Primeros pasos con módulos y comandos ad-hoc

Concectarse a nuestra primera máquina

Documentación módulo ping

Creamos el inventario. En este punto tenemos que añadir las IPs que tengan nuestras máquinas. En caso de tener servidor DNS podemos no poner las IPs, en este momento lo haremos con IPs.

```
ansible@ControlNode:~$ mkdir practicas/pr01 && touch practicas/pr01/maquinas
ansible@ControlNode:~$ cd practicas/
ansible@ControlNode:~/practicas/pr01$ ls
maquinas
ansible@ControlNode:~/practicas/pr01$ cat maquinas
srv1 ansible_host=192.168.10.6 ansible_user=ansible ansible_become=yes
```

Lanzamos nuestro comando con el comando **ansible**.

```
ansible -i [inventario] [máquina destino] -m [módulo]
```

- **inventario**: Fichero donde definimos el conjunto de máquinas.
- **máquina destino**: Conjunto de máquinas del inventario que vamos a gestionar.
- **módulo**: Conjunto de funcionalidades que vamos a emplear. Podemos consultar información de los módulos del sistema en el siguiente apartado de la documentación técnica.

Nota: Recordemos que anteriormente hemos creado el par de claves y podemos hacer ssh a nuestra máquina *server1*: **ssh 192.168.10.6**.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m ping
srv1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false,
    "ping": "pong"
}
```

La salida indica que el comando Ansible logró conectarse correctamente con el host 192.168.122.6 usando el módulo **ping** y no hubo cambios en el sistema remoto.

Interpretación de cada parte

- **192.168.122.6 | SUCCESS**: El host respondió correctamente y es accesible para Ansible.

- **"ansible_facts": { "discovered_interpreter_python": "/usr/bin/python3.11" }**: Ansible detectó que usará el intérprete de Python ubicado en `/usr/bin/python3.11` para ejecutar comandos en ese equipo. Posteriormente volveremos sobre este concepto pero por el momento, son propiedades que se preguntan a la máquina de destino.
- **"changed": false**: El módulo ping no realiza cambios, solo verifica conectividad, por eso el valor es "false".
- **"ping": "pong"**: Esta respuesta confirma que el host está disponible y preparado para que Ansible ejecute más automatizaciones.

Concectarse a varias máquinas

Nota: Crearemos una clonación enlazada de nuestra plantilla de máquinas servidor y añadiremos el reenvío de puertos necesario.

```
ansible@ControlNode:~/practicas/pr01$ cat maquinas
[servers]
srv1 ansible_host=192.168.10.6 ansible_user=ansible ansible_become=yes
srv2 ansible_host=192.168.10.5 ansible_user=ansible ansible_become=yes
```

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts servers -m ping
srv1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false,
    "ping": "pong"
}
srv2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false,
    "ping": "pong"
}
```

- *Nota 5*: También puedo hacer uso de metacaracteres: **ansible -i hosts srv* -m ping**.

Comando ad hoc

Documentación módulo command Documentación módulo shell

No son muy útiles ya que ejecutan de forma rápida una instrucción contra las máquinas remotas. El problema es que estas instrucciones no son reutilizables lo que va en contra de la filosofía de ansible de tener playbooks que puedan administrar múltiples máquinas.

- *Nota 1*: Estos dos módulos que no garantizan la *idempotencia* de un playbook.

