08 Control de flujo

En Ansible, los playbooks pueden contener múltiples tareas similares o repetidas. Reescribirlas una por una no resulta eficiente ni mantenible. Para mejorar esta situación, Ansible permite estructurar datos (listas, diccionarios...) y aplicar mecanismos de control de flujo que nos permiten iterar, condicionar la ejecución o incluso gestionar tareas de forma dinámica.

El control de flujo es clave cuando trabajamos con configuraciones masivas, múltiples servidores o conjuntos de datos que deben ser tratados de forma homogénea.

Iteraciones con bucles

Una de las formas más comunes de evitar la repetición es mediante el uso de bucles. En versiones actuales de Ansible se utiliza la directiva *loop*, aunque todavía se admite el uso de with_items, entre otros muchos, por retrocompatibilidad. Podemos tener entonces un playbook para instalar MySQL como se realizó en la anterior tarea que estuviera implementado a través de bucles.

Antes de nada, si tenemos el servicio instalado debemos eliminarlo.

```
root@debian64:~$ apt purge -y mysql-server mysql-client mysql-common mysql-server-
core-* mysql-client-core-*
root@debian64:~$ systemctl status mysql
```

Siendo los datos un array de diccionarios en otro archivo diferente. Esto permite que siempre que queramos añadir nuevos datos únicamente modificamos este archivo.

```
# Contraseña del usuario administrador
mysql_root_password: "toor"

# Versión de paquete apt
version_package: "mysql-apt-config_0.8.34-1_all.deb"

# Paquetes MariaDB conflictivos
mariadb_conflict_packages:
  - mariadb-server
  - mariadb-client
  - mariadb-client-core
  - mariadb-server-core

# Variables de configuración de MySQL
config_initial:
  - question: "mysql-apt-config/repo-codename"
    value: "bookworm"
    vtype: "select"
    name: "mysql-apt-config"

  - question: "mysql-apt-config/repo-distro"
```

```
      value: "debian"
      vtype: "select"
      name: "mysql-apt-config"

    - question: "mysql-apt-config/repo-url"
      value: "http://repo.mysql.com/apt"
      vtype: "string"
      name: "mysql-apt-config"

    - question: "mysql-apt-config/select-connectors"
      value: "Enabled"
      vtype: "select"
      name: "mysql-apt-config"

    - question: "mysql-apt-config/select-product"
      value: "Ok"
      vtype: "select"
      name: "mysql-apt-config"

    - question: "mysql-apt-config/select-server"
      value: "mysql-8.0"
      vtype: "select"
      name: "mysql-apt-config"

    - question: "mysql-apt-config/unsupported-platform"
      value: "abort"
      vtype: "select"
      name: "mysql-apt-config"

  config_server:
    - question: "mysql-community-server/data-dir"
      value: ""
      vtype: "note"
      name: "mysql-community-server"

    - question: "mysql-community-server/root-pass"
      value: "{{ mysql_root_password }}"
      vtype: "password"
      name: "mysql-community-server"

    - question: "mysql-community-server/re-root-pass"
      value: "{{ mysql_root_password }}"
      vtype: "password"
      name: "mysql-community-server"

    - question: "mysql-community-server/remove-data-dir"
      value: "false"
      vtype: "boolean"
      name: "mysql-community-server"

    - question: "mysql-community-server/root-pass-mismatch"
      value: ""
      vtype: "error"
      name: "mysql-community-server"
```

```
  - question: "mysql-server/default-auth-override"
    value: "Use Strong Password Encryption (RECOMMENDED)"
    vtype: "select"
    name: "mysql-community-server"

  - question: "mysql-server/lowercase-table-names"
    value: ""
    vtype: "select"
    name: "mysql-community-server"
```

