Sintaxis Ansible Preview

> Documentado por Andrés Ruslan Abadías Otal | Nisamov

▶ Contenido y Estructura de los Inventarios

Contenido y Estructura de los Inventarios:

En el inventario, las direcciones los servidores pueden ser tanto por DNS como por Dirección IP, por ello es posible indicarlas de ambas formas. El nombre del grupo donde están esas direcciones agrupadas se encuentra entre "[]", lo que posteriormente podemos utilizar para llamar a esa agrupación de direcciones y asignarle un playbook.

Los hosts pueden estar en diferentes grupos, segun el rol del host, su ubicación fisica, ya sea en producción o no y demás bariables, lo que permite aplicar dichas playbooks a conjuntos especificos del host en funcón de sus propóstos, características o ubicación fisica.

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db-servers]
db1.example.com
db2.example.com

[development]
192.0.2.42
```


▶ Definición de grupos Anidados

Definición de grupos Anidados:

Los inventarios de Ansible pueden incluir grupos de grupos de hosts, con lo que podemos crear grupos que aniden grupos en su interior, uniendo asi dos grupos declarados previamente, en un solo grupo, pero facilitando la posterior llamada individual a estos mismos grupos.

Para lograr esto, usamos el sufijo `":children"`, se aplica siguiendo el siguiente ejemplo:

```
[administracion_red]
192.168.1.12
administracion.admred.com
192.168.1.22

[oficinas_p1]
192.168.0.44
192.168.0.45
```

```
192.168.0.46

[full-enterprise:children]
administracion_red
oficinas_p1
```

▶ Definición de Hosts con Rangos

Definición de Hosts con Rangos:

Ansible cuenta con un sistema de especificación de rangos para los hosts de forma que permite especificar rangos numericos o alfabéticos gracias a la siguiente sintaxis: `[START:END]`.

Esto permitirá elegir u rango sin tener que escribir cada una de las direcciones, facilitando y automatizando la tarea en grandes cantidades.

Este sistem pede ser visto de la siguiente manera:

```
[red_local]
192.168[1:9].[0.255] # En este caso coincide con todas las direcciones IPv4 de la
192.168.1.0 a la 192.168.9.255

[lista_servidores]
server[01:25].example.com # En este caso, coincidirá con los hosts del
server01.example.com al server25.example.com

[direcciones_dns]
[a:c].dns.example.com #En este caso, coincidirá con todos los hosts denominados
(a,b y c).dns.example.com

[direcciones_ipv6]
2001:db8::[a:c] #En este caso coincidirá con todas las direcciones IPv6 de la
2001:db8::("a" a la "c")
```

```
- [CUIDADO] - Posibles problemas a la hora de llamar direcciones.

En el ejemplo "lista_servidores", las direccines no coincidirán con
"server2.example.com", pero si lo harán con "server02.example.com"
```

Esto indicado anteriormente es importante a la hora de saber que direcciones necesitamos manejar.

▶ Administración de Ficheros de Configuración

Administración de Ficheros de Configuración:

Para gestionar la configuración de Ansible, podemos crear un fichero `.cfg`, este permitirá aplicar configuraciones a varias herramientas de Ansible.

El fichero de configurción contiene parámetros definidos como `clave-valor` con los siguientes parámetros:

Aqui cuentas con un ejemplo típico de un fichero de configuración básico (`ejemplo.cfg`):

```
[defaults]
inventory = ./inventario # Ubicacion del inventario
remote_user = usuario    # Usuario remoto
ask_pass = false         # Solicitar clave de acceso

[escalado_de_privilegios]
become = true            # Permitir escalado de privilegios
become_method = sudo     # Metodo de escalado
become_user = root       # Usuario de escalado
become_ask_pass = false  # Solicitar clave de acceso al escalar
```

▶ Estructura y Formato de un Playbook

Estructura y Formato de un Playbook:

Un playbook es un fichero escrito en formato `YAML` que se guarda generalmente con la extensión `.yml`.

El playbook usa un sistema de espaciados para indicar la estructura de los datos almacenados, `YAML` no establece ningun requisito sobre cuántos espacios se usan para la sangría pero se aplican las siguientes reglas para su correcto funcionamiento:

- Los elementos de los datos deben estar en el mismo nivel de la estructura con la misma sangría.
- Los elementos secundarios deben tener más sangría que los elementos previos.

En el playbook se comienza con una linea formada por tresguiones consecutivos (`---`) marcando el inicio del documento, del mismo modo, se usan tres puntos consecutivos (`...`) para marcar el final del documento, no obstante, es una práctica comúnmente omitida.

Un ejemplo de un playbook puede ser el siguiente:

