Proyecto de administración de sistemas informáticos: Mansible

Objetivos

El objetivo de este proyecto práctico de desarrollo en pareja, aunque también puede realizarse de forma individual, es revisar de forma aplicada algunos de los conceptos estudiados en la parte teórica de la asignatura tales como:

- La gestión de usuarios.
- La administración de los dispositivos de almacenamiento.
- El uso de SSH para realizar la administración de equipos remotos.
- Los fundamentos de las técnicas usadas para automatizar la administración.
- El modo de operación de Ansible, que es la herramienta de gestión de configuración más usada.
- La programación de *scripts*.

NOTA: El tema dedicado a Ansible se presenta al final de la asignatura pero para afrontar la práctica no es necesario un conocimiento profundo de esta herramienta, sino que basta con entender, en términos generales, su modo de operación.

Planteamiento general

El trabajo se centra en la creación de una infraestructura de automatización de la administración similar a Ansible, que es la herramienta de gestión de configuración más popular actualmente y que se estudia en la parte teórica de la asignatura. Se trata de una versión muy simplificada, evidentemente, a la que denominaremos *Mansible* (por *mini Ansible*), pero que permite apreciar el fundamento y modo de operación de esta herramienta.

Ansible está organizado como una colección de módulos instalados en la máquina maestra de control que son copiados mediante SCP en las máquinas administradas y ejecutados usando SSH. Con esta herramienta, las operaciones de configuración pueden ejecutarse puntualmente como mandatos *ad hoc* o, lo habitual, englobarse dentro de un *playbook*. A continuación, se presenta un ejemplo de cada caso:

- Un mandato *ad hoc* que, usando el módulo `user`, crea en todas las máquinas gestionadas un usuario llamado `test` sin contraseña y que tiene como *shell* asociado `/bin/bash`:

  ```
  ansible all -b -m user -a "name=test state=present shell=/bin/bash password=''"
  ```

- Un *playbook* que en un grupo de máquinas formatea un cierto disco (`/dev/loop17`), que está presente en todas ellas, con el sistema de ficheros *ext4* y lo monta en determinado directorio (`/mnt/misDatos`) también existente en esos equipos:

  ```
  ---
  - name: Crea un sistema de ficheros de tipo ext4 sobre un disco y lo monta
    hosts: servidores_almacenamiento
    become: true
    tasks:

      - name: Formatea el dispositivo con un SF ext4
        community.general.filesystem:
          fstype: ext4
          dev: /dev/loop17

      - name: Monta el dispositivo sobre el directorio
        ansible.posix.mount:
          path: /mnt/misDatos
          src: /dev/loop17
          fstype: ext4
          state: mounted
  ```

  Este *playbook* contenido en el fichero `ejemplo.yml` se ejecutaría con el mandato `ansible-playbook`:

  ```
  ansible-playbook ejemplo.yml
  ```

En la práctica se plantea, en consecuencia, la implementación de tres tipos de funcionalidades que corresponden a tres fases:

1. Módulos: se tendrán que implementar los módulos correspondientes a los ejemplos que se acaban de mostrar (`filesystem`, `user` y `mount`).
2. El mandato que permite la ejecución de operaciones *ad hoc*, al que denominaremos `mansible`. Solo se puede evaluar esta fase si ha obtenido en la primera al menos 3,5 puntos.
3. El mandato que permite la ejecución de *playbooks*, al que denominaremos `mansible-playbook`. Solo se puede evaluar esta fase si ha obtenido en la primera al menos 3,5 puntos.

Recapitulando, hay que desarrollar los siguientes *scripts* **independientes**:

1. `filesystem` (2 puntos): módulo que crea sistemas de ficheros.
2. `user` (2,25 puntos): módulo que gestiona cuentas de usuario.
3. `mount` (2,5 puntos): módulo que realiza el montaje de dispositivos.
4. `mansible` (1,25 puntos): implementa el mandato que permite ejecutar operaciones *ad hoc*.
5. `mansible-playbook` (2 puntos): implementa el mandato que permite ejecutar *playbooks*.

Los tres primeros corresponden a la primera fase y no requieren su ejecución remota para depurarlos por lo que puede desarrollarlos en un único equipo. Los dos últimos *scripts*, que corresponden a las dos últimas fases, sí requieren usar al menos dos equipos.

Plataforma de desarrollo

La distribución de Linux usada para preparar la práctica ha sido Ubuntu 24.04, aunque no debería de dar problemas utilizar otras versiones de esa misma distribución.

Para el desarrollo de la práctica se presentan varias opciones:

- Máquinas reales. Se desaconseja esta opción dado que los errores que puedan producirse durante el desarrollo de la práctica pueden dejar el equipo no operativo.
- Escritorio virtual Ubuntu de la UPM.
- Máquinas virtuales, como, por ejemplo, las proporcionadas por VirtualBox.
- Contenedores Docker, la opción recomendada por su flexibilidad. Se incluye una sección explicando cómo usar esta opción.

Cualquiera de las tres últimas opciones sería adecuada para el desarrollo de la primera fase de la práctica. Sin embargo, para las dos últimas fases, se requiere usar varios equipos (se podrían hacer pruebas en un solo equipo usando esa única máquina tanto como de nodo de control como de nodo configurable utilizando como dirección `localhost`).

A continuación, se presentan algunas recomendaciones a la hora de configurar la plataforma de desarrollo de la práctica aplicables a cualquiera de los entornos.

NOTA: La práctica será corregida en el contexto de un contenedor Docker con la imagen oficial de Ubuntu 24.04.

