm
├── repo
│   └── html
│       └── index.htm
└── shared
```

Por favor, ten en cuenta:

- el enlace simbólico `current` a la versión `./releases/20210722155312Z`
- la presencia de un directorio llamado `shared`
- la repsencia de repositorios de git en `./repo/`
- Desde el servidor de Ansible, reinicie el despliegue **3** veces, y luego compruebe en el cliente.

```
$ tree /var/www/site/
var/www/site
├── current -> ./releases/20210722160048Z
├── releases
│   ├── 20210722155312Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   ├── 20210722160032Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
```

```
│   ├── 20210722160040Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   └── 20210722160048Z
│       ├── REVISION
│       └── html
│           └── index.htm
├── repo
│   └── html
│       └── index.htm
└── shared
```

Por favor, ten en cuenta:

- `ansistrano` mantiene las 4 ultimas versiones,
- el enlace simbólico `current` ahora apunta a la última versión

7.2.4 Limitar el número de versiones

La variable `ansistrano_keep_releases` se utiliza para especificar el número de versiones a mantener.

- Utilizando la variable `ansistrano_keep_releases`, mantén sólo 3 versiones del proyecto. Compruebe.

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3

  roles:
     - { role: ansistrano.deploy }
```

```
---
$ ansible-playbook -i hosts playbook-deploy.yml
```

En la máquina cliente:

```
$ tree /var/www/site/
/var/www/site
├── current -> ./releases/20210722160318Z
├── releases
│   ├── 20210722160040Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   ├── 20210722160048Z
│   │   ├── REVISION
│   │   └── html
│   │       └── index.htm
│   └── 20210722160318Z
│       ├── REVISION
│       └── html
│           └── index.htm
├── repo
│   └── html
│       └── index.htm
└── shared
```

7.2.5 Uso de shared_paths y shared_files

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"

  roles:
     - { role: ansistrano.deploy }
```

En el equipo cliente, cree el archivo `logs` en el directorio `shared`:

```
sudo touch /var/www/site/shared/logs
```

Despues ejecute el playbook:

```
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths targets are absent]
*********************************************************
ok: [192.168.10.11] => (item=img)
ok: [192.168.10.11] => (item=css)
ok: [192.168.10.11] => (item=logs/log)

TASK [ansistrano.deploy : ANSISTRANO | Create softlinks for shared paths and
files] ************************************************
changed: [192.168.10.11] => (item=img)
changed: [192.168.10.11] => (item=css)
changed: [192.168.10.11] => (item=logs)
```

En la máquina cliente:

```
$  tree -F /var/www/site/
/var/www/site/
├── current -> ./releases/20210722160631Z/
├── releases/
│   ├── 20210722160048Z/
│   │   ├── REVISION
│   │   └── html/
│   │       └── index.htm
│   ├── 20210722160318Z/
│   │   ├── REVISION
│   │   └── html/
│   │       └── index.htm
│   └── 20210722160631Z/
│       ├── REVISION
│       ├── css -> ../../shared/css/
│       ├── html/
│       │   └── index.htm
│       ├── img -> ../../shared/img/
│       └── logs -> ../../shared/logs
├── repo/
│   └── html/
│       └── index.htm
└── shared/
    ├── css/
```

```
    ├── img/
    └── logs
```

Tenga en cuenta que la última versión contiene 3 enlaces: `css`, `img`, and `logs`

- desde `/var/www/site/releases/css` al directorio `../../shared/css/`.
- desde `/var/www/site/releases/img` al directorio `../../shared/img/`.
- desde `/var/www/site/releases/logs` al archivo `../../shared/logs`.

Por lo tanto, los archivos contenidos en estas 2 carpetas y el archivo `logs` son siempre accesibles a través de las siguientes rutas:

- `/var/www/site/current/css/`,
- `/var/www/site/current/img/`,
- `/var/www/site/current/logs`,

pero sobre todo se mantendrán de un despliegue a otro.

7.2.6 Utilizar un subdirectorio del repositorio para el despliegue

En nuestro caso, el repositorio contiene una carpeta `html`, que contiene los archivos del sitio.

- Para evitar este nivel extra de directorio, utilice la variable `ansistrano_git_repo_tree` especificando la ruta del subdirectorio a utilizar.

¡No olvide modificar la configuración de Apache para tener en cuenta este cambio!

Cambie el playbook para la configuración del servidor `playbook-config-server.yml`

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
      DirectoryIndex index.php index.htm
    apache_vhosts:
      - servername: "website"
        documentroot: "{{ dest }}current/" # <1>
```

```
  tasks:

    - name: create directory for website
      file:
        path: /var/www/site/
        state: directory
        mode: 0755

    - name: install git
      package:
        name: git
        state: latest

  roles:
    - { role: geerlingguy.apache }
```

<1> Edite esta línea

Cambie el playbook para el despliegue `playbook-deploy.yml`

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html' # <1>

  roles:
     - { role: ansistrano.deploy }
```

<1> Edite esta línea

- No se olvide de ejecutar los dos playbooks
- Compruebe en la máquina cliente:

```
$  tree -F /var/www/site/
/var/www/site/
├── current -> ./releases/20210722161542Z/
├── releases/
│   ├── 20210722160318Z/
│   │   ├── REVISION
│   │   └── html/
│   │       └── index.htm
│   ├── 20210722160631Z/
│   │   ├── REVISION
│   │   ├── css -> ../../shared/css/
│   │   ├── html/
│   │   │   └── index.htm
│   │   ├── img -> ../../shared/img/
│   │   └── logs -> ../../shared/logs
│   └── 20210722161542Z/
│       ├── REVISION
│       ├── css -> ../../shared/css/
│       ├── img -> ../../shared/img/
│       ├── index.htm
│       └── logs -> ../../shared/logs
├── repo/
│   └── html/
│       └── index.htm
└── shared/
    ├── css/
    ├── img/
    └── logs
```

<1> Obsérvese la ausencia de `html`

7.2.7 Gestión de las ramas o de las etiquetas de Git

La variable `ansistrano_git_branch` se utiliza para especificar una `rama` o una `etiqueta` de Git a desplegar.

- Despliegue de la rama `releases/v1.1.0`:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'releases/v1.1.0'

  roles:
     - { role: ansistrano.deploy }
```

Note

Puede divertirse, durante el despliegue, refrescando su navegador, para ver el cambio en 'vivo'.

```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version 1.0.1</h1>
</body>
<html>
```

- Despliegue de la etiqueta `v2.0.0`:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
```

```
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'

  roles:
     - { role: ansistrano.deploy }
```

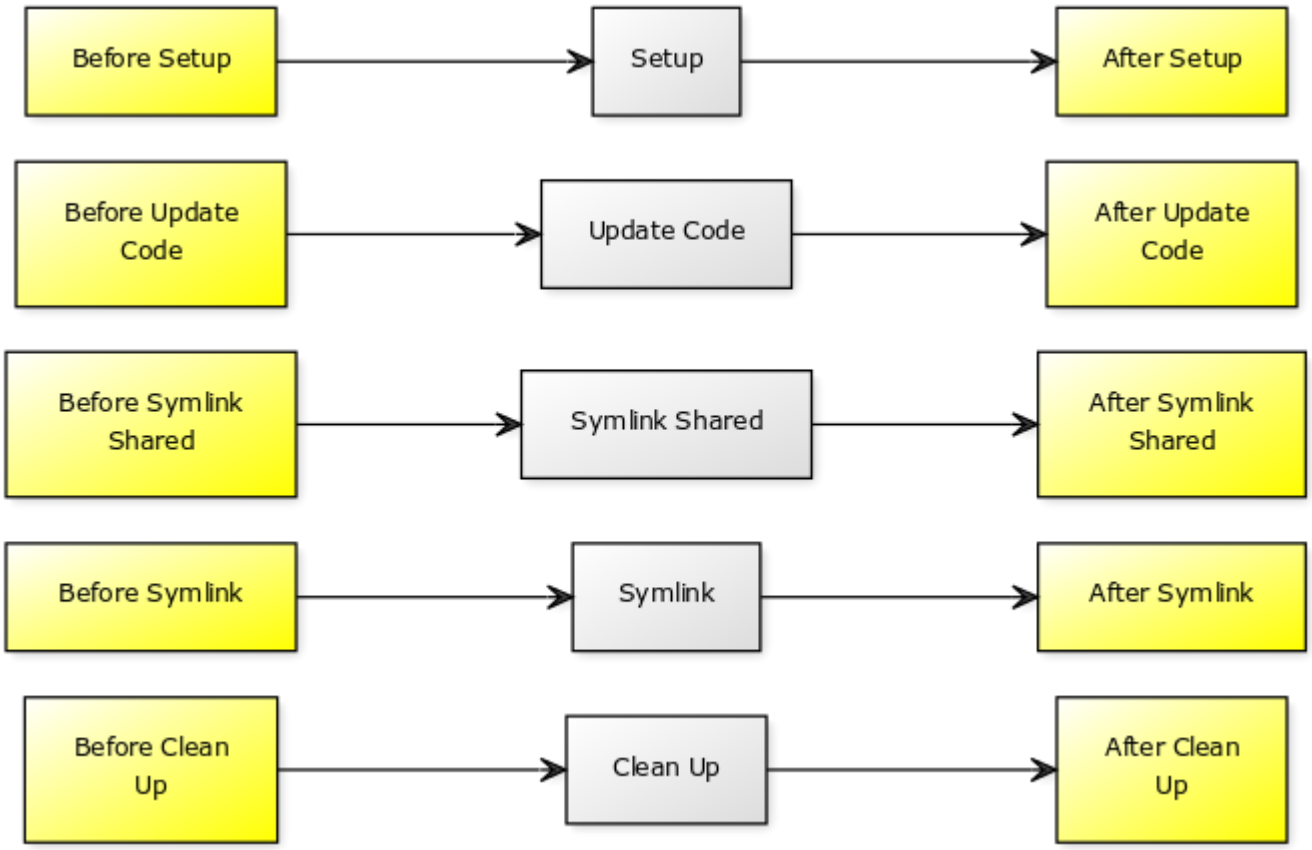
```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version 2.0.0</h1>
</body>
<html>
```

7.2.8 Acciones entre los pasos del despliegue

Un despliegue con Ansistrano respeta los siguientes pasos:

- `Setup`
- `Update Code`
- `Symlink Shared`
- `Symlink Shared`
- `Clean Up`

Es posible intervenir antes y después de cada uno de estos pasos.



Se puede incluir un playbook a través de las variables previstas para ello:

- `ansistrano_before_<task>_tasks_file`
- o `ansistrano_after_<task>_tasks_file`
- Ejemplo sencillo: Enviar un correo electrónico (o lo que quiera como una notificación de Slack) al comenzar el despliegue:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"
```

```
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'
    ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-
setup-tasks.yml"

  roles:
     - { role: ansistrano.deploy }
```