- *Nota 2*: Idempotencia significa que una tarea o módulo se puede ejecutar varias veces sin cambiar el resultado una vez que el sistema ha alcanzado el estado deseado. Esto asegura que el playbook puede repetirse sin causar efectos secundarios o cambios innecesarios, dejando siempre el sistema en el mismo estado final esperado.

```
ansible -i [inventario] [máquina destino] -m command -a "date"
```

- inventario: Fichero donde definimos el conjunto de máquinas.
- máquina destino: Conjunto de máquinas del inventario que vamos a gestionar.
- command: Me permite indicar los argumentos o comandos a ejecutar en las máquinas destino.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m command -a "date"
srv1 | CHANGED | rc=0 >>
dom 21 sep 2025 10:20:00 CEST
```

- server1: máquina alcanzada.
- CHANGED: Ha entrado dentro y ha realizado un resultado.
- rc=0: result code 0, código resultante de la operación 0, ha sido correcto.
    - rc=0: Operación realizada con éxito.
    - rc=1: Cambio en el sistema.
    - rc=2: Error de ejecución.
- dom 21 sep 2025 10:20:00 CEST: Resultado del comando.

Otros ejemplos.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m command -a "touch
/tmp/ficheroPrueba.txt"
server1 | CHANGED | rc=0 >>

ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m command -a "ls -l
/tmp/ficheroPrueba.txt"
server1 | CHANGED | rc=0 >>
-rw-r--r-- 1 ansible ansible 0 sep 21 10:25 /tmp/ficheroPrueba.txt
```

Podemos tener problemas como el uso de caracteres especiales, pipes o tuberias... Esto es por el uso del módulo *command* que no es capaz de interpretar estos caracteres. Para ello tenemos que hacer uso del módulo *shell* que hace uso de una shell */bin/bash*.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m command -a "ls -l
/tmp/ | grep ficheroPrueba.txt"
server1 | FAILED | rc=2 >>
/tmp/:
total 12
drwx------ 2 ansible ansible 4096 sep 21 10:29
```

```
ansible_ansible.legacy.command_payload_49w97c2k
-rw-r--r-- 1 ansible ansible    0 sep 21 10:25 ficheroPrueba.txt
drwx------ 3 root    root    4096 sep 21 09:18 systemd-private-
fd2de44b5eef4036a99cf06fa06932c2-systemd-logind.service-3SiAa4
drwx------ 3 root    root    4096 sep 21 09:18 systemd-private-
fd2de44b5eef4036a99cf06fa06932c2-systemd-timesyncd.service-BqNRihls: no se puede
acceder a '|': No existe el fichero o el directorio
ls: no se puede acceder a 'grep': No existe el fichero o el directorio
ls: no se puede acceder a 'ficheroPrueba.txt': No existe el fichero o el
directorionon-zero return code

ansible@ControlNode:~/practicas$ ansible -i hosts srv1 -m shell -a "ls -l /tmp/ |
grep ficheroPrueba.txt"
server1 | CHANGED | rc=0 >>
-rw-r--r-- 1 ansible ansible    0 sep 21 10:25 ficheroPrueba.txt
```

Tabla comparativa entre los módulos **shell** y **command**:

Característica	Módulo command	Módulo shell
Ejecución	Ejecuta el comando directamente sin shell	Ejecuta el comando a través de un shell (/bin/bash)
Soporte para funcionalidades de shell	No soporta pipes, redirecciones, expansiones, operadores lógicos	Soporta pipes (
Seguridad	Más seguro porque no ejecuta shell, menos riesgo de inyección	Menos seguro, usa shell que puede ser vulnerable si se abusa
Uso recomendado	Para comandos simples sin características de shell	Para comandos complejos que necesitan funcionalidades del shell
Expansión de comodines (ej. *)	No expande comodines	Expande comodines correctamente
Ejemplo	`ansible -m command -a "ls -l"`	`ansible -m shell -a "ls -l *"`
Resultado esperado	Más predecible, evita efectos secundarios	Permite comandos más complejos e interacciones con shell

Módulo setup o gather_facts para recuperar información del sistema

Documentación módulo setup Documentación módulo gather_facts

En Ansible, el módulo **setup** es un módulo muy importante que se utiliza para recopilar y devolver toda la información del sistema (facts) del nodo remoto al que se conecta Ansible. Estos facts son datos sobre el sistema objetivo, como detalles del hardware, configuración de red, sistema operativo, interfaces de red, espacio en disco, memoria, y más. Actualmente es preferible hacer uso del nuevo módulo **gather_facts** pero es cierto que existe retrocompatibilidad entre ambos y los dos se pueden emplear para lo mismo.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m setup -a
"filter=ansible_distribution,ansible_distribution_version,ansible_os_family"
srv1 | SUCCESS => {
    "ansible_facts": {
        "ansible_distribution": "Debian",
        "ansible_distribution_version": "12.12",
        "ansible_os_family": "Debian",
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false
}
```