Necesidad de dispositivos de almacenamiento

Para poder probar algunos módulos, el equipo debería disponer de al menos un disco que pueda reutilizarse a discreción. Una opción aplicable a cualquier plataforma es el uso de dispositivos de tipo *loop*:

```
# crea un fichero de 1GiB, que inicialmente no ocupa espacio
$ truncate -s 1GiB miDisco
$ sudo losetup -f --show miDisco # le asocia un dispositivo de tipo loop
/dev/loop17
```

A partir de ese momento, puede usar ese dispositivo (en el ejemplo, `/dev/loop17`, que se usará en el resto del documento) en cualquier mandato que espere como parámetro un disco. Nótese que este tipo de "discos" desaparecen al reiniciar el sistema.

Uso de `sudo` sin contraseña

La ejecución de los módulos requiere el uso del mandato `sudo`, por lo que para facilitar el desarrollo de la práctica, es conveniente poder usar ese mandato sin necesidad de introducir la contraseña.

Para ello, se configurará el fichero `/etc/sudoers` (mandato `sudo visudo /etc/sudoers`) para que no requiera contraseña. En Ubuntu 24.04, habría que realizar las siguientes modificaciones en el fichero:

```
# Allow members of group sudo to execute any command
#%sudo  ALL=(ALL:ALL) ALL            # comentar esta línea
%sudo   ALL=(ALL:ALL) NOPASSWD:ALL   # y añadir esta
```

Instalación y configuración del servidor de `ssh`

Para las dos últimas fases, la práctica requiere usar `ssh` entre la máquina maestra de control y los equipos que se pretende configurar. En Ubuntu 24.04, no viene instalado por defecto el servidor `ssh`, por lo que es necesario instalar el paquete `openssh-server`. Dado que los dos últimos *scripts* se basan en el uso de `ssh` y, además, no se permite la ejecución interactiva, hay que asegurarse de que `ssh` está configurado en la máquina de control tal que el acceso a todas las máquinas use un modo de operación sin contraseñas. Para ello, puede ejecutar en la máquina de control la siguiente secuencia (suponiendo que 172.17.0.2 es la dirección IP de la máquina a la que se quiere acceder con `ssh` sin contraseña):

```
# genera las claves dejando vacía la passphrase
ssh-keygen
# copiar las claves a la máquina remota usando todavía la contraseña
ssh-copy-id 172.17.0.2
```

Uso de Docker

El objetivo de esta sección es explicar cómo se puede usar Docker como infraestructura para el desarrollo de la práctica, en caso de que haya optado por esta alternativa, lo que conlleva varias ventajas:

- Permite usar la misma distribución que se ha utilizado para preparar la práctica (Ubuntu 24.04) sin necesidad de instalarla en una máquina virtual o física.
- Proporciona un entorno virtual ágil que, ante cualquier incidencia que deje corrupto el estado de la máquina, permite crear rápidamente una nueva instancia.
- Posibilita crear de forma eficiente una red de máquinas lo que permite probar las dos últimas fases de la práctica sin necesidad de tener varias máquinas físicas o virtuales.
- La práctica se corregirá en un entorno basado en contenedores.

Creación de la imagen Docker

Se proporciona un `Dockerfile` que permite crear una imagen adaptada a los requisitos de la práctica:

- Está basada en Ubuntu 24.04.
- Tiene instalado el paquete que corresponde a la funcionalidad servidor de SSH.
- Incluye el paquete que corresponde al editor `nano`, para poder disponer de un editor dentro del contenedor.
- Dispone del paquete que habilita la funcionalidad de SUDO.
- Crea un usuario con el nombre especificado a la hora de construir la imagen y con ese mismo nombre como contraseña.
- Configura el nuevo usuario para que pueda usar SUDO sin contraseña.
- Establece una disposición de manera que al arrancar un contenedor de esa imagen se active el servicio SSH.
- Especifica que el contenedor ejecutará en el contexto del nuevo usuario tal que su directorio base sea el directorio de trabajo inicial.
- Establece que al activarse la imagen en un contenedor se arrancará un proceso que ejecuta `bash`.

El primer paso es, evidentemente, instalar Docker.

Una vez instalado podemos crear la imagen, a la que denominaremos `ASI/image`, para lo que ejecutaremos este mandato en el directorio donde reside el `Dockerfile` especificando en el argumento `USUARIO` cuál será el nombre del usuario en cuyo contexto se ejecutará un contenedor creado con esta imagen:

```
docker build --build-arg USUARIO=$USER -t ASI/image .
```

De cara a la práctica, queremos que el usuario sea el mismo que el que está construyendo la imagen.

Finalizada la generación de la imagen, podemos comprobar que se ha creado:

```
$ docker image ls ASI/image
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ASI/image    latest    4743b4da2af8   15 hours ago   276MB
```

Creación de la infraestructura con Docker

Para la primera fase de la práctica, bastaría con crear un contenedor y trabajar dentro del mismo, lo que asegura que el equipo no sufre ningún daño. Para ello, el contenedor compartirá un directorio con el anfitrión, concretamente, el directorio de la práctica será visible en el directorio `/ASI` del contenedor, lo que permite cambiar el código de los *scripts* desde el anfitrión, sin necesidad de volver a crear el contenedor.

Como se ha explicado previamente, se plantea crear un dispositivo de tipo `loop`, acción que se llevaría a cabo en el equipo *anfitrión*, que se le haría llegar al contenedor mediante el argumento `device`.

Por tanto, el mandato de arranque de un contenedor denominado `equipo1` basado en la imagen construida sería:

```
docker run --rm -it --name equipo1 -v $PWD:/ASI --device /dev/loop17 ASI/image
```

Revisemos los argumentos del mandato que crea un contenedor de nombre `equipo1`:

- `--rm`: el contenedor desaparece (se elimina el "equipo") cuando termina.
- `-it`: modo de operación interactivo con un terminal asociado.
- `--name`: nombre del contenedor.
- `-v $PWD:/ASI`: deja accesible el directorio actual, que contiene los módulos, a través del directorio `/ASI` del contenedor.
- `--device`: da visibilidad de ese disco dentro del contenedor.
- `ASI/image`: imagen que se instancia.