```
- name: Escribir nombre de playbook
  hosts: Nombre_Grupo_Servidores_Asignados
  become: yes
  tasks:
    - name: Nombre de Tarea
      apt: # Usas apt para instalar
        name: git # instalar el paquete git
        state: present # aseguramos que lo has descargado correctamente

    - name: Obtener el directorio del usuario actual
      ansible.builtin.set_fact:
        user_home: "{{ ansible_env.HOME }}"

    - name: Clonar el repositorio SSP
      git:
        repo: https://github.com/Nisamov/ssp
```

```
        dest: "{{ user_home }}/ssp"
        update: yes

    - name: Ejecutar comandos dentro del repositorio clonado
      command: ./install.sh
      args:
        chdir: "{{ user_home }}/ssp"
```

En el ejemplo de playbook anterior se ejecuta el siguiente codigo:

- 1: Creación de task (Nombre de Tarea). Instala git y se asegra de su correcta instalación.
- 2: Obtención de usuario actual. Guarda el nombre del usuario y ru tua "/home" actual en la variable "user_home".
- 3: Clonación de repositorio SSP. Clona el repositorio SSP y lo guara dentro del "/home" almacenado previamente.
- 4: Ejecución de instalador. Ejecuta el instalador de SSP.

> [NOTA]: SSP (Secure Service Protocol) es un servicio creado por Andrés Ruslan Abadías Otal, el cual detiene todos los servicios del sistema que no se encuentren dentro de la "whitelist", es un servicio que pude ser peligroso, se recomienda su uso con cuidado.

Iniciación en Playbooks:

Los playbooks son una lista de ordenes organizados de tal forma de ejecuten listas de forma ordenada.

Para cmenar con los playbooks es necesario crear unfichero `.yml` donde aplicar claves, siendo estas las siguientes asignaciones:

```
---
name: nombre del playbook
hosts: nombre_inventory
...
```

- `name`: Name nos permite asignar un nombre, puede o no relacionarse con el playbook, pues sirve como etiqueta o identificador.
- `hosts`: Hosts nos permite asignar las siguientes tareas a un grupo de direcciones creadas en el fichero `inventory.ini`.

Para crear un item dentro de una lista, usamos guiones de la siguiente forma

```
- ejemplo
- item2
- item3
```

Esto nos permite saber la estructura a seguir a la hora de crear una tarea (`task`) de la siguiente manera:

```
--- # Inicio del fichero
name: Ejemplo tasks # Nombre ejemplo
hosts: direccion_ejemplo # Hosts que usarán las tareas
task: # Tareas
    - ejemplo
    - item2
    - item3
... # Fin del fichero
```

▶ Módulos en los Playbooks

Módulos en los Playbooks:

> `ansible.builtin.user`: Es un módulo que usa los elementos (name, uid y state) para saber información sobre un usuario.

```
tasks:
# En esta tarea garantizamos que el usuario1 cuente con el UID 4000
    - name: Informacion usuario
      ansible.builtin.user:
        - name: usuario1
        - uid: 4000
        - state: present
```


> `ansible.builtin.user`: Es un módulo que usa los elementos (name y state) para gestionar grupos.

```
tasks:
# En esta tarea creamos un grupo con el nombre "nuevo_grupo"
- name: Crear grupo
  ansible.builtin.group:
    name: nuevo_grupo
    state: present
```

> `ansible.builtin.service`: Es un módulo que usa los elementos (name y state) para gesionar servicios en sistemas Unix-like.

```
tasks:
# En esta tarea garantizamos que el servicio ssp.service estén en ejecución
    - name: Servicio SSP
      ansible.builtin.service:
        name: ssp
        enabled: true
```

> `ansible.builtin.systemd`: Es un módulo que usa los elementos (name y state) para gestionar servicios que usan `systemd`.

Podemos usa los siguientes parámetros para manejar los servicios con mayor precisión:

- `started`: Inicia el servicio.
- `stopped`: Detiene el servicio.
- `restarted`: Reinicia el servicio.
- `reloaded`: Recarga la configuración sin reiniciar.
- `absent`: Desactiva y elimina el servicio.