Un playbook que haga uso de *loop* para la instalación de mysql sería el siguiente.

```
---
# Playbook para instalar MySQL 8.0 en Debian 12 Bookworm

- name: Instalar MySQL Oracle
  hosts: srv1

  tasks:
    - name: Incluir variables de configuración MySQL
      ansible.builtin.include_vars:
        file: debconf_datos.yaml

    - name: Eliminar paquetes MariaDB conflictivos
      ansible.builtin.apt:
        name: "{{ mariadb_conflict_packages }}"
        state: absent
        purge: yes
        update_cache: yes

    - name: Configurar debconf mysql-apt-config
      ansible.builtin.debconf:
        name: "{{ item.name }}"
        question: "{{ item.question }}"
        value: "{{ item.value }}"
        vtype: "{{ item.vtype }}"
      loop: "{{ config_initial }}"

    - name: Descargar paquete inicial
      ansible.builtin.shell:
        cmd: wget -O /tmp/{{ version_package }} https://dev.mysql.com/get/{{
version_package }}

    - name: Instalar paquete inicial
      ansible.builtin.apt:
        deb: /tmp/{{ version_package }}

    - name: Configurar debconf mysql-community-server
      ansible.builtin.debconf:
        name: "{{ item.name }}"
        question: "{{ item.question }}"
```

```
        value: "{{ item.value }}"
        vtype: "{{ item.vtype }}"
      loop: "{{ config_server }}"

    - name: Instalar mysql-server
      ansible.builtin.apt:
        name: mysql-server
        update_cache: yes

    - name: Asegurar que el servicio MySQL esté corriendo
      ansible.builtin.systemd:
        name: mysql
        state: started
        enabled: yes
```

Nota: Podemos tambien añadir en la propia clausula *loop* el propio array en vez de una variable.

```
- name: Configurar debconf mysql-apt-config
  ansible.builtin.debconf:
    name: "{{ item.name }}"
    question: "{{ item.question }}"
    value: "{{ item.value }}"
    vtype: "{{ item.vtype }}"
  loop:
    - question: "mysql-apt-config/repo-codename"
      value: "bookworm"
      vtype: "select"
      name: "mysql-apt-config"

    - question: "mysql-apt-config/repo-distro"
      value: "debian"
      vtype: "select"
      name: "mysql-apt-config"
    ...
```

Variable loop_var en loop_control

La variable `loop_var` en Ansible permite personalizar el nombre de la variable que representa cada elemento durante la iteración de un bucle, en lugar de usar el nombre predeterminado `item`. Esto mejora la legibilidad, evita conflictos de variables y facilita el mantenimiento del código, especialmente en bucles anidados o múltiples bucles dentro de la misma tarea o playbook.

Claridad en el código

Usar `loop_var` permite asignar nombres descriptivos a los elementos del bucle, lo que hace que el playbook sea más comprensible. Por ejemplo, en lugar de referirse a `{{ item.name }}`, se puede usar `{{ server.name }}` si el bucle itera sobre servidores, lo que deja claro el propósito de los datos.

Evitar colisiones de variables

Cuando se usan múltiples bucles en una misma tarea o se anidan bucles, el uso de `item` puede causar confusiones o sobreescritura de valores. Definir variables personalizadas con `loop_var` evita estos conflictos. Por ejemplo, se puede tener `loop_var: current_server_config` en un bucle y `loop_var: current_initial_config` en otro, asegurando que cada uno opere con su propio contexto.

Soporte para bucles anidados

En escenarios donde un bucle está dentro de otro, `loop_var` es esencial para mantener la claridad. Permite que cada nivel del bucle use un nombre distinto, evitando ambigüedades. Aunque Ansible no permite bucles anidados directamente en una misma tarea, el uso de `loop_var` facilita la organización cuando se combinan con otras estructuras como `include_tasks` o `import_tasks`.

```
---
# Playbook para instalar MySQL 8.0 en Debian 12 Bookworm

- hosts: server1
  become: yes

  tasks:
    - name: Incluir variables de configuración MySQL
      ansible.builtin.include_vars:
        file: debconf_datos.yaml

    - name: Eliminar paquetes MariaDB conflictivos
      ansible.builtin.apt:
        name: "{{ mariadb_conflict_packages }}"
        state: absent
        purge: yes
        update_cache: yes

    - name: Configurar debconf mysql-apt-config
      ansible.builtin.debconf:
        name: "{{ initial.name }}"
        question: "{{ initial.question }}"
        value: "{{ initial.value }}"
        vtype: "{{ initial.vtype }}"
      loop: "{{ config_initial }}"
      loop_control:
        loop_var: initial

    - name: Descargar paquete inicial
      ansible.builtin.shell:
        cmd: wget -O /tmp/{{ version_package }} https://dev.mysql.com/get/{{
version_package }}

    - name: Instalar paquete inicial
      ansible.builtin.apt:
        deb: /tmp/{{ version_package }}

    - name: Configurar debconf mysql-community-server
      ansible.builtin.debconf:
```

```
        name: "{{ server.name }}"
        question: "{{ server.question }}"
        value: "{{ server.value }}"
        vtype: "{{ server.vtype }}"
      loop: "{{ config_server }}"
      loop_control:
        loop_var: server

    - name: Instalar mysql-server
      ansible.builtin.apt:
        name: mysql-server
        update_cache: yes

    - name: Asegurar que el servicio MySQL esté corriendo
      ansible.builtin.systemd:
        name: mysql
        state: started
        enabled: yes
```