Estos *facts* serán empleados en futuros playbooks ya que puede interesarnos saber el sistema operativo de una máquina para realizar un acción concreta u otro dato. Podemos ver una propiedad en concreto si filtramos aquello que queremos ver.

```
ansible@ControlNode:~/practicas/pr3$ ansible -i hosts server1 -m setup -a
"filter=ansible_architecture,ansible_all_ipv4_addresses"
server1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.122.6"
        ],
        "ansible_architecture": "x86_64",
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false
}
```

Además, el módulo más extendido en estos momentos, que presenta el mismo comportamiento es **gather_facts**.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m gather_facts -a
"filter=ansible_distribution,ansible_distribution_version,ansible_os_family"
srv1 | SUCCESS => {
    "ansible_facts": {
        "ansible_distribution": "Debian",
        "ansible_distribution_version": "12.12",
        "ansible_os_family": "Debian",
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false
}
```

Copiando ficheros con el módulo copy

Documentación módulo copy

El módulo **copy** en Ansible es utilizado para copiar archivos o directorios desde el nodo de control (local) hacia un nodo remoto administrado. Permite transferir archivos y configurar metadatos del sistema de archivos como permisos, propietario y grupo del archivo copiado.

Haciendo uso del módulo **copy** enviamos el archivo *copy.txt* como *copia.txt* en la máquina de destino.

```
ansible@debian64:~/practicas/pr01$ echo "Fichero a enviar" > /tmp/copy.txt

ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m copy -a
"src=/tmp/copy.txt dest=/tmp/copia.txt"
srv1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": true,
    "checksum": "629eae3695e16055679be3a315e8061ad32529d9",
    "dest": "/tmp/copia.txt",
    "gid": 0,
    "group": "root",
    "md5sum": "340ec97940c84736e126d9e19c36ff1c",
    "mode": "0644",
    "owner": "root",
    "size": 17,
    "src": "/home/ansible/.ansible/tmp/ansible-tmp-1762381031.4800127-6111-
10117447403728/.source.txt",
    "state": "file",
    "uid": 0
}

ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m shell -a "ls -l
/tmp/copia.txt"
srv1 | CHANGED | rc=0 >>
-rw-r--r-- 1 root root 21 sep  5 10:25 /tmp/copia.txt
```

Tambien podemos relizarlo con directorios.

```
ansible@debian64:~/practicas/pr01$ mkdir -p /tmp/directorio && echo "Directorio a
enviar" > /tmp/directorio/enviar.txt
```

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m copy -a
"src=/tmp/directorio dest=/tmp/dir_enviado"
srv1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": true,
    "checksum": "91df5fccd21e09ded6fa7e999592347e1ed4c175",
    "dest": "/tmp/dir_enviado/directorio/enviar.txt",
```

```
    "gid": 0,
    "group": "root",
    "md5sum": "aeba3fff2290930148f911fa28d209a1",
    "mode": "0644",
    "owner": "root",
    "size": 20,
    "src": "/home/ansible/.ansible/tmp/ansible-tmp-1762381301.370823-6368-
66463491644703/.source.txt",
    "state": "file",
    "uid": 0
}
```