Cree el fichero `deploy/before-setup-tasks.yml`:

```
---
- name: Send a mail
  mail:
    subject: Starting deployment on {{ ansible_hostname }}.
  delegate_to: localhost
```

```
TASK [ansistrano.deploy : include]
*********************************************************************************
included: /home/ansible/deploy/before-setup-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : Send a mail]
*********************************************************************************
ok: [192.168.10.11 -> localhost]
```

```
[root] # mailx
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N  1 root@localhost.local  Tue Aug 21 14:41  28/946   "Starting deployment on
localhost."
```

- Probablemente tendrá que reiniciar algunos servicios al final del despliegue, para vaciar las cachés, por ejemplo. Vamos a reiniciar Apache al final del despliegue:

```
---
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
```

```
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'
    ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-
setup-tasks.yml"
    ansistrano_after_symlink_tasks_file: "{{ playbook_dir }}/deploy/after-
symlink-tasks.yml"

  roles:
     - { role: ansistrano.deploy }
```

Cree el fichero `deploy/after-symlink-tasks.yml`:

```
---
- name: restart apache
  systemd:
    name: httpd
    state: restarted
```

```
TASK [ansistrano.deploy : include]
*********************************************************************************
included: /home/ansible/deploy/after-symlink-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : restart apache]
*********************************************************************************
changed: [192.168.10.11]
```

Como ha visto durante este capítulo, Ansible puede mejorar mucho la vida del administrador de sistemas. Los roles muy inteligentes como Ansistrano se convierten rápidamente en indispensables.

El uso de Ansistrano garantiza el respeto de las buenas prácticas en los despliegues, reduce el tiempo necesario para poner un sistema nuevo en producción y evita el riesgo de posibles errores humanos. ¡Las máquinas funcionan rápidas, bien y rara vez cometen errores.!

8. Ansible - Infraestructura a gran escala

En este capítulo aprenderá a escalar su sistema de gestión de la configuración.

Objetivos : En este capítulo aprenderá a:

✔ Organice su código para la gestión de una gran infraestructura;
✔ Aplicar toda o una parte de la gestión de la configuración a un grupo de nodos;

🏁 **ansible**, **gestion de la configuración**, **escalado**

Conocimiento: ★ ★ ★
Complejidad: ★ ★ ★ ★

Tiempo de lectura: 30 minutos

En los capítulos anteriores hemos visto cómo organizar nuestro código en forma de roles, pero también cómo utilizar algunos roles para la gestión de actualizaciones (gestión de parches) o el despliegue de código.

Pero ¿Qué ocurre con la gestión de la configuración? ¿Cómo gestionar la configuración de decenas, cientos o incluso miles de máquinas virtuales con Ansible?

La llegada de la nube ha cambiado un poco los métodos de administración tradicionales. La máquina virtual se configura en el momento del despliegue. Si su configuración ya no es válida, se destruye y se sustituye por una nueva.

La organización del sistema de gestión de la configuración que se presenta en este capítulo responderá a estas dos formas de consumir TI: uso "puntual" o "reconfiguración" regular de una flota de servidores.

Sin embargo, hay que tener cuidado: el uso de Ansible para garantizar el cumplimiento del parque del parque de servidores equiere cambiar sus hábitos de trabajo. Ya no es posible modificar manualmente la configuración de un sistema

gestor de servicios sin que estas modificaciones se sobrescriban la próxima vez que se ejecute Ansible.

> **Note**
>
> Lo que vamos a configurar a continuación no es el terreno favorito de Ansible. Tecnologías como Puppet o Salt lo harán mucho mejor. Recordemos que Ansible es una navaja suiza de la automatización y no tiene agentes, lo que explica las diferencias de rendimiento.

> **Note**
>
> Puede encontrar más información aquí

8.1 Almacenamiento de variables

Lo primero que tenemos que discutir es la separación entre los datos y el código de Ansible.

A medida que el código se hace más grande y complejo, cada vez será más complicado modificar las variables que contiene.

Para asegurar el mantenimiento de su sitio web, lo más importante es separar correctamente las variables del código de Ansible.

Todavía no lo hemos discutido aquí, pero debe saber que Ansible puede cargar automáticamente las variables que encuentra en carpetas específicas dependiendo del nombre de inventario del nodo gestionado, o de sus grupos de miembros.

La documentación de Ansible sugiere que organicemos nuestro código tal y como se indica a continuación:

```
inventories/
   production/
      hosts               # inventory file for production servers
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         hostname1.yml    # here we assign variables to particular systems
         hostname2.yml
```

Si el nodo objetivo es `hostname1` que pertenece a `group1`, las variables contenidas en los archivos `hostname1.yml` y en `group1.yml` se cargarán automáticamente. Es

una buena manera de almacenar todos los datos de todos sus roles en el mismo lugar.

De esta manera, el archivo de inventario de su servidor se convierte en su tarjeta de identidad. Ya que contiene todas las variables que difieren de las variables por defecto de su servidor.