Nótese que para el módulo `mount` es necesario arrancar el contenedor con la opción `--privileged` puesto que la operación de montaje de un dispositivo no está disponible en un contenedor normal..

Para las dos últimas fases, el equipo en el que hemos realizado la instalación de Docker ejercerá el rol de máquina maestra de control, creando previamente instancias de esa imagen que corresponderán a las máquinas que se pretenden gestionar.

Recuerde que, como se ha explicado previamente, para evitar un comportamiento interactivo de los *scripts*, es necesario copiar la clave pública del anfitrión en los contenedores (`ssh-copy-id`; recuerde que en la imagen creada la contraseña es el propio nombre del usuario).

Primera fase: desarrollo de módulos

Como ocurre en Ansible, a la hora de crear un nuevo módulo, es necesario seguir las pautas establecidas en ese entorno en lo que se refiere a cómo recibe los argumentos y cómo genera los resultados. En Mansible, se han establecido las siguientes reglas:

- Los módulos reciben sus argumentos con el mismo formato que en Ansible: parejas con el nombre de la variable y su valor separados por un igual (`argumento=valor`). En el caso de que el argumento sea una lista, los valores se especificarán separados por comas (`argumento=valor1,valor2,valor3`).
- Para facilitar el manejo de ese formato, se proporciona el *script* `prologue` que genera por cada argumento una variable cuyo nombre y valor asociado corresponden a los del argumento.
- En el caso de una lista, además, se crea una variable adicional con el mismo nombre pero con el sufijo `_split` que tendrá como valor todos los elementos de la lista separados por espacios.
- En el material de apoyo, además del fichero `prologue`, se proporciona el *script* `test_prologue` que permite apreciar cómo opera `prologue`.
- El módulo terminará con un valor 0 si no ha habido errores y la operación ha realizado algún tipo de actualización, con un valor 128 si la operación ha sido correcta pero no ha producido ninguna actualización, usándose el resto de valores para codificar los distintos tipos de error.
- Los módulos no deben escribir por la salida estándar. Para facilitar la depuración de la práctica, puede incluir mensajes por la salida de error.
- Los módulos no deben ser interactivos: no pueden quedarse a la espera de alguna entrada de teclado. Algunos mandatos que requieren entrada de datos tienen una opción para asumir un valor por defecto. En aquellos mandatos interactivos que no tengan esa opción, se puede inyectarles por la entrada estándar el valor requerido:

  ```
  # este mandato nos pide "yes" para continuar
  mandato <<< "yes"
  ```

- El modo de operación de un módulo debe seguir la pauta de Ansible: deben ser idempotentes, comprobando todas las condiciones de error previamente y ejecutando de forma atómica.
- La funcionalidad de un módulo debe estar centrada en un solo aspecto. Así, por ejemplo, si un módulo requiere la instalación de un paquete, ese módulo no realizará dicha operación sino que el administrador previamente deberá invocar en una tarea el módulo de instalación de paquetes para llevar a cabo esa labor.

Módulo `filesystem`

Tiene una funcionalidad similar al módulo del mismo nombre de Ansible: permite crear sistemas de ficheros sobre dispositivos de bloques.

NOTA: Dado que la práctica será corregida en el contexto de un contenedor Docker con la imagen oficial de Ubuntu 24.04, a la hora de averiguar qué tipo de sistema de ficheros está presente en el disco, en caso de que haya alguno, **no puede usar el mandato `lsblk`** para hacer, concretamente, esa labor, ya que no funciona correctamente en el entorno de ese contenedor. **Utilice en su lugar `blkid`**.

A continuación, se especifican sus argumentos:

- `disk`: el dispositivo de bloques, que es un argumento obligatorio.
- `state`: indica si hay que crear el sistema de ficheros (`present`) o eliminarlo (`absent`), siendo, por defecto, `present`.
- `fs_type`: tipo de sistema de ficheros, que es obligatorio si `present` y debe corresponder a un tipo soportado en el equipo, siendo ignorado si `absent`.

- `force`: crea el sistema de ficheros aunque ya exista uno de otro tipo en el dispositivo. Tiene como valores `true` o `false`, siendo, por defecto, `false` y solo consultado si `present`.
- `opts`: lista de opciones separadas por comas que se usarán en la creación del sistema de ficheros, siendo de carácter opcional y solo consultado si `present`.

En cuanto al valor devuelto:

- 0: se ha completado correctamente realizando cambios: se ha creado o eliminado un sistema de ficheros.
- 128: se ha completado correctamente pero sin realizar cambios.
- 1: error por la ejecución sin ser superusuario.
- 2: error en los argumentos.
- 3: `disk` no es un dispositivo de bloques.
- 4: `fs_type` no es un tipo de sistema de ficheros soportado en el equipo
- 5: ya existía un sistema de ficheros de otro tipo y `force` es igual a `false`.
- 6: error en mandato que crea el sistema de ficheros.
- 7: error en mandato que elimina el sistema de ficheros.

A continuación, se describen las acciones que hay que llevar a cabo (nótese que los dos primeros casos de error ya están tratados en la versión inicial del *script*):

- Hay que comprobar que `disk` es un dispositivo de bloques, devolviendo un valor 3 en caso contrario.
- Es necesario descubrir si el dispositivo tiene ya un sistema de ficheros y, en caso afirmativo, de qué tipo. **Debe usar el mandato `blkid` para realizar esta operación** (**NOTA**: si prueba ese mandato de forma interactiva fuera del *script* asegúrese de hacerlo con `sudo` ya que, en caso contrario, puede proporcionar información obsoleta).
- En el caso de la creación, hay que realizar las siguientes acciones:
  - Se debe comprobar que el tipo de sistema de ficheros está soportado en el equipo, para lo que puede usar el fichero `/proc/filesystems`, devolviendo un valor 4 en caso contrario.
  - En el caso de que ya tuviera el tipo de sistema de ficheros pedido, se devolvería un 128.
  - Si tuviera ya un sistema de ficheros de otro tipo, se retornaría el error 5, a no ser que estuviera activa la orden `force`, en cuyo caso se crearía.
  - Finalmente, se crearía el sistema de ficheros con las opciones especificadas (recuerde que habrá que usar la variable `opts_split` que contiene una lista separada por blancos), asegurando un comportamiento no interactivo y devolviendo 6 en caso de error.
- Para la eliminación, que solo consulta el campo `disk`, hay que borrar el sistema de ficheros del disco (puede usar el mandato `wipefs`) devolviendo 7 en caso de error. Se retornará un 128 si no existía previamente un sistema de ficheros en el dispositivo.