Un parámetro muy empleado es el parámetro *mode*. Este parámetro nos va a permitir establecer los permisos en octal del directorio/fichero.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m copy -a
"src=/tmp/directorio dest=/tmp/dir_enviado mode=755"
srv1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": true,
    "checksum": "91df5fccd21e09ded6fa7e999592347e1ed4c175",
    "dest": "/tmp/dir_enviado/directorio/enviar.txt",
    "gid": 0,
    "group": "root",
    "mode": "0755",
    "owner": "root",
    "path": "/tmp/dir_enviado/directorio/enviar.txt",
    "size": 20,
    "state": "file",
    "uid": 0
}
```

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m shell -a "ls -l
/tmp/dir_enviado"
srv1 | CHANGED | rc=0 >>
total 4
drwxr-xr-x 2 root root 4096 sep 21 12:12 directorio
```

Instalación de paquetes en nuestra máquina debian

Documentación módulo apt

Vamos a realizar la instalación con el módulo **apt** del servicio *Apache*. Este módulo hace uso entre otros de los siguientes parámetros.

- name: Nombre del paquete a instalar.

- state: Indica el estádo del paquete y puede tomar los diferentes valores.
  - *present*: Para asegurarse que el paquete esté disponible (presente) en el sistema sin modificarlo si ya está instalado.
  - *absent*: Para eliminar el paquete y liberar espacio o evitar conflictos. Solo elimina el paquete pero no los archivos de configuración.
  - *latest*: Para mantener el sistema actualizado con la versión más reciente del paquete.
  - *build-dep*: Utilizado principalmente para preparar el entorno de compilación de un paquete desde el código fuente.
- update_cache: Permite actualizar los repositorios del equipo.
- purge: Permite purgar el paquete eliminado los archivos de configuración.
- autoremove: Permite eliminar paquetes huerfanos que fueron instalados como dependencias.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m apt -a "name=apache2
state=present update_cache=yes"
...

ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m shell -a "systemctl
status apache2"
srv1 | CHANGED | rc=0 >>
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset:
enabled)
     Active: active (running) since Wed 2025-11-05 23:39:19 CET; 38s ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 3930 (apache2)
      Tasks: 55 (limit: 1108)
     Memory: 9.2M
        CPU: 50ms
     CGroup: /system.slice/apache2.service
             ├─3930 /usr/sbin/apache2 -k start
             ├─3932 /usr/sbin/apache2 -k start
             └─3933 /usr/sbin/apache2 -k start

nov 05 23:39:18 debian64 systemd[1]: Starting apache2.service - The Apache HTTP
Server...
nov 05 23:39:19 debian64 systemd[1]: Started apache2.service - The Apache HTTP
Server.
```

En estos momentos con un navegador podremos ver apache. En caso de hacer uso de VirtualBox añadiendo un reenvío de puertos podremos acceder a la ruta a apache.

Llegados a este punto podemos desinstalar apache2 del equipo.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m apt -a "name=apache2
state=absent"
...
```

O puedo purgarlo completamente eliminado los ficheros de configuración así como paquetes huérfanos que ya no tienen ningún paquete que dependa de ellos.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m apt -a "name=apache2
state=absent autoremove=yes purge=true"
```

Nota: Conectados en el propio nodo *srv1* podríamos hacerlo del siguiente modo.

```
$ ssh server1
$ apt list --installed apache2
Listando... Hecho
apache2/oldstable,now 2.4.65-1~deb12u1 amd64 [instalado]