Desde el punto de vista de la centralización de las variables, se hace imprescindible utilizar una nomenclatura para las variables utilizadas en sus roles anteponiendo, por ejemplo, el nombre del rol. También se recomienda utilizar nombres de variables planos en lugar de diccionarios.

Por ejemplo, si quiere que el valor del parámetro `PermitRootLogin` en el archivo `sshd_config` sea configurable, un buen nombre de variable podría ser `sshd_config_permitrootlogin` (en lugar de `sshd.config.permitrootlogin` el cual también podría ser un buen nombre de variable).

8.2 Acerca de las etiquetas Ansible

El uso de las etiquetas de Ansible le permite ejecutar u omitir un conjunto de las tareas definidas en su código.

> **Note**
>
> Puede encontrar más información aquí

Por ejemplo, vamos a modificar nuestra tarea de creación de usuarios:

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  loop:
     - antoine
     - patrick
     - steven
     - xavier
  tags: users
```

Ahora puede reproducir sólo las tareas con la etiqueta `users` mediante la opción `--tags` de la herramienta `ansible-playbook`:

```
ansible-playbook -i inventories/production/hosts --tags users site.yml
```

También es posible utilizar la opción `--skip-tags`.

8.3 Acerca del diseño de directorios

Vamos a centrarnos en una propuesta de organización de los archivos y directorios necesarios para el buen funcionamiento de un CMS (Sistema de Gestión de Contenidos).

Nuestro punto de partida será el archivo `site.yml`. Este archivo, es un poco como el director de orquesta del CMS, ya que sólo incluirá los roles necesarios para los nodos de destino si es necesario:

```
---
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"

  roles:

    - role: roles/functionality1

    - role: roles/functionality2
```

Por supuesto, esos roles deben ser creados bajo el directorio `roles` al mismo nivel que el archivo `site.yml`.

Me gusta manejar mis vars globales dentro del archivo `vars/global_vars.yml`, aunque tambien podría almacenarlos dentro de un archivo ubicado en `inventories/production/group_vars/all.yml`

```
---
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"
  vars_files:
    - vars/global_vars.yml
  roles:
```

```
    - role: roles/functionality1

    - role: roles/functionality2
```

Personalmente, también me gusta mantener la posibilidad de desactivar una funcionalidad. Por lo que incluyo en mis roles una condición y un valor por defecto como este:

```
---
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"
  vars_files:
    - vars/global_vars.yml
  roles:

    - role: roles/functionality1
      when:
        - enable_functionality1|default(true)

    - role: roles/functionality2
      when:
        - enable_functionality2|default(false)
```

No se olvide de utilizar las etiquetas:

```
- name: "Config Management for {{ target }}"
  hosts: "{{ target }}"
  vars_files:
    - vars/global_vars.yml
  roles:

    - role: roles/functionality1
      when:
        - enable_functionality1|default(true)
      tags:
        - functionality1

    - role: roles/functionality2
      when:
        - enable_functionality2|default(false)
      tags:
        - functionality2
```

Debería ver algo como esto:

```
$ tree cms
cms
├── inventories
│   └── production
│       ├── group_vars
│       │   └── plateform.yml
│       ├── hosts
│       └── host_vars
│           ├── client1.yml
│           └── client2.yml
├── roles
│   ├── functionality1
│   │   ├── defaults
│   │   │   └── main.yml
│   │   └── tasks
│   │       └── main.yml
│   └── functionality2
│       ├── defaults
│       │   └── main.yml
│       └── tasks
│           └── main.yml
├── site.yml
└── vars
    └── global_vars.yml
```

Note

Es libre de desarrollar sus roles dentro de una colección

8.4 Pruebas

Vamos a ejecutar el playbook y a realizar algunas pruebas:

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" site.yml

PLAY [Config Management for client1]
*******************************************************************************

TASK [Gathering Facts]
*******************************************************************************
ok: [client1]

TASK [roles/functionality1 : Task in functionality 1]
*******************************************************
ok: [client1] => {
```

```
    "msg": "You are in functionality 1"
}

TASK [roles/functionality2 : Task in functionality 2]
*********************************************************
skipping: [client1]

PLAY RECAP
*****************************************************************************************
client1                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Como puede ver, por defecto, sólo se ejecutan las tareas del rol `functionality1`.

Activemos en el inventario la `functionality2` para nuestro nodo objetivo y volvamos a ejecutar el playbook:

```
$ vim inventories/production/host_vars/client1.yml
---
enable_functionality2: true
```

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" site.yml

PLAY [Config Management for client1]
*****************************************************************************

TASK [Gathering Facts]
*****************************************************************************************
ok: [client1]

TASK [roles/functionality1 : Task in functionality 1]
*********************************************************
ok: [client1] => {
    "msg": "You are in functionality 1"
}

TASK [roles/functionality2 : Task in functionality 2]
*********************************************************
ok: [client1] => {
    "msg": "You are in functionality 2"
}

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

```
client1                    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Intentamos aplicar únicamente la tarea `functionality2`:

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" --tags
functionality2 site.yml

PLAY [Config Management for client1]
*************************************************************************

TASK [Gathering Facts]
*************************************************************************************
ok: [client1]

TASK [roles/functionality2 : Task in functionality 2]
*******************************************************
ok: [client1] => {
    "msg": "You are in functionality 2"
}