Pruebas

Antes de nada, vamos a crear un disco virtual para evitar que un disco real del equipo pueda verse afectado en caso de error (en el caso de Docker, ejecutaremos este fragmento en el *anfitrión* y le pasaremos el dispositivo resultante al contenedor).

```
$ truncate -s 1G miDisco # fichero que soporta el disco virtual
$ DISCO=$(sudo losetup -f --show miDisco)
$ echo $DISCO # veamos cómo se llama: puede salir cualquier valor
/dev/loop17
```

A continuación, se especifican algunas pruebas para este módulo que podrán ser ejecutadas en la plataforma que considere oportuno (disponibles en el fichero `pruebas_filesystem` del material de apoyo).

```
$ sudo ./filesystem disk=/dev/NO fs_type=ext4 # dispo. no existe; error 3
$ echo $?
3
$ sudo ./filesystem disk=/etc/passwd fs_type=ext4 # no dispo. bloques; error 3
$ echo $?
3
$ sudo ./filesystem disk=$DISCO fs_type=ext # mal fs_type; error 4
$ echo $?
4
$ sudo ./filesystem disk=$DISCO fs_type=ext4 # OK retorna 0
$ echo $?
0
$ sudo blkid $DISCO # para comprobar que lo ha creado
/dev/loop17: UUID="25db4818-d882-473f-bc04-3d6b0404066a" BLOCK_SIZE="4096" TYPE="ext4"
$ sudo lsblk -o FSTYPE $DISCO # no funciona porque se está probando en contenedor con la imagen de Ubuntu 24.04 oficial
FSTYPE

$ sudo ./filesystem disk=$DISCO fs_type=ext4 state=present # OK retorna 128
$ echo $?
128
$ sudo ./filesystem disk=$DISCO fs_type=ext2 # error 5: ya existe SF y diferente
$ echo $?
5
$ sudo ./filesystem disk=$DISCO fs_type=ext2 force=true # OK 0
$ echo $?
0
$ sudo blkid $DISCO # para comprobar que lo ha creado
/dev/loop17: UUID="271f6e20-e029-46fd-a25e-b2382378149b" BLOCK_SIZE="4096" TYPE="ext2"
$ sudo ./filesystem disk=$DISCO opts=-c,-D fs_type=ext4 force=true # prueba opciones; debe ejecutar lento por las opciones especificadas (rec
$ echo $?
0
$ sudo blkid $DISCO # para comprobar que lo ha creado
/dev/loop17: UUID="e6673293-3f08-4e86-ba4a-43f436bee984" BLOCK_SIZE="4096" TYPE="ext4"
$ sudo ./filesystem disk=$DISCO opts=-XXXX fs_type=ext2 force=true # fuerza error 6: mkfs con opción errónea
$ echo $?
6
$ sudo ./filesystem disk=$DISCO state=absent # OK 0
$ echo $?
0
$ sudo ./filesystem disk=$DISCO state=absent # OK 128
$ echo $?
128
```

Módulo `user`

Tiene una funcionalidad similar al módulo del mismo nombre de Ansible: Se encarga de la gestión de cuentas de usuario ofreciendo operaciones para crearlas, modificarlas y eliminarlas.

Estos son los argumentos del módulo:

- `name`: el nombre de la cuenta de usuario, que es un argumento de carácter obligatorio.
- `state`: el valor `present` indica que hay que crear o modificar una cuenta mientras que `absent` especifica que se debe eliminar. El valor por defecto es `present`.
- `uid`: UID asociado a la cuenta; de carácter opcional; si no se especifica, lo elige el sistema.
- `group`: nombre del grupo asociado a la cuenta; de carácter opcional; si no se especifica, lo elige el sistema.
- `groups`: grupos suplementarios asociados a la cuenta; de carácter opcional; si no se especifica, no se considera ninguno.
- `comment`: campo GECOS; de carácter opcional; si no se especifica, se considera vacío.
- `shell`: *shell* asociado a la cuenta; de carácter opcional; si no se especifica, lo elige el sistema.

En cuanto al valor devuelto:

- 0: se ha completado correctamente realizando cambios.
- 128: se ha completado correctamente pero sin realizar cambios.
- 1: error por la ejecución sin ser superusuario.
- 2: error en los argumentos.
- 3: grupo especificado en `group` no existe.
- 4: al menos uno de los grupos especificados en `groups` no existe.
- 5: error en mandato que crea el usuario.
- 6: error en mandato que modifica el usuario.
- 7: error en mandato que elimina el usuario.