```
tasks:
# En esta tarea reiniciamos el servicio nginx
- name: Restart a systemd service
  ansible.builtin.systemd:
    name: nginx
    state: restarted
```

> `ansible.builtin.apt`: Es un módulo que usa los elementos (name y state) para gestionar paquetes usando APT.

```
tasks:
# En esta tarea instalamos nginx
- name: Instalar paquete Nginx en Ubuntu/Debian
  ansible.builtin.apt:
    name: nginx
    state: present
```


> `ansible.builtin.yum`: Es un módulo para CentOS/RedHat que usa los elementos (name y state) para gestionar paquetes usando YUM.

```
tasks:
# En esta tarea instalamos nginx uando YUM
- name: Instalar paquete en RedHat/CentOS
  ansible.builtin.yum:
    name: httpd
    state: present
```

> `ansible.builtin.copy`: Es un módulo que usa los elementos (src y dest), permitiendo copiar archivos a nuevas direcciones.

```
tasks:
# En esta tarea copiamos un fichero a una ruta diferente
- name: Copiar un archivo local
  ansible.builtin.copy:
    src: /ruta/local/archivo.txt
    dest: /ruta/remota/archivo.txt
```

> `ansible.builtin.file`: Es un módulo que usa los elementos (pat, state y mode) para la gestión de archivos o directorios.

```
tasks:
# En esta tarea creamos un directorio con los permisos 0755
- name: Crear un directorio
  ansible.builtin.file:
    path: /ruta/directorio
    state: directory # Indica que se cree un directorio
    mode: '0755'
# En esta tarea creamos un fichero vacio con los permisos 0644
- name: Crear un fichero
  ansible.builtin.file:
    path: /ruta/fichero.txt
    state: touch #Indica que se cree un fichero vacio
    mode: '0644'
# En esta tarea eliminamos un fichero
- name: Eliminar un fichero
  ansible.builtin.file:
    path: /ruta/fichero.txt
    state: absent #Indica la eliminación de archivo o directorio
# En esta tarea eliminamos un directorio y todo su contenido
- name: Eliminar un directorio
  ansible.builtin.file:
    path: /ruta/directorio
    state: absent
```


> `ansible.builtin.template`: Es un módulo que usa los elementos (src y dest) para gestionar plantillas jinja2.

```
tasks:
# En esta tarea, copiamos la plantilla reemplazamos las variables
- name: Copiar plantilla y reemplazar variables
  ansible.builtin.template:
    src: plantilla.j2
    dest: /ruta/destino/archivo.conf
```

> `ansible.builtin.iptables`: Es un módulo que usa los elementos (chain source y jump) para gestionar las reglas de las iptables.

[CONSEJO] Para obtener más informacion sobre las IPTables, puedes ir a la documentación compartida por Andrés Ruslan Abadías Otal, haciendo clic aquí.


```
tasks:
# En esta tarea, creamos una regla de firewall, donde aceptamos el tráfico con la
ip 192.168.1.0
- name: Añadir una regla de firewall
  ansible.builtin.iptables:
```

```
    chain: INPUT
    source: 192.168.1.0/24
    jump: ACCEPT
# En esta tarea, creamos una regla de firewall, donde denegamos el tráfico con la
ip 192.168.1.100
- name: Añadir una regla de firewall
  ansible.builtin.iptables:
    chain: INPUT
    source: 192.168.1.100/24
    jump: DROP
# En esta tarea, eliminamos una regla del firewall
- name: Eliminar una regla de firewall
  ansible.builtin.iptables:
    chain: INPUT
    source: 192.168.1.0/24
    jump: ACCEPT
    state: absent # Indicamos con absent que la regla ha de eliminarse
```

> `ansible.netcommon.network_config`: Es un módulo que usa los elementos (lnes y provider) para gestionar la configuración de red.

```
tasks:
#En esta tarea cambamos la configuracion de red
- name: Aplicar configuración de red
  ansible.netcommon.network_config:
    lines:
      - interface eth0
      - ip address 192.168.1.10/24
    provider: ansible.netcommon.cli
```

> `community.mysql.mysql_db`: Es un módulo que usa los elementos (name y state) para gestionar bases de datos MySQL.

```
tasks:
# En esta tarea creamos una base de datos MySQL
- name: Crear una base de datos MySQL
  community.mysql.mysql_db:
    name: base_datos
    state: present
```

> `community.postgresql.postgresql_db`: Es un módulo que usa los elementos (name y state) para gestionar bases de datos PostgreSQL.