PLAY RECAP
*************************************************************************************
client1                    : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Vamos a ejecutar la tarea contra todo el inventario:

```
$ ansible-playbook -i inventories/production/hosts -e "target=plateform"
site.yml

PLAY [Config Management for plateform]
*************************************************************************

TASK [Gathering Facts]
*************************************************************************************
ok: [client1]
ok: [client2]

TASK [roles/functionality1 : Task in functionality 1]
*******************************************************
ok: [client1] => {
    "msg": "You are in functionality 1"
}
ok: [client2] => {
    "msg": "You are in functionality 1"
```

```
}

TASK [roles/functionality2 : Task in functionality 2]
*****************************************************
ok: [client1] => {
    "msg": "You are in functionality 2"
}
skipping: [client2]

PLAY RECAP
*********************************************************************************
client1                    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
client2                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Como puede ver, `functionality2` sólo se ejecuta en el `cliente1`.

8.5 Beneficios

Siguiendo los consejos incluidos en la documentación de Ansible, obtendrá rápidamente un:

- Un código fuente fácil de mantener aunque contenga un gran número de funciones.
- Un sistema de cumplimiento relativamente rápido y repetible que puede aplicar parcial o totalmente.
- Puede adaptarse en función de cada caso y del servidor.
- Los detalles de su sistema de información están separados del código, son fácilmente auditables y están centralizados en los archivos de inventario de su gestión de la configuración.

9. Ansible - Trabajar con filtros

En este capítulo aprenderá a transformar datos mediante filtros de Jinja.

Objetivos : En este capítulo aprenderá a:

✔ Transformar estructuras de datos como diccionarios o listas;
✔ Transformar variables;

🏁 **ansible**, **jinja**, **filtros**

Conocimiento: ⭐ ⭐ ⭐
Complejidad: ⭐ ⭐ ⭐ ⭐

Tiempo de lectura: 20 minutos

Ya hemos tenido la oportunidad, durante los capítulos anteriores, de utilizar los filtros de Jinja.

Estos filtros, escritos en python, nos permiten manipular y transformar nuestras variables de ansible.

> **Note**
>
> Puede encontrar más información aquí.

A lo largo de este capítulo, utilizaremos el siguiente playbook para probar los diferentes filtros presentados:

```
- name: Manipulating the data
  hosts: localhost
  gather_facts: false
  vars:
    zero: 0
    zero_string: "0"
    non_zero: 4
    true_booleen: True
    true_non_booleen: "True"
    false_boolean: False
```

```
    false_non_boolean: "False"
    whatever: "It's false!"
    user_name: antoine
    my_dictionary:
      key1: value1
      key2: value2
    my_simple_list:
      - value_list_1
      - value_list_2
      - value_list_3
    my_simple_list_2:
      - value_list_3
      - value_list_4
      - value_list_5
    my_list:
      - element: element1
        value: value1
      - element: element2
        value: value2

  tasks:
    - name: Print an integer
      debug:
        var: zero
```

Note

La siguiente es una lista no exhaustiva de los filtros que probablemente se encontrará o necesitará mientras trabaje con Ansible. Afortunadamente, hay muchos otros. ¡Incluso podría escribir el suyo propio!

El playbook se ejecutará de la siguiente manera:

```
ansible-playbook play-filter.yml
```

9.1 Convertir datos

Los datos se pueden convertir de un tipo a otro.

Para conocer el tipo de un dato (el tipo en lenguaje python), hay que utilizar el filtro `type_debug`.

Ejemplo:

```
- name: Display the type of a variable
  debug:
    var: true_boolean|type_debug
```

esto produce:

```
TASK [Display the type of a variable]
*********************************************************************
ok: [localhost] => {
    "true_boolean|type_debug": "bool"
}
```

Es posible transformar un entero en una cadena:

```
- name: Transforming a variable type
  debug:
    var: zero|string
```

```
TASK [Transforming a variable type]
*********************************************************************
ok: [localhost] => {
    "zero|string": "0"
}
```

Convertir una cadena en un entero:

```
- name: Transforming a variable type
  debug:
    var: zero_string|int
```

o una variable en un booleano:

```
- name: Display an integer as a boolean
  debug:
    var: non_zero | bool

- name: Display a string as a boolean
  debug:
    var: true_non_boolean | bool

- name: Display a string as a boolean
  debug:
    var: false_non_boolean | bool
```

```
- name: Display a string as a boolean
  debug:
    var: whatever | bool
```

Una cadena de caracteres se puede transformar a mayúsculas o minúsculas:

```
- name: Lowercase a string of characters
  debug:
    var: whatever | lower

- name: Upercase a string of characters
  debug:
    var: whatever | upper
```

esto produce:

```
TASK [Lowercase a string of characters]
*****************************************************
ok: [localhost] => {
    "whatever | lower": "it's false!"
}

TASK [Upercase a string of characters]
*****************************************************
ok: [localhost] => {
    "whatever | upper": "IT'S FALSE!"
}
```

El filtro `reemplazar` le permite sustituir unos caracteres por otros.

En este ejemplo, eliminamos espacios o incluso sustituimos una palabra:

```
- name: Replace a character in a string
  debug:
    var: whatever | replace(" ", "")

- name: Replace a word in a string
  debug:
    var: whatever | replace("false", "true")
```

esto produce:

```
TASK [Replace a character in a string]
*************************************************
ok: [localhost] => {
    "whatever | replace(\" \", \"\")": "It'sfalse!"
}

TASK [Replace a word in a string]
*************************************************
ok: [localhost] => {
    "whatever | replace(\"false\", \"true\")": "It's true !"
}
```

El filtro `split` le permite dividir una cadena en una lista basada en un carácter:

```
- name: Cutting a string of characters
  debug:
    var: whatever | split(" ", "")
```

```
TASK [Cutting a string of characters]
*************************************************
ok: [localhost] => {
    "whatever | split(\" \")": [
        "It's",
        "false!"
    ]
}
```

9.2 Unir los elementos de una lista

Es frecuente tener que unir los diferentes elementos de una lista en una única cadena. A continuación, podemos especificar un carácter o una cadena para insertar entre cada elemento.