Teniendo en cuenta que los dos primeros casos de error ya están resueltos en la versión inicial del *script*, a continuación, se describen las acciones que hay que llevar a cabo si `state` contiene el valor `present`:

- Hay que comprobar que, si se ha especificado un grupo (`group`), este existe, devolviendo un valor 3 en caso contrario. Para esta operación y otras similares, puede usarse el mandato `getent`.
- Se debe comprobar que existan todos los grupos secundarios especificados (`groups`), en caso de que los haya, devolviendo un valor 4 en caso contrario.
- Si el usuario no existía previamente, se debe crear la cuenta (`useradd`) con todas las opciones especificadas (`shell`, `comment`, `uid`, `group` y `groups`), sin contraseña (`-p ''`) y creando el directorio `home` (`-m`), retornado un valor 5 en caso de error.
- En caso de que el usuario existiera previamente, habría que modificar la cuenta (`usermod`) si cualquiera de los parámetros de la cuenta ha cambiado. Si no se especifica ningún parámetro, se devolverá un 128. Sin embargo, en caso contrario, habría que realizar un análisis parámetro a parámetro para ver si algo ha cambiado. Ese análisis sería laborioso (tenga en cuenta que, por ejemplo, en el parámetro `groups` se podrían especificar los mismos grupos secundarios pero en diferente orden) por lo que se propone la siguiente estrategia basada en hacer siempre un `usermod`:
  - Se obtienen primero todos los parámetros actuales de la cuenta (puede guardar en dos variables el resultado de los mandatos `getent` y `groups` aplicados a esa cuenta).
  - A continuación, se ejecuta `usermod` con los parámetros pedidos. Nótese que la manera de construir la línea correspondiente a este mandato es la misma que para `useradd`, pero eliminando la contraseña y la creación del `home`. Devolverá un 6 en caso de error.
  - Se vuelven a obtener los parámetros de la cuenta y, si son iguales a los que había antes, se devuelve un 128, retornado 0 en caso contrario.

Acto seguido, se describen las acciones que hay que llevar a cabo si `state` es `absent`:

- Si el usuario no existe, devuelve un 128. En caso de que exista, lo borra (`userdel`) eliminando el directorio `home` (`-r`), retornando un 7 en caso de error y 0 en caso contrario.

Pruebas

A continuación, se especifican algunas pruebas para este módulo que podrán ser ejecutadas en la plataforma que considere oportuno (disponibles en el fichero `pruebas_user` del material de apoyo).

```
$ sudo ./user name=test group=noexiste # error 3: grupo no existente
$ echo $?
3
$ sudo ./user name=test groups=adm,noexiste,root # error 4: un grupo secundario no existe
$ echo $?
4
$ sudo ./user name=12345 # error 5: nombre de cuenta errónea
$ echo $?
5
$ sudo ./user name=test # OK 0: con valores por defecto
$ echo $?
0
$ getent passwd test # comprobar que se ha creado correctamente
test:x:1002:1002::/home/test:/bin/sh
$ sudo ./user name=test # OK 128: ya existe
$ echo $?
128
$ sudo ./user name=test shell=/bin/sh # OK 128: usermod no cambia nada
$ echo $?
128
$ getent passwd test # comprobar que nada ha cambiado
test:x:1002:1002::/home/test:/bin/sh
$ sudo ./user name=test shell=/bin/bash # OK 0: usermod cambia shell
$ echo $?
0
$ getent passwd test # comprobar que ha cambiado
test:x:1002:1002::/home/test:/bin/bash
$ sudo ./user name=test state=absent # OK 0
$ echo $?
0
$ getent passwd test # comprobar que se ha eliminado correctamente
$ sudo ./user name=test state=absent # OK 128
$ echo $?
128
$ sudo ./user name=test shell=/bin/bash comment=TEST # OK 0: con algunas opciones
```

```
$ echo $?
0
$ getent passwd test # comprobar que se ha creado correctamente
test:x:1002:1002:TEST:/home/test:/bin/bash
$ sudo ./user name=test shell=/bin/sh comment=Test # OK 0: usermod cambia
$ echo $?
0
$ getent passwd test # comprobar que ha cambiado
test:x:1002:1002:Test:/home/test:/bin/sh
$ sudo ./user name=test state=absent # OK 0
$ echo $?
0
$ sudo ./user name=test uid=2000 group=adm # OK 0: probando uid y group; elija uid que no exista previamente y un grupo que sí
$ echo $?
0
$ getent passwd test # comprobar que se ha creado correctamente
test:x:2000:4::/home/test:/bin/sh
$ sudo ./user name=test state=absent # OK 0
$ echo $?
0
$ sudo ./user name=test groups=adm,root # OK 0: probando grupos secundarios
$ echo $?
0
$ getent passwd test # comprobar que se ha creado correctamente
test:x:1002:1002::/home/test:/bin/sh
$ groups test # comprobar que se ha creado correctamente
test : test root adm
$ sudo ./user name=test groups=root,adm # OK 128: nada ha cambiado
$ echo $?
128
$ getent passwd test # comprobar que nada ha cambiado
test:x:1002:1002::/home/test:/bin/sh
$ groups test # comprobar que nada ha cambiado
test : test root adm
$ sudo ./user name=test groups=root,adm,bin # OK 0
$ echo $?
0
$ getent passwd test # comprobar que nada ha cambiado
test:x:1002:1002::/home/test:/bin/sh
$ groups test # comprobar que ha cambiado
test : test root bin adm
$ sudo ./user name=test state=absent # OK 0
$ echo $?
0
$ getent passwd test # comprobar que se ha eliminado correctamente
```

Módulo `mount`

Tiene una funcionalidad similar al módulo del mismo nombre de Ansible: Se encarga de la gestión de las operaciones de montaje gestionando, además, el fichero `/etc/fstab`. Se establecen ciertas restricciones y simplificaciones. Por un lado, no se permiten montajes anidados de varios dispositivos sobre el mismo punto de montaje. Esta restricción también está presente en el módulo Ansible correspondiente (*the module will fail if multiple devices are mounted on the same mount point*). Por otro lado, como simplificación, no se van a gestionar las campos `dump` (quinto campo del fichero `/etc/fstab`) y `passno` (sexto campo del fichero `/etc/fstab`), no incluyéndolos en ese fichero, lo que significa que toman los valores por defecto. Por último, se recomienda realizar inicialmente una copia del fichero `/etc/fstab` para poder restaurarlo en caso de error, lo cual no sería necesario si se trabaja en un contenedor.