$ sudo apt purge -y apache2
$ apt list --installed apache2
Listando... Hecho

$ dpkg -l apache2
dpkg-query: no se ha encontrado ningún paquete que corresponda con apache2.
```

Nota: Para distribuciones de tipo Red Hat tenemos el *módulo dnf*.

Documentación módulo dnf

Módulo service para la gestión de servicios

Documentación módulo service

El módulo **service** en Ansible se utiliza para gestionar servicios en sistemas operativos remotos, como arrancar, parar, reiniciar o habilitar servicios. Es un módulo que actúa como proxy para el gestor de servicios subyacente del sistema (systemd, sysvinit, etc.) del host remoto. Los principales parámetros del módulo service son:

- **name**: Nombre del servicio a gestionar (por ejemplo, `apache2`, `nginx`, `sshd`). Este parámetro es obligatorio.
- **state**: Define el estado deseado del servicio. Puede tener valores como:
  - `started`: Arranca el servicio si no está en ejecución.
  - `stopped`: Detiene el servicio si está en ejecución.
  - `restarted`: Reinicia el servicio.
  - `reloaded`: Recarga la configuración del servicio sin reiniciarlo.
- **enabled**: Booleano (`yes`/`no`) que indica si el servicio debe estar activo en el arranque del sistema (usando systemctl enable, por ejemplo).

Ejemplos de uso del módulo service

Volvemos a instalar el servicio *apache2*.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m apt -a "name=apache2
state=present update_cache=yes"
```

- Parar un servicio:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m service -a
"name=apache2 state=stopped"
```

- Iniciar un servicio:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m service -a
"name=apache2 state=started"
```

- Reiniciar un servicio:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m service -a
"name=apache2 state=restarted"
```

- Habilitar un servicio para que arranque con el sistema:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m service -a
"name=apache2 state=restarted enabled=yes"
```

Este módulo es fundamental para la administración de servicios en entornos automatizados con Ansible, facilitando la gestión y control de los mismos de forma simple y declarativa.

Módulo file para la gestión de archivos y directorios

Documentación módulo file

El módulo file de Ansible es una herramienta fundamental para la gestión de archivos y directorios en hosts remotos. Permite crear, eliminar y modificar archivos o directorios, así como gestionar sus permisos, propiedad y enlaces simbólicos. Con este módulo podemos:

- **path**: (Obligatorio) Ruta del archivo, directorio o enlace simbólico a administrar. Ejemplo: `/tmp/archivo.txt`
- **state**: Define el estado deseado del archivo o directorio. Valores posibles:
  - `absent`: Elimina el archivo, directorio o enlace simbólico.
  - `directory`: Crea un directorio.
  - `file`: Verifica que la ruta sea un archivo normal (no crea ni modifica si no existe).
  - `hard`: Crea un enlace físico (hard link).
  - `link`: Crea un enlace simbólico (symlink).

    - `touch`: Crea un archivo vacío si no existe o actualiza su timestamp.
- **owner**: Usuario propietario del archivo/directorio. Ejemplo: `ansible`
- **group**: Grupo propietario del archivo/directorio. Ejemplo: `ansible`
- **mode**: Permisos del archivo/directorio en formato octal. Ejemplo: `0755`
- **src**: (Solo para enlaces) Ruta del archivo original al que apunta el enlace.
- **recurse**: Booleano. Si es `yes` y el estado es `directory`, aplica cambios recursivamente dentro del directorio.

1. Crear el fichero vacío (touch):

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m file -a
"path=/tmp/prueba.txt state=touch"
```

2. Cambiar permisos y propietario del fichero:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m file -a
"path=/tmp/prueba.txt owner=ansible group=ansible mode=0777"
```

3. Crear un directorio:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m file -a
"path=/tmp/directorio state=directory mode=0755"
```

4. Borrar el fichero:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m file -a
"path=/tmp/prueba.txt state=absent"
```