Clausula dict2items

El filtro `dict2items` en Ansible se usa para transformar un diccionario en una lista de diccionarios, donde cada elemento tiene dos claves: `key` con la clave original del diccionario, y `value` con su valor correspondiente. Esto es útil porque los bucles de Ansible funcionan nativamente con listas, no con diccionarios, entonces para iterar sobre un diccionario conviene primero convertirlo con `dict2items`. Por este motivo si tenemos un diccionario como el siguiente:

```
mi_dict:
  nombre: Juan
  edad: 30
```

Aplicaremos el filtro `dict2items` de la siguiente forma:

```
{{ mi_dict | dict2items }}
```

Así generamos una lista con esta estructura:

```
- key: nombre
  value: Juan
- key: edad
  value: 30
```

Así puedes recorrerlo con un loop típico de Ansible, accediendo a `item.key` y `item.value`. A mayores, podemos personalizar el nombres de los campos `key` y `value` usando los parámetros `key_name` y `value_name`:

```
{{ mi_dict | dict2items(key_name='clave', value_name='dato') }}
```

Lo que generará:

```
- clave: nombre
  dato: Juan
- clave: edad
  dato: 30
```

Esto permite adaptar la estructura a nuestras necesidades.

Uso típico en playbooks

Por ejemplo, para crear usuarios desde un diccionario:

```
users_dict:
  juan:
    uid: 1000
    grupo: dev
  ana:
    uid: 1001
    grupo: prod
```

Será transformado a:

```
- key: juan
  value:
    uid: 1000
    grupo: dev
- key: ana
  value:
    uid: 1001
    grupo: prod
```

En un playbook sería algo como lo siguiente:

```
users_dict:
  juan:
    uid: 1000
    grupo: dev
  ana:
    uid: 1001
    grupo: prod
```

```
tasks:
  - name: Crear usuarios
    user:
      name: "{{ item.key }}"
      uid: "{{ item.value.uid }}"
      group: "{{ item.value.grupo }}"
    loop: "{{ users_dict | dict2items }}"
```

Actualización de nuestro playbook para MySQL

Anteriormente haciamos uso en nuestro playbook de una estructura de datos para el debconf que correspondía con un array de diccionarios. Esto es perfectamente correcto pero tambien se podía hacer uso de un diccionario de diccionarios similar al siguiente.