Estos son los argumentos del módulo:

- `state`: el valor `present` indica que hay que realizar el montaje mientras que `absent` especifica que se debe eliminar. El valor por defecto es `present`.
- `mount_point`: directorio donde se monta o desmonta, que es de carácter obligatorio.
- `source`: dispositivo que se monta, que es un argumento obligatorio si `present`, ignorándose en caso contrario.
- `fs_type`: tipo de sistema de ficheros, que es un argumento obligatorio si `present`, ignorándose en caso contrario.
- `opts`: opciones de montaje, que se reciben como una lista separada por comas, siendo obligatorio si `present`, e ignorándose en caso contrario.
- `fstab`: añade, si `present`, o elimina, si `absent`, el montaje del fichero en `/etc/fstab`. Sus valores pueden ser `true` o `false`, considerándose, por defecto, `false`.
- `dir_handling`: al montar, crea el directorio correspondiente al punto de montaje si no existe y, al desmontar, elimina dicho directorio si existe. Sus valores pueden ser `true` o `false`, considerándose, por defecto, `false`.

En cuanto al valor devuelto:

- 0: se ha completado correctamente realizando cambios.
- 128: se ha completado correctamente pero sin realizar cambios.
- 1: error por la ejecución sin ser superusuario.
- 2: error en los argumentos.
- 3: error ya existía un montaje previo incompatible.
- 4: error ya existía una entrada en /etc/fstab incompatible.
- 5: error al no existir el directorio.
- 6: error al crear el directorio.
- 7: error en mandato de montaje.
- 8: error en mandato de desmontaje.

Teniendo en cuenta que los dos primeros casos de error ya están resueltos en la versión inicial del *script* y que se proporcionan funciones para el manejo del fichero `/etc/fstab`, a continuación, se describen las acciones que hay que llevar a cabo si `state` contiene el valor `present`:

- Se debe determinar si ya está activo un montaje sobre ese directorio distinguiendo si es idéntico al solicitado (mismos dispositivo, punto de montaje, tipo de sistema de ficheros y opciones de montaje) o no, para lo que se puede usar el mandato `findmnt`. En caso de existir, pero no ser igual, se tratará como un error, terminando la ejecución del *script* con un valor de 3, puesto que no se permiten montajes anidados sobre el mismo punto de montaje.
- Si está activa la opción `fstab`, se debe comprobar si ya existe una entrada en el fichero `/etc/fstab` asociada al punto de montaje solicitado, distinguiendo si es idéntica a la que correspondería a la petición (mismos dispositivo, punto de montaje, tipo de sistema de ficheros y opciones de montaje) o no. Para ello, se pueden usar las funciones `match_first_four_fields`, pasándole como argumentos el fichero `/etc/fstab` y los cuatro campos que definen el montaje, y

`match_second_field`, pasándole como argumentos el fichero `/etc/fstab` y el punto de montaje, respectivamente. En caso de existir, pero no ser igual, se tratará como un error, terminando la ejecución del *script* con un valor de 4, puesto que no se permiten montajes anidados sobre el mismo punto de montaje.
- Se devolverá un 128 si ya existía un montaje activo idéntico y, en caso de que se haya especificado la opción `fstab`, además, esté presente en el fichero `/etc/fstab` ese mismo montaje.
- Si se ha llegado a este punto, o bien hay que realizar el montaje, si no existía previamente, o bien la operación de inserción sobre el fichero `/etc/fstab`, si se ha especificado esta opción y no existía esa entrada en el fichero, o bien ambas.
- Para realizar el montaje, debe comprobarse previamente si el directorio de montaje existe. En caso negativo y si la opción `dir_handling` no ha sido especificada, se terminará el *script* con el valor de error 5. En caso de que no existiera previamente, pero esa opción sí estuviera activa, se crearía el directorio dando el error 6 en caso de fallo. Realizado este proceso, se realiza la operación de montaje propiamente dicha, que devolverá un error 7 si falla, eliminando en ese caso el directorio si este se hubiera creado.
- Si es necesario actualizar el fichero `/etc/fstab`, se añadiría al final de dicho fichero la línea correspondiente con los 4 campos separados por tabuladores.

A continuación, se describen las acciones que hay que llevar a cabo si `state` contiene el valor `absent`:

- Se debe comprobar si ya existe un montaje sobre ese directorio (mandato `findmnt`).
- Si está activa la opción `fstab`, se debe comprobar si ya existe una entrada en el fichero `/etc/fstab` correspondiente a este punto de montaje especificado, para lo que puede usar la función `match_second_field`, pasándole como argumentos el fichero `/etc/fstab` y el punto de montaje.
- Se devolverá un 128 si no existía un montaje activo sobre ese directorio y, en caso de que se haya especificado la opción `fstab`, además, no está presente en el fichero `/etc/fstab` ese punto de montaje.
- Si se ha llegado a este punto, o bien hay que realizar el desmontaje, si existía previamente, o bien la operación de eliminación sobre el fichero `/etc/fstab`, si se ha especificado esta opción y estaba presente esa entrada en el fichero, o bien ambas.
- Con respecto a la operación de desmontaje, si falla, se devolverá el error 8. Además, si se ha especificado la opción `dir_handling`, habrá que borrar el directorio.
- Si es necesario actualizar el fichero `/etc/fstab`, basta con eliminar la línea que contiene el montaje indicado. Nótese que la función `match_second_field` imprime por la salida estándar ese número de línea, por lo que puede usar mandatos como `sed -i` sobre `/etc/fstab` para eliminarla.