5. Borrar el directorio:

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m file -a
"path=/tmp/directorio state=absent"
```

Módulo user y módulo group para gestionar usuarios y grupos

Documentación módulo user Documentación módulo group

En Ansible, los módulos **user** y **group** se usan para la administración de usuarios y grupos en sistemas gestionados.

El módulo **user** permite crear, modificar o eliminar cuentas de usuario en sistemas Linux, BSD, y otros compatibles. Entre sus funcionalidades principales están:

- Crear un usuario nuevo con opciones para definir nombre, identificador (UID), grupo principal, grupos secundarios, shell de login, directorio home y comentario.
- Modificar atributos de usuarios existentes como contraseña (hash), fechas de expiración de cuenta o contraseña, bloqueo de cuenta, etc.
- Eliminar usuarios y, opcionalmente, eliminar también sus archivos.
- Generar claves SSH para el usuario. Este módulo es idempotente y automatiza tareas comunes de gestión de usuarios con múltiples parámetros para personalización avanzada.

El módulo **group** se encarga de la gestión de grupos locales. Permite:

- Crear grupos nuevos con nombre y GID.
- Modificar atributos de grupos existentes.
- Eliminar grupos.
- Añadir usuarios a grupos secundarios. También es idempotente y muy útil para arreglar la organización de usuarios y permisos por grupos en los sistemas gestionados.

Estos dos módulos son básicos para la administración de cuentas y control de acceso en sistemas Unix-like dentro de automatizaciones con Ansible, ayudando a mantener consistencia y facilidad de administración de usuarios y grupos.

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m group -a "name=smr2
state=present"
```

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m user -a "name=martin
group=smr2 state=present home=/home/martin shell=/bin/bash
password='$y$j9T$L7UsRSemGDLwCnJiK40wG1$M9fvD2ir0hUD0lCVgqnSxOPpdFybFke18cJVxDrsGE
4' create_home=yes"
```

Nota: Para añadirlo a grupos secundarios podemos hacer lo siguiente.

```
ansible -i hosts srv1 -m ansible.builtin.user -a "name=martin groups=sudo,smr2
append=yes state=present home=/home/martin shell=/bin/bash
password='$y$j9T$L7UsRSemGDLwCnJiK40wG1$M9fvD2ir0hUD0lCVgqnSxOPpdFybFke18cJVxDrsGE
4' create_home=yes"
```

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m user -a "name=martin
state=absent"
```

```
ansible@debian64:~/practicas/pr01$ ansible -i hosts srv1 -m group -a "name=smr2
state=absent"
```

Nota: Es importante tener instalado en la máquina el paquete *whois* para generar la contraseña cifrada.

```
ansible@debian64:~/practicas/pr01 sudo apt install -y whois
ansible@debian64:~/practicas/pr01 mkpasswd --method=sha-512
Contraseña:
$y$j9T$L7UsRSemGDLwCnJiK40wG1$M9fvD2ir0hUD0lCVgqnSxOPpdFybFke18cJVxDrsGE4
```

Escalada de privilegios

Es importante aclarar aunque nosotros estámos definiendo en nuestro inventario que el usuario que realiza la conexión puede realizar una escalada de privilegios, podemos realizar esto en la propia ejecución en la terminal con el parámetro *-b*. Este indica que Ansible ejecutará el módulo apt con permisos elevados (sudo), lo cual es necesario para instalar paquetes. En otras palabras, tenemos que indicar que vamos a realizar una escalada de privilegios. Esto lo podemos hacer ya que el usuario ansible tiene permisos de *sudo* en *server1* (a mayores tenemos configurado que no necesita passwd, podríamos solicitarla con el parámetro *--ask-become-pass*).

- Añadir *-b* o **--become** en las llamadas a Ansible cuando la tarea requiera permisos de administrador.
- Verificar que el usuario tiene permisos sudo sin contraseña o preguntar la contraseña si es necesario.
- Alternativamente, configurar en el playbook o inventario la opción become: yes.

```
[servers]
srv1 ansible_host=192.168.10.6 ansible_user=ansible
srv2 ansible_host=192.168.10.5 ansible_user=ansible
```

```
ansible@debian64:~/practicas/pr01$  ansible -i hosts srv1 -m apt -a "name=apache2
state=present update_cache=yes" -b
...
```

```
ansible@debian64:~/practicas/pr01$  ansible -i hosts srv1 -m apt -a "name=apache2
state=present update_cache=yes" --become
...
```

Ejercicios para repasar los conceptos anteriores

Se propone la Tarea 1 para repasar los conceptos vistos hasta este punto.