Tendríamos el siguiente fichero de variables.

```
---
# Contraseña del usuario administrador
mysql_root_password: "toor"

# Versión de paquete apt
version_package: "mysql-apt-config_0.8.34-1_all.deb"

# Paquetes MariaDB conflictivos
mariadb_conflict_packages:
  - mariadb-server
  - mariadb-client
  - mariadb-client-core
  - mariadb-server-core

# Variables de configuración de MySQL (diccionario de diccionarios)
config_initial:
  mysql-apt-config/repo-codename:
    value: "bookworm"
    vtype: "select"
    name: "mysql-apt-config"

  mysql-apt-config/repo-distro:
    value: "debian"
    vtype: "select"
    name: "mysql-apt-config"

  mysql-apt-config/repo-url:
    value: "http://repo.mysql.com/apt"
    vtype: "string"
    name: "mysql-apt-config"

  mysql-apt-config/select-connectors:
    value: "Enabled"
    vtype: "select"
```

```
      name: "mysql-apt-config"

    mysql-apt-config/select-product:
      value: "Ok"
      vtype: "select"
      name: "mysql-apt-config"

    mysql-apt-config/select-server:
      value: "mysql-8.0"
      vtype: "select"
      name: "mysql-apt-config"

    mysql-apt-config/unsupported-platform:
      value: "abort"
      vtype: "select"
      name: "mysql-apt-config"

  config_server:
    mysql-community-server/data-dir:
      value: ""
      vtype: "note"
      name: "mysql-community-server"

    mysql-community-server/root-pass:
      value: "{{ mysql_root_password }}"
      vtype: "password"
      name: "mysql-community-server"

    mysql-community-server/re-root-pass:
      value: "{{ mysql_root_password }}"
      vtype: "password"
      name: "mysql-community-server"

    mysql-community-server/remove-data-dir:
      value: "false"
      vtype: "boolean"
      name: "mysql-community-server"

    mysql-community-server/root-pass-mismatch:
      value: ""
      vtype: "error"
      name: "mysql-community-server"

    mysql-server/default-auth-override:
      value: "Use Strong Password Encryption (RECOMMENDED)"
      vtype: "select"
      name: "mysql-community-server"

    mysql-server/lowercase-table-names:
      value: ""
      vtype: "select"
      name: "mysql-community-server"
```

Y haríamos uso en nuestro playbook de *dict2items* de la siguiente forma:

```
---
# Playbook para instalar MySQL 8.0 en Debian 12 Bookworm

- hosts: server1
  become: yes

  tasks:
    - name: Incluir variables de configuración MySQL
      ansible.builtin.include_vars:
        file: debconf_datos_dict2items.yaml

    - name: Eliminar paquetes MariaDB conflictivos
      ansible.builtin.apt:
        name: "{{ mariadb_conflict_packages }}"
        state: absent
        purge: yes
        update_cache: yes

    - name: Configurar debconf mysql-apt-config
      ansible.builtin.debconf:
        name: "{{ initial.value.name }}"
        question: "{{ initial.key }}"
        value: "{{ initial.value.value }}"
        vtype: "{{ initial.value.vtype }}"
      loop: "{{ config_initial | dict2items }}"
      loop_control:
        loop_var: initial

    - name: Descargar paquete inicial
      ansible.builtin.shell:
        cmd: wget -O /tmp/{{ version_package }} https://dev.mysql.com/get/{{
version_package }}

    - name: Instalar paquete inicial
      ansible.builtin.apt:
        deb: /tmp/{{ version_package }}

    - name: Configurar debconf mysql-community-server
      ansible.builtin.debconf:
        name: "{{ server.value.name }}"
        question: "{{ server.key }}"
        value: "{{ server.value.value }}"
        vtype: "{{ server.value.vtype }}"
      loop: "{{ config_server | dict2items }}"
      loop_control:
        loop_var: server

    - name: Instalar mysql-server
      ansible.builtin.apt:
        name: mysql-server
        update_cache: yes
```

```
    - name: Asegurar que el servicio MySQL esté corriendo
      ansible.builtin.systemd:
        name: mysql
        state: started
        enabled: yes
```

Bucle until

En Ansible, **until** se usa para repetir una tarea hasta que se cumpla una condición especificada. Esto es útil para esperar a que un estado cambie, como que un servicio esté activo o que un archivo contenga un resultado esperado. Se configura junto con *retries* (número máximo de intentos) y *delay* (segundos a esperar entre intentos). En el playbook siguiente:

```
---
- name: Uso de until
  hosts: server1

  tasks:

  - name: Repite una tarea has que se cumple la condición
    ansible.builtin.shell:
      cmd: cat /tmp/tarea
    register: result
    until: result.stdout=="Fin de Proceso"
    retries: 5
    delay: 2
```

La tarea ejecuta **cat /tmp/tarea** y guarda su salida en *result*. Se repetirá hasta que la salida contenga exactamente el texto "Fin de Proceso", intentando hasta 5 veces con un retardo de 2 segundos entre intentos. Si después de 5 intentos la condición no se cumple, la tarea fallará.

Esto permite que Ansible espere en bucle a que el contenido de `/tmp/tarea` cambie a la condición deseada antes de continuar con las tareas siguientes.