Pruebas

A continuación, se especifican algunas pruebas para este módulo que podrán ser ejecutadas en la plataforma que considere oportuno (disponibles en el fichero `pruebas_mount` del material de apoyo).

```
# asumimos que en la variable de entorno DISCO tenemos un dispositivo en el que se ha creado previamente un sistema de ficheros ext4
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=rw,relatime # OK 0
$ echo $?
0
$ findmnt -M /mnt
TARGET
     SOURCE      FSTYPE OPTIONS
/mnt /dev/loop17 ext4   rw,relatime
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=rw,relatime # OK 128
$ echo $?
128
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=ro # error 3: existe un montaje previo incompatible
$ echo $?
3
$ sudo ./mount state=absent mount_point=/mnt # OK 0
$ echo $?
0
$ findmnt -M /mnt
$ sudo ./mount state=absent mount_point=/mnt # OK 128
$ echo $?
128
$ sudo ./mount source=$DISCO mount_point=/mnt2 fs_type=ext4 opts=rw,relatime # error 5 directorio no existe
$ echo $?
5
$ sudo ./mount source=$DISCO mount_point=/mnt2 fs_type=ext4 opts=rw,relatime dir_handling=true # OK 0: fuerza crear directorio
$ echo $?
0
$ findmnt -M /mnt2
TARGET
      SOURCE      FSTYPE OPTIONS
/mnt2 /dev/loop17 ext4   rw,relatime
$ sudo ./mount state=absent mount_point=/mnt2 dir_handling=true # OK 0: borra dir.
$ echo $?
0
$ findmnt -M /mnt2
$ ls /mnt2
ls: cannot access '/mnt2': No such file or directory
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=xxx # error 7: fallo en mount
$ echo $?
7
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=rw,relatime fstab=true # OK 0: con fstab
$ echo $?
0
$ findmnt -M /mnt
TARGET
     SOURCE      FSTYPE OPTIONS
/mnt /dev/loop17 ext4   rw,relatime
$ grep mnt /etc/fstab
/dev/loop17     /mnt    ext4    rw,relatime
$ sudo ./mount source=$DISCO state=absent mount_point=/mnt fstab=true # OK 0: elimina con fstab
$ echo $?
0
$ findmnt -M /mnt
$ grep mnt /etc/fstab
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=rw,relatime fstab=true # OK 0: con fstab de nuevo
$ sudo ./mount source=$DISCO state=absent mount_point=/mnt # OK 0: pero elimina sin fstab
$ echo $?
0
$ findmnt -M /mnt
$ grep mnt /etc/fstab
/dev/loop17     /mnt    ext4    rw,relatime
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=ro,relatime fstab=true # error 4: no está montado pero en fstab hay una entra
```

```
$ echo $?
4
$ sudo ./mount source=$DISCO state=absent mount_point=/mnt fstab=true # OK 0: solo elimina la entrada de fstab
$ grep mnt /etc/fstab
$ sudo ./mount source=$DISCO mount_point=/mnt fs_type=ext4 opts=ro,relatime fstab=true # OK 0: puede montarse de nuevo
$ echo $?
0
$ findmnt -M /mnt
TARGET
     SOURCE      FSTYPE OPTIONS
/mnt /dev/loop17 ext4   ro,relatime
$ grep mnt /etc/fstab
/dev/loop17     /mnt    ext4    ro,relatime
$ sudo ./mount source=$DISCO state=absent mount_point=/mnt fstab=true # OK 0
$ echo $?
0
$ findmnt -M /mnt
$ grep mnt /etc/fstab
```

Mandato `mansible`

Permite la ejecución de un módulo con unos argumentos en el conjunto de máquinas especificadas. Por tanto, el formato del mandato es el siguiente:

```
./mansible "host..." módulo "args..."
```

Tanto este mandato como el siguiente se ejecutarán sin `sudo`, pero sí lo usarán internamente en la ejecución de los *scripts* remotos. Nótese que se asume que en todos los equipos existe un mismo usuario con acceso a `sudo` y a `ssh` sin contraseña, de manera que la ejecución remota se realice en el contexto de este usuario.

La funcionalidad de este mandato es la siguiente:

- Debe comprobar que el número de argumentos es correcto (tres: equipos, módulo y argumentos del módulo), imprimiendo un mensaje por la salida de error en caso de que no lo sea y devolviendo el valor 1.
- Está organizado como un bucle que ejecuta una iteración por cada máquina remota especificada.
- Debe copiar con `scp` el módulo y el prólogo al directorio `/tmp` del equipo remoto.
- Y ejecutarlo con `ssh` en el contexto de un `sudo` en el equipo remoto pasándole los argumentos. Nótese que debe usarse la opción `-n` que impide que `ssh` lea de la entrada estándar.
- Este mandato no escribirá nada por la salida estándar ni por la de error excepto lo que se indica a continuación.
- La ejecución del módulo puede dar cuatro posibles resultados que se reflejarán en la salida estándar:
  - El equipo remoto `X` es inaccesible: `X | UNREACHABLE!!`
  - La ejecución del módulo ha retornado un 0: `X | CHANGED`
  - La ejecución del módulo ha retornado un 128: `X | SUCCESS`
  - La ejecución del módulo ha retornado un error Y: `X | FAILED (code=Y)`
- En cualquier caso, el mandato pasará a ejecutar la siguiente iteración que corresponde a otro equipo.