```
- name: Joining elements of a list
  debug:
    var: my_simple_list|join(",")

- name: Joining elements of a list
  debug:
    var: my_simple_list|join(" | ")
```

esto produce:

```
TASK [Joining elements of a list]
*****************************************************************
ok: [localhost] => {
    "my_simple_list|join(\",\")": "value_list_1,value_list_2,value_list_3"
}

TASK [Joining elements of a list]
*****************************************************************
ok: [localhost] => {
    "my_simple_list|join(\" | \")": "value_list_1 | value_list_2 |
value_list_3"
}
```

9.3 Transformar diccionarios en listas (y viceversa)

Los filtros `dict2items` y `itemstodict`, son filtros más complejos de implementar y se utilizan frecuentemente, especialmente en bucles.

Observe que es posible especificar el nombre de la clave y del valor a utilizar en la transformación.

```
- name: Display a dictionary
  debug:
    var: my_dictionary

- name: Transforming a dictionary into a list
  debug:
    var: my_dictionary | dict2items

- name: Transforming a dictionary into a list
  debug:
    var: my_dictionary | dict2items(key_name='key', value_name='value')

- name: Transforming a list into a dictionary
  debug:
    var: my_list | items2dict(key_name='element', value_name='value')
```

```
TASK [Display a dictionary]
*********************************************************************
ok: [localhost] => {
    "my_dictionary": {
        "key1": "value1",
        "key2": "value2"
    }
```

```
}

TASK [Transforming a dictionary into a list]
************************************************************
ok: [localhost] => {
    "my_dictionary | dict2items": [
        {
            "key": "key1",
            "value": "value1"
        },
        {
            "key": "key2",
            "value": "value2"
        }
    ]
}

TASK [Transforming a dictionary into a list]
************************************************************
ok: [localhost] => {
    "my_dictionary | dict2items (key_name = 'key', value_name = 'value')": [
        {
            "key": "key1",
            "value": "value1"
        },
        {
            "key": "key2",
            "value": "value2"
        }
    ]
}

TASK [Transforming a list into a dictionary]
************************************************************
ok: [localhost] => {
    "my_list | items2dict(key_name='element', value_name='value')": {
        "element1": "value1",
        "element2": "value2"
    }
}
```

9.4 Trabajar con listas

Es posible fusionar o filtrar datos de una o varias listas:

```
- name: Merger of two lists
  debug:
    var: my_simple_list | union(my_simple_list_2)
```

```
ok: [localhost] => {
    "my_simple_list | union(my_simple_list_2)": [
        "value_list_1",
        "value_list_2",
        "value_list_3",
        "value_list_4",
        "value_list_5"
    ]
}
```

Para mantener únicamente la intersección de las 2 listas (los valores presentes en las 2 listas):

```
- name: Merger of two lists
  debug:
    var: my_simple_list | intersect(my_simple_list_2)
```

```
TASK [Merger of two lists]
*********************************************************************************
ok: [localhost] => {
    "my_simple_list | intersect(my_simple_list_2)": [
        "value_list_3"
    ]
}
```

O, por el contrario, mantener sólo la diferencia (los valores que no existen en la segunda lista):

```
- name: Merger of two lists
  debug:
    var: my_simple_list | difference(my_simple_list_2)
```

```
TASK [Merger of two lists]
*********************************************************************************
ok: [localhost] => {
    "my_simple_list | difference(my_simple_list_2)": [
        "value_list_1",
        "value_list_2",
```

```
    ]
}
```

Si su lista contiene valores que no son únicos, también es posible filtrarlos mediante el filtro `unique`.

```
- name: Unique value in a list
  debug:
    var: my_simple_list | unique
```

9.5 Transformación json/yaml

Es posible que tenga que importar datos en formato json (desde una API, por ejemplo) o exportar datos en formato yaml o json.