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook until.yaml

PLAY [Uso de until]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [server1]

TASK [Repite una tarea has que se cumple la condición]
************************************
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (5
retries left).
```

```
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (4
retries left).
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (3
retries left).
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (2
retries left).
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (1
retries left).
[ERROR]: Task failed: Module failed: non-zero return code
Origin: /home/ansible/practicas/pr8/unti.yaml:7:5

5   tasks:
6
7   - name: Repite una tarea has que se cumple la condición
      ^ column 5

fatal: [server1]: FAILED! => {"attempts": 5, "changed": true, "cmd": "cat
/tmp/tarea", "delta": "0:00:00.005340", "end": "2025-10-02 19:14:09.688522",
"msg": "non-zero return code", "rc": 1, "start": "2025-10-02 19:14:09.683182",
"stderr": "cat: /tmp/tarea: No existe el fichero o el directorio", "stderr_lines":
["cat: /tmp/tarea: No existe el fichero o el directorio"], "stdout": "",
"stdout_lines": []}

PLAY RECAP
*********************************************************************************
*
server1                    : ok=1    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

Si antes de terminar el bucle se cumple la condición tendremos una salida como la siguiente.

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook unti.yaml

PLAY [Uso de until]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [server1]

TASK [Repite una tarea has que se cumple la condición]
************************************
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (5
retries left).
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (4
retries left).
FAILED - RETRYING: [server1]: Repite una tarea has que se cumple la condición (3
retries left).
changed: [server1]

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

```
*
server1                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Otros aspectos

Clausula when

El condicional *when* en Ansible se usa para ejecutar una tarea solo si se cumple una condición especificada. Funciona evaluando expresiones basadas en variables, hechos del sistema o cualquier otra información disponible, y permite controlar el flujo del playbook para que ciertas tareas se realicen solo cuando se den condiciones específicas, como que el sistema sea una distribución determinada o que una variable esté definida. Esto hace que las automatizaciones sean más flexibles y adaptables. Además, *when* puede combinar condiciones usando operadores lógicos como "and" y "or" y puede aplicarse a múltiples condiciones al mismo tiempo.

- Se trata de una expresión Jinja2 (motor de plantillas utilizado por Ansible).
- No se usan las dobles llaves `{{ }}` dentro de la condición *when*.
- La expresión se evalúa para todos los hosts donde se ejecuta el playbook.
- Los operadores disponibles son los habituales: `<`, `>`, `>=`, `<=`, `!=`.
- La comparación de igualdad se hace con `==`.
- Se puede usar la palabra clave `is defined` para verificar si una variable existe.
- La palabra clave `not` permite la negación en las condiciones.
- Para buscar si un valor existe dentro de un array o lista se usa el operador `in`.

```
---
- name: Prueba con WHEN. Visualiza la fecha del host server1
  hosts: aula1

  tasks:
  - name: Capturar fecha de server1
    ansible.builtin.shell:
      cmd: date
    register: fecha
    when: ansible_hostname=="server1" and ansible_distribution=="Debian"

  - name: Visualizar fecha
    ansible.builtin.debug:
      msg: "{{fecha.stdout}}"
    when: ansible_hostname=="server1" and ansible_distribution=="Debian"
```

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook when.yaml

PLAY [Prueba con WHEN. Visualiza la fecha del host server1]
*************************************
```

```
TASK [Gathering Facts]
*********************************************************************************
ok: [server1]
ok: [server2]

TASK [Capturar fecha de server1]
*****************************************************************
skipping: [server2]
changed: [server1]

TASK [Visualizar fecha]
*********************************************************************************
ok: [server1] => {
    "msg": "jue 02 oct 2025 17:22:21 CEST"
}
skipping: [server2]

PLAY RECAP
*********************************************************************************
*******
server1                    : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
server2                    : ok=1    changed=0    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

Nota: Fijese que cuando el nodo objetivo no cumple la condición aparece *skipped*.

When con variables del playbook

Tambien podemos hacer uso de variables del propio playbook.

```
---
- name: Prueba con WHEN. Uso con Variables
  hosts: server1
  vars:
      ejecutar: True

  tasks:
  - name: Instalar GIT
    ansible.builtin.apt:
     name: git
     state: present
    when: ejecutar
```

When con variables registradas

Podemos registrar el contenido de una consulta a nuestra máquina objetivo y consultar esa misma variable en tareas posteriores.