```
tasks:
# En esta tarea creamos una base de datos PostgreSQL
- name: Crear una base de datos PostgreSQL
  community.postgresql.postgresql_db:
```

```
    name: base_datos
    state: present
```

> `ansible.builtin.mount`: Es un móulo que usa los elementos (path, src, fstype y state) para gestionar puntos de montaje.

```
tasks:
# En esta tarea montamos un sistema de archivos dentro de /mnt/disco
- name: Montar un sistema de archivos
  ansible.builtin.mount:
    path: /mnt/disco
    src: /dev/sdb1
    fstype: ext4
    state: mounted
```

> `ansible.builtin.shell`: Es un módulo que permite ejecutar comandos en el sistema usando la shell.

[CONSEJO] Para obtener más información sobre los posibles comandos que pueden usarse en el sistema (Casos Linux), haz clic aquí.

```
task:
# En esta tarea guardamos informacion dentro de un fichero
- name: Ejecutar un comando
  ansible.builtin.shell: "echo 'Hola Mundo' > /tmp/hola.txt"
```


> `ansible.builtin.command`: Es un módulo que usa los elementos (cmd), permitiendo ejecutar comandos en el sistema sin usar la shell.

```
tasks:
# En esta tarea ejectamos el comando "whoami"
- name: Ejecutar un comando básico
  ansible.builtin.command:
    cmd: whoami
```

▶ Implementación de Plays Indefinidos

Implementación de Plays Indefinidos:

es posible implementar diferentes plays en un solo playbook, lo que permite ejeuctar muchas tareas en diferentes hosts, los plays se escriben como un item más de la lista, de la siguiente forma:

```
---
- name: Instalar y habilitar Nginx en los servidores web
  hosts: webservers
```

```
  become: true
  tasks:
    - name: Instalar Nginx
      ansible.builtin.yum:
        name: nginx
        state: present

    - name: Iniciar y habilitar Nginx
      ansible.builtin.systemd:
        name: nginx
        state: started
        enabled: true

- name: Ejecutar un comando Bash en los servidores de base de datos
  hosts: dbservers
  become: true
  tasks:
    - name: Crear un archivo usando Bash
      ansible.builtin.shell: |
        echo "Este es un archivo de prueba" > /tmp/archivo_prueba.txt
...
```

▶ Sintaxis Playbook Ansible

Sintaxis Playbook Ansible:

> **Comentarios**: Los playbooks permiten el uso de comentarios dentro de ellos, esto puede lograrse mediante dos diferentes maneras:

- El uso de "#" (Almohadilla), de manera similar a Bash.
  - `# Esto es un comentario`
  - `Contenido sin comentario # Contenido comentado`
- El uso de ";" (Punto y Coma), este método al igual que la almohadilla, permite comentar toda la linea.

> **Cadenas de texto (Strings)**: Generalmente no es necesario colocar comillas para las cadenas de texto, se pueden incluir entre comillas dobles o simples.

- Ejemplo sin comillas:
  - `Esto es una cadena de texto`
- Ejemplo comillas simples:
  - `'Esto es una cadena de texto'`
- Ejemplo comillas dobles:
  - `"Esto es una cadena de texto"` Es posbl escribir cadenas de varias lineas gracias a la barra vertical ["|"] (pipe), de la siguiente froma, podemos incluir más carácteres:

```
include_newlines: |
    Ejemplo linea 1
    Ejemplo linea 2
    Ejemplo linea 3
```

No solo usando la barra vertical podemos realizar esto, con el carácter mayor ">", podemos indicar que los caracteres de nueva linea han de ser convertidos en espacios y se deben quitar espacios en blanco dentro de las lineas.

Este método suele usarse para convertir largas cadenas en carácteres, pudiendo abarcar múltiples lineas.

```
fold_newlines: >
    Ejemplo linea 1
    Ejemplo linea 2
    Ejemplo linea 3
```

> **Diccionarios**: Un diccionario permite almacenar informacion en formato de lista , de donde posteriormente se podrán sacar los datos.

Un ejemplo de diccionario es el siguiente:

```
# Primer formato de escritura - sencillo y facil para humanos
name: ejemplo
example: example2
example3: example4
```



```
# Segundo formato de escritura horizontal - dificil para humanos
{name: ejemplo, example: example2, example3: example4}
```

> **Listas**: Las listas de la misma forma que los diccionarios cuentan con una estructura similar, con algunos cambios.