```
- name: Display a variable in yaml
  debug:
    var: my_list | to_nice_yaml(indent=4)

- name: Display a variable in json
  debug:
    var: my_list | to_nice_json(indent=4)
```

```
TASK [Display a variable in yaml]
**********************************************************************
ok: [localhost] => {
    "my_list | to_nice_yaml(indent=4)": "-   element: element1\n    value:
value1\n-   element: element2\n    value: value2\n"
}

TASK [Display a variable in json]
**********************************************************************
ok: [localhost] => {
    "my_list | to_nice_json(indent=4)": "[\n    {\n        \"element\":
\"element1\",\n        \"value\": \"value1\"\n    },\n    {\n
\"element\": \"element2\",\n        \"value\": \"value2\"\n    }\n]"
}
```

9.6 Valores por defecto, variables opcionales, variables protegidas

Se encontrará rápidamente con errores en la ejecución de sus playbooks si no proporciona valores por defecto para sus variables, o si no las proteges.

Si no existe el valor de una variable, se puede ser sustituir por otro valor mediante el filtro `default`:

```
- name: Default value
  debug:
    var: variablethatdoesnotexists | default(whatever)
```

```
TASK [Default value]
*********************************************************************************
ok: [localhost] => {
    "variablethatdoesnotexists | default(whatever)": "It's false!"
}
```

Observe la presencia del apóstrofe `'` que debería estar protegido, por ejemplo, si utiliza el modulo `shell`:

```
- name: Default value
  debug:
    var: variablethatdoesnotexists | default(whatever| quote)
```

```
TASK [Default value]
*********************************************************************************
ok: [localhost] => {
    "variablethatdoesnotexists | default(whatever|quote)": "'It'\"'\"'s
false!'"
}
```

Finalmente, si no existe una variable opcional en un módulo se puede ignorar utilizando con la palabra clave `omit` en el filtro `default`, lo que le ahorrará un error en tiempo de ejecución.

```
- name: Add a new user
  ansible.builtin.user:
    name: "{{ user_name }}"
    comment: "{{ user_comment | default(omit) }}"
```

9.7 Asociar un valor en función de otro (`ternary`)

A veces es necesario utilizar una condición para asignar un valor a una variable, en cuyo caso es común pasar por un paso `set_fact`.

Esto se puede evitar utilizando el filtro `ternary`:

```
- name: Default value
  debug:
    var: (user_name == 'antoine') | ternary('admin', 'normal_user')
```

```
TASK [Default value]
*********************************************************************************
ok: [localhost] => {
    "(user_name == 'antoine') | ternary('admin', 'normal_user')": "admin"
}
```

9.8 Otros filtros

- `{{ 10000 | random }}`: Como su nombre indica, da un valor aleatorio.
- `{{ my_simple_list | first }}`: Extrae el primer elemento de la lista.
- `{{ my_simple_list | length }}`: Da la longitud (de una lista o una cadena).
- `{{ ip_list | ansible.netcommon.ipv4 }}`: Sólo muestra las IPs v4. Sin insistir en esto, si lo necesita, hay muchos filtros dedicados a la red.
- `{{ user_password | password_hash('sha512') }}`: Genera una contraseña con hash en sha512.

10. Optimizaciones del servidor de gestión

En este capítulo revisaremos las opciones de configuración que pueden ser de interés para optimizar nuestro servidor de gestión de Ansible.

10.1 El archivo de configuración `ansible.cfg`

A continuación vamos a comentar algunas opciones de configuración interesantes de Ansible:

- `forks`: Establecido por defecto a 5, es el número de procesos que Ansible lanzará en paralelo para comunicarse con los hosts remotos. Cuanto más alto sea este número, más clientes podrá gestionar Ansible al mismo tiempo, y así acelerar los procesos. El valor que puede establecer depende de los límites de CPU/RAM de su servidor de gestión. Observe que el valor por defecto, `5`, es muy pequeño, la documentación de Ansible indica que muchos usuarios lo establecen en 50, en 500 o incluso en valores más altos.
- `gathering`: Esta variable cambia la política de recogida de datos. Por defecto, el valor se establece a `implicit`, lo que implica que los datos se recopilarán sistemáticamente. El cambio de esta variable a `smart` permite recopilar las colecciones de datos sólo cuando no se han recogido con anterioridad. Si se combina con una caché de datos (véase más adelante), esta opción puede aumentar considerablemente el rendimiento.
- `host_key_checking`: ¡Cuidado con la seguridad de su servidor! Sin embargo, si tiene control de su entorno, puede ser interesante desactivar el control de llaves de sus servidores remotos y ahorrar algo de tiempo en la conexión. En servidores remotos, puede deshabilitar el uso de DNS en el servidor SSH (en `/etc/sshd_config`, medianrte la opción `UseDNS no`). Esta opción hace perder tiempo en la conexión y, la mayoría de las veces, sólo se utiliza para los registros de conexión.
- `ansible_managed`: Esta variable, que contiene el valor `Ansible managed` por defecto, se suele utilizar en plantillas de archivos que se despliegan en servidores remotos. Permite a un administrador especificar que el archivo se gestiona automáticamente y que potencialmente cualquier cambio que haga en él se perderá. Puede ser interesante que los administradores tengan mensajes más completos. Sin embargo, tenga cuidado, si cambia esta variable, puede hacer que los demonios se reinicien (a través de los handlers asociados a las plantillas).
- `ssh_args = -C -o ControlMaster=auto -o ControlPersist=300s -o PreferredAuthentications=publickey`: Especificar las opciones de conexión mediante ssh. Puede ahorrar mucho tiempo, si desactiva todos los métodos de autenticación que no sean de clave pública. También puede incrementar el valor del parámetro `ControlPersist` para mejorar el rendimiento (la documentación sugiere que un valor equivalente a 30 minutos puede ser apropiado). La conexión con un cliente permanecerá abierta durante más tiempo y podrá reutilizarse

cuando se vuelva a conectar al mismo servidor, lo que supone un importante ahorro de tiempo.

- `control_path_dir`: Especifica la ruta de acceso a los sockets de conexión. Si la ruta es demasiado larga, puede provocar problemas. Valore la posibilidad de cambiarlo por algo más corto, como `/tmp/.cp`.
- `pipelining`: El establecer este valor a `True` aumenta el rendimiento al reducir el número de conexiones SSH necesarias cuando se ejecutan los módulos remotos. Primero debe asegurarse de que la opción `requiretty` está desactivada en las opciones de `sudoers` (vea la documentación).

10.2 Almacenamiento de los datos

La recopilación de datos es un proceso que puede llevar algún tiempo. Puede resultar interesante desactivar la recopilación de datos para los playbooks que no la necesiten (mediante la opción `gather_facts`) o mantener estos datos en una memoria caché durante un periodo de tiempo determinado (por ejemplo 24H).

Estos datos se pueden almacenar fácilmente en una base de datos `redis`:

```
sudo yum install redis
sudo systemctl start redis
sudo systemctl enable redis
sudo pip3 install redis
```

No olvide modificar la configuración de ansible:

```
fact_caching = redis
fact_caching_timeout = 86400
fact_caching_connection = localhost:6379:0
```

Para comprobar el correcto funcionamiento, basta con hacer una petición al servidor `redis`:

```
redis-cli
127.0.0.1:6379> keys *
127.0.0.1:6379> get ansible_facts_SERVERNAME
```

10.3 Utilizar Vault

Las distintas contraseñas y secretos no e pueden almacenar en texto plano dentro del código de Ansible, ni localmente en el servidor de gestión ni en una posible herramienta de gestión de código fuente.

Ansible propone utilizar un gestor de encriptado: `ansible-vault`.

El principio es cifrar una variable o un archivo completo mediante el comando `ansible-vault`.

Ansible podrá descifrar este archivo en tiempo de ejecución recuperando la clave de cifrado del archivo (por ejemplo) `/etc/ansible/ansible.cfg`. Esto último también puede ser un script de python o cualquier otra cosa.

Edite el archivo `/etc/ansible/ansible.cfg`:

```
#vault_password_file = /path/to/vault_password_file
vault_password_file = /etc/ansible/vault_pass
```

Guarde la contraseña en el archivo `/etc/ansible/vault_pass` y asigne los permisos y las restricciones necesarias:

```
mysecretpassword
```

A continuación, puede encriptar sus archivos mediante el comando:

```
ansible-vault encrypt myfile.yml
```

Un archivo encriptado mediante `ansible-vault` se puede reconocer fácilmente por su cabecera:

```
$ANSIBLE_VAULT;1.1;AES256
353765323436633533306131336638346261363162343239643337353633333961366132663839666
666432226163326135656638343839373838616533396666 0a34303266323334376263393631363 0
343732301245616637663061346562353862333239643362393366616534336630366333343666616
434656630306261650a313364636261393931313739363931336664386536333766326264633330
6334
```

Una vez que un archivo está encriptado, todavía puede editarse mediante el comando:

```
ansible-vault edit myfile.yml
```

También puede delegar el almacenamiento de contraseñas a su gestor de contraseñas de confianza.

Por ejemplo, para recuperar una contraseña que estaría almacenada en la aplicación Rundeck:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import urllib.request
import io
import ssl

def get_password():
    '''
    :return: Vault password
    :return_type: str
    '''
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE

    url = 'https://rundeck.rockylinux.org/api/11/storage/keys/ansible/vault'
    req = urllib.request.Request(url, headers={
                          'Accept': '*/*',
                          'X-Rundeck-Auth-Token': '****token-rundeck****'
                          })
    response = urllib.request.urlopen(req, context=ctx)

    return response.read().decode('utf-8')

if __name__ == '__main__':
    print(get_password())
```

10.4 Trabajar con servidores Windows

Será necesario instalar varios paquetes en el servidor de gestión:

- Mediante el gestor de paquetes:

```
sudo dnf install python38-devel krb5-devel krb5-libs krb5-workstation
```

y configure el archivo `/etc/krb5.conf` para especificar el valor correcto de los `realms`:

```
[realms]
ROCKYLINUX.ORG = {
    kdc = dc1.rockylinux.org
    kdc = dc2.rockylinux.org
}
[domain_realm]
  .rockylinux.org = ROCKYLINUX.ORG
```

- Mediante el gestor de paquetes de Python:

```
pip3 install pywinrm
pip3 install pywinrm[credssp]
pip3 install kerberos requests-kerberos
```

10.5 Trabajar con módulos IP

Lo módulos de red normalmente requieren el uso del módulo de python `netaddr`:

```
sudo pip3 install netaddr
```

10.6 Generación de una CMDB

Se ha desarrollado una herramienta, `ansible-cmdb` para generar una CMDB desde ansible.

```
pip3 install ansible-cmdb
```

Los "facts" se deben exportar mediante ansible con el siguiente comando:

```
ansible --become --become-user=root -o -m setup --tree /var/www/ansible/cmdb/
out/
```

A continuación, puede generar un archivo `json` global:

```
ansible-cmdb -t json /var/www/ansible/cmdb/out/linux > /var/www/ansible/cmdb/
cmdb-linux.json
```

Si prefiere una interfaz web:

```
ansible-cmdb -t html_fancy_split /var/www/ansible/cmdb/out/
```

https://docs.rockylinux.org/