Pruebas

A continuación, se realizan pruebas de este mandato usando el módulo `user`:

```
# la tercera máquina no existe
$ ./mansible "172.17.0.2 172.17.0.3 172.17.0.10" user "name=test"
172.17.0.2 | CHANGED
172.17.0.3 | CHANGED
172.17.0.10 | UNREACHABLE!
# verificando que ha funcionado
$ ssh 172.17.0.2 getent passwd test
test:x:1002:1002::/home/test:/bin/sh
$ ssh 172.17.0.3 getent passwd test
test:x:1002:1002::/home/test:/bin/sh
# probando la idempotencia
$ ./mansible "172.17.0.2 172.17.0.3 172.17.0.10" user "name=test"
172.17.0.2 | SUCCESS
172.17.0.3 | SUCCESS
172.17.0.10 | UNREACHABLE!
# probando mandato erróneo (nombre de usuario tiene solo números)
$ ./mansible "172.17.0.2 172.17.0.3 172.17.0.10" user "name=12345"
2.17.0.10" user "name=12345"
172.17.0.2 | FAILED (code=5)
172.17.0.3 | FAILED (code=5)
172.17.0.10 | UNREACHABLE!
# eliminando usuario
$ ./mansible "172.17.0.2 172.17.0.3 172.17.0.10" user "name=test state=absent"
172.17.0.2 | CHANGED
172.17.0.3 | CHANGED
172.17.0.10 | UNREACHABLE!
# verificando que ha funcionado
$ ssh 172.17.0.2 getent passwd test
$ ssh 172.17.0.3 getent passwd test
# probando la idempotencia
$ ./mansible "172.17.0.2 172.17.0.3 172.17.0.10" user "name=test state=absent"
172.17.0.2 | SUCCESS
172.17.0.3 | SUCCESS
172.17.0.10 | UNREACHABLE!
```

Mandato `mansible-playbook`

Se trata del mandato que ejecuta `playbooks`, que es la esencia de Ansible:

```
mansible-playbook fich_playbook
```

El formato del fichero de configuración es el que se muestra a continuación (nótese que es una versión simplificada del usado en Ansible; tenga en cuenta que Ansible también permite la notación basada en el símbolo = aunque no sea la que se usa habitualmente en los `playbooks`):

- La primera línea especifica a qué equipos se le aplica el `playbook` (el inventario).
- Cada una de las líneas restantes es una tarea que especifica a su vez:
  - un nombre, que no puede incluir espacios.
  - un módulo.
  - los argumentos con los que se ejecutará el módulo (siempre habrá como mínimo uno).

En cuanto a la salida del mandato, debe seguir estrictamente el siguiente formato (puede ver el detalle en las pruebas):

- Una línea con la etiqueta `PLAY` y el nombre del fichero que se está procesando.
- Una línea con la etiqueta `TASK` y el nombre de la tarea por cada una que se vaya ejecutando.
- Por cada tarea una línea por cada equipo remoto con el resultado de la ejecución de la tarea:
  - El equipo remoto `X` es inaccesible: `unreachable: [X]`
  - La ejecución del módulo ha retornado un 0: `changed: [X]`
  - La ejecución del módulo ha retornado un 128: `ok: [X]`
  - La ejecución del módulo ha retornado un error: `fatal: [X]`
- Una línea con la etiqueta `PLAY RECAP` al final de la ejecución y, a continuación, una línea por cada equipo `X` con el siguiente formato:

  ```
  X: ok=n changed=n unreachable=n failed=n
  ```

  tal que cada contador indica cuántas tareas han producido ese resultado en ese equipo (aunque sea contraintuitivo, siguiendo la pauta de Ansible, **`ok` incluye también `changed`**).

La funcionalidad de este mandato es la siguiente:

- Debe comprobar que recibe un único argumento, imprimiendo un mensaje por la salida de error en caso de error y retornando un valor de 1.
- Debe comprobar que corresponde a un fichero normal con acceso de lectura, imprimiendo un mensaje por la salida de error en caso de que no lo sea y retornando un valor de 2.
- De la primera línea debe extraer los equipos remotos.
- Por cada una de las siguientes líneas debe:
  - Comprobar que el formato es correcto: debe incluir los campos correspondientes y el módulo debe ser un fichero ejecutable. En caso de error, se pasa a procesar la siguiente línea.
  - Con respecto al procesado de cada línea, es similar al realizado por el mandato `mansible`, adaptando la salida al formato indicado.
- Terminado todo el procesamiento hay que imprimir la recapitulación. Se sugiere el uso de `arrays` asociativos para implementar esta contabilidad.

Pruebas

A continuación, se realizan pruebas de este mandato. Este es el fichero `test.play`:

```
172.17.0.2 172.17.0.3 172.17.0.10
crea_usuario user name=test
recrea_usuario user name=test
crea_usuario_error user name=12345
elimina_usuario user name=test state=absent
reelimina_usuario user name=test state=absent
```

El resultado de la ejecución es:

```
./mansible-playbook test.play
PLAY [test.play] ***********************************************

TASK [crea_usuario] ********************************************
changed: [172.17.0.2]
changed: [172.17.0.3]
unreachable: [172.17.0.10]

TASK [recrea_usuario] ******************************************
ok: [172.17.0.2]
ok: [172.17.0.3]
unreachable: [172.17.0.10]

TASK [crea_usuario_error] **************************************
fatal: [172.17.0.2]
fatal: [172.17.0.3]
unreachable: [172.17.0.10]

TASK [elimina_usuario] *****************************************
changed: [172.17.0.2]
changed: [172.17.0.3]
unreachable: [172.17.0.10]

TASK [reelimina_usuario] ***************************************
ok: [172.17.0.2]
ok: [172.17.0.3]
unreachable: [172.17.0.10]

PLAY RECAP *****************************************************
172.17.0.2: ok=4 changed=2 unreachable=0 failed=1
172.17.0.3: ok=4 changed=2 unreachable=0 failed=1
172.17.0.10: ok=0 changed=0 unreachable=5 failed=0
```

Entrega de la práctica

El plazo se extiende hasta el final del **26 de enero de 2026** en la convocatoria ordinaria y hasta el **10 de julio de 2026** en la extraordinaria. Se realizará en la máquina triqui, usando el mandato:

```
entrega.asi proyecto.2025
```

Este mandato recogerá los siguientes ficheros:

- `autores.txt`: Fichero con los datos de los autores:

  ```
  DNI APELLIDOS NOMBRE MATRÍCULA
  ```

- `filesystem`
- `user`
- `mount`
- `mansible`
- `mansible-playbook`