```
# Primer formato de escritura - sencillo y facil para humanos
ejemplo:
    - ejemplo1
    - ejemplo2
    - ejemplo3
```

```
#Segundo formato de escritura horizontal - dificil para humanos
ejemplo: [ejemplo1, ejemplo2, ejemplo3]
```

▶ Gestión de Variables y Datos

Gestión de Variables y Datos:

Las variables permiten almacenar parámetros para llamarlos de manera indefinida, ahorrando tiempo y sencillez a la hora de crear código.

Las variables son una gran herramienta para cualquier lenguaje de programación.

Las variables pueden almacenar diferente contenido, un ejemplo de los posibles valores a almacenar son los siguientes:

- Usuarios (Usuarios a gestionar)
- Paquetes (Paquetes a instalar/desinstalar)
- Servicios (Servicios que iniciar o detener)
- Archivos (Archivos que crear, eliminar, mover...)
- Direcciones (IP/DNS) (Direcciones a conectarse)

Para declarar variables hay que tener en cuenta su estrcutura, estas no pueden contener espacios dentro de ellas, ni carácteres no permitidos.

Nombres No Válidos	Nombres Válidos
ejemplo inicio	ejemplo_inicio
ejemplo.punto	ejemplo_punto
1er fichero	fichero_01
ejemplo$1	ejemplo_1

> **Definición de Variable en Playbook**: Podemos definir variables dentro de playbooks de diferentes maneras, las cuales son las siguientes.

- Bloque `vars` a comienzo de un play:

```
# Interior de playbook
- hosts: all
  vars:
    user: usuario       # Hemos declarado la variable "user" con el valor
"usuario"
    home: /home/usuario # Hemos declarado la variable "home" con el valor
"/home/usuario"
```

- Podemos definir vairables de archivos exernos para no usar el bloque `vars`.

```
# Interior de playbook
- hosts: all
  vars_files:
    - vars/users.yml    # Las variables se encuentran en el fichero de la ruta
"vars/users.yml"
```

```
# Interior de users.yml
user: usuario
home: /home/usuario
```

> **Referencia a Variable**: Tras haber declarado las variables, se llaman a estas mismas (haciendo referencia), mediante el uso de llaves dobles "{{ }}", lo que permite que Ansible reemplace la variable con su valor original al ejecutar la tarea.

```
vars:
    user: ejemploUsuario

tasks:
    # Esta linea será leida como "Crea el usuario ejemploUsuario"
    - name: Crea el usuario {{ user }}
      user:
        # Con esta declaración, creamos el usuario "ejemploUsuario"
        name: "{{ user }}"
```

Cuando se usa una variable como el primer elemento para iniciar un valor, es obligatorio poner comillas, de esta evitamos que Ansile crea que queremos iniciar un diccionario YAML (Visto anteriormente).

En caso de no hacerlo, sacaría un error similar al siguiente:

```
- We could be wrong, but this one looks like it might be an issue with mising
quotes.
- Always quiote template expression brackets when they start a value.
```


> **Variables de Hosts y Variables de Grupos**: Estas variables pueden ser tanto de host como de grupo.

Pra poder comprender las variables de host individual y las de grupo compartidas, usaremos los siguientes ejemplos:

```
#Variable individual
[servidorEjemplo]
192.168.1.10 usuario=usuario1 #Variable asignada a un solo equipo

#Variable de grupo
[servidoresEjemplo]
192.168.1.[20:50] # Rango de IPs a las que se le asignaran la variable

[servidoresEjemplo:vars] # Grupo que asigna las variables a servidoresEjemplo
usuarioCompartido=usuarioComun
```

▶ Gestión de Claves

Gestión de Claves:

Puede darse el caso en el que necesitemos gestionar claves o datos importantes para configurar determiados hosts, por lo que pare evitar arriesgar la información que generalmente se encuentra almacenada como texto

sin formato, Ansible cuenta con una herramienta para cifrar y descifra cualquier archivo de datos.

Se trata de Vault, una herramienta que viene con Ansible, puede ser ejecutada con el comando `ansible-vault`, lo que nos permtirá crear, editar, cifrar, descifrar y ver archivos.

> **Creación de cifrado**: Para crear un cifrado, usaremos el comando `ansible-vault create <nombre fichero>` de la siguiente forma:

```
ansible-vault create ficherocifrado1
```

Si es la primera vez que se usa vault, este solicitará una clave de acceso con la que podremos usar vault posteriormente de la siguiente forma:

```
ansible-vault create ficherocifrado1
New Vault password: clavedeacceso
Confirm New Vault password: clavedeacceso
```

Si no se quiere usar los métodos tradicionales, Ansible permite pasarle a Vault la clave por medio de un fichero de la siguiente manera:

```
# Comando a pasar
ansible-vault create --vault-password-file=vault-pass misecreto.yml
```

```
# Interior de misecreto.yml
clavedeacceso
```

El codigo para proteger archivos es AES256 en las versiones más recientes de Ansible, no obstante las versiones previas podían llegar a usar AES de 128 bits.

> **Ver contenido de archivo cifrado**: Para ver el contenido de un archivo protegido, usamos `ansible-vault view <fichero>`.

```
# Abrir archivo protegido
ansible-vault view misecreto.yaml
Vault password: clavedeacceso
misecreto: "clavedeacceso"
```

> **Edición de arhivo cifrado existente**: Para editar un archivo cifrado existente, usamos el comando `ansible-vault edit <archivo>`.

```

```

```
ansible-vault edit misecreto.yml
Vault password: clavedeacceso
```

> **Cifrar un archivo existente**: Cifrar un archivo existente, usamos el comando `ansible-vault encrypt <archivo>`, esta acción solicitará una nueva clave de acceso ya que se está cifrando nuevamente el fichero con una nueva clave de acceso (No se cifra dos veces como capas (layers)).

```
# Ciframos dos ficheros
ansible-vault encrypt misecreto.yml ficheroacifrar.yml
# Creamos una nueva clave de acceso
New Vault Password: clavedeacceso2
Confirm New Vault password: clavedeacceso2
Encryption successful
```

> **Descifrar un archivo existente**: Para descifrar un archivo de forma permanente, usamos el comando `ansible-vault decrypt <archivo>` con la opción `--output` podemos guardar el archivo descifrado con un nombre diferente.

```
ansible-vault decrypt misecreto.yml --output=misecretodesencriptado.yml
Vault password: clavedeacceso2
Decryption successful
```


> **Camiar clave de acceso de un archivo cifrado**: Para cambiar la clave de acceso en un archivo cifrado, usamos el comando `ànsible-vualt rekey <archivo>`, este comando permite el cambio de varios archivos al mismo tiempo.

```
ansible-vault rekey misecreto.yml midocumento.yml documento2.yml
Vault password: clavedeacceso2
Confirm New Vault password: clavedeacceso2
Rekey successful
```

Para pasar una clave por ficheros usamos el siguiente comando lineal:

```
ansible-vault rekey \
--new-vault-password-file=NuevaClaveDeAcceso misecreto.yml midocumento.yml
documento2.yml
```

▶ Claves de Acceso con Playbooks

Claves de Acceso con Playbooks:

> **Acceso a un archivo cifrado desde un Playbook**: Para poder acceder a un archivo cifrado, es necesario proporcionar la clave de acceso al comando `ansible-navigator`.

[CONSEJO] Para obtener más informacion sobre Ansible Navigator, puedes ir a la documentación compartida por Andrés Ruslan Abadías Otal, haciendo clic aquí.

```
ansible-navigator run -m studut misecreto.yml
- ERROR! Attempting to decrypt but no vault secrets found
```

Para poder proporcionarle la clave a una Playbook contamos con tres opciones diferentes:

- Indicar de forma interactiva

```
ansible-navigator run -m stdout \
#Deshabilitamos los artefactos para ingresar la clave de Vault
--playbook-artifact-enable false \ # Los artefactos del playbook están deshabilitados de forma predeterminada
# Podemos volver a habilitar los artefactos con el siguiente comando: ansible-navigatoe --playbook-artifact-enable true
playbook.yml --vault-id @prompt
Vault password (default): clavedeacceso2
```


- Especificar el archivo de clave Vault con `--vault-password-file`

```
ansible-navigator run -m stdout playbook.yml \ # Indicamos el playbook que queremos que tenga acceso
--vault-password-file=/ruta/al/fichero.yml # Indicamos la ruta al fichero que alberga la clave
```

- Usar la variable de entorno `ANSIBLE_VAULT_PASSWORD_FILE`