```
---
- name: Prueba con WHEN. Crear un fichero en un directorio si está vacío.
  hosts: server1
  become: yes

  tasks:
  - name: Crear el directorio
    ansible.builtin.file:
      path: /tmp/dir1
      state: directory

  - name: Ver si el directorio está vacío
    ansible.builtin.shell:
      cmd: ls /tmp/dir1
    register: resultado

  - name: Crear el fichero si no existe nada en el directorio
    ansible.builtin.file:
      path: /tmp/dir1/f1.txt
      state: touch
    when: resultado.stdout==""
```

Ignorar errores en playbooks

La clausula **ignore_errors** en Ansible es una opción que permite continuar la ejecución del playbook aunque una tarea falle. Si se especifica **ignore_errors**: yes (o true) en una tarea, esta no detendrá el playbook si produce un error, sino que lo ignorará y seguirá con las siguientes tareas. Esto es útil cuando un fallo no es crítico y se desea evitar que el playbook se detenga por completo ya que hasta este punto siempre que teniamos un error se detenía la ejecución del playbook.

```
---
- name: Prueba con WHEN. Ignorar errores
  hosts: server1

  tasks:
  - name: Acceder a un firectorio que no existe
    ansible.builtin.command:
     cmd: ls /no_existe
    register: resultado
    ignore_errors: yes

  - name: Visualizar resultado de error
    ansible.builtin.debug:
      msg: "{{resultado.stderr}}"
    when: resultado.stderr !=""
```

Podemos ver en la ejecución del playbook como no se detiene el playbook. El error se muestra por pantalla y pasa a la siguiente tarea.

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook ignorar.yaml

PLAY [Prueba con WHEN. Ignorar errores]
****************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [server1]

TASK [Acceder a un firectorio que no existe]
***********************************************
[ERROR]: Task failed: Module failed: non-zero return code
Origin: /home/ansible/practicas/pr8/ignorar.yaml:6:5

4
5   tasks:
6   - name: Acceder a un firectorio que no existe
      ^ column 5

fatal: [server1]: FAILED! => {"changed": true, "cmd": ["ls", "/no_existe"],
"delta": "0:00:00.005484", "end": "2025-10-02 17:56:56.023436", "msg": "non-zero
return code", "rc": 2, "start": "2025-10-02 17:56:56.017952", "stderr": "ls: no se
puede acceder a '/no_existe': No existe el fichero o el directorio",
"stderr_lines": ["ls: no se puede acceder a '/no_existe': No existe el fichero o
el directorio"], "stdout": "", "stdout_lines": []}
...ignoring

TASK [Visualizar resultado de error]
*****************************************************
ok: [server1] => {
    "msg": "ls: no se puede acceder a '/no_existe': No existe el fichero o el
directorio"
}

PLAY RECAP
*********************************************************************************
*
server1                    : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=1
```

Ignorar servidores detenidos

La cláusula **ignore_unreachable** en Ansible es una opción que permite ignorar errores de hosts que no son accesibles (unreachable) durante la ejecución de una tarea o playbook.

Cuando un host está marcado como "UNREACHABLE" (por ejemplo, por problemas de conexión, red o SSH), Ansible normalmente detiene la ejecución para ese host, pero con **ignore_unreachable**: true, Ansible ignora ese error y continúa intentado ejecutar tareas futuras para ese host.

```
---
- name: Control de errores
  hosts: aula1

  tasks:
  - name: Visualizar un mensaje
    ansible.builtin.debug:
      msg: Mensaje del servidor

  - name: Visualizar directorio principal
    ansible.builtin.command:
      cmd: ls -l /
    ignore_unreachable: true
```

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook ignorar_servidores.yaml

PLAY [Control de errores]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [server1]
[ERROR]: Task failed: Failed to connect to the host via ssh: ssh: connect to host
192.168.122.5 port 22: Connection timed out
fatal: [server2]: UNREACHABLE! => {"changed": false, "msg": "Task failed: Failed
to connect to the host via ssh: ssh: connect to host 192.168.122.5 port 22:
Connection timed out", "unreachable": true}

TASK [Visualizar un mensaje]
*********************************************************************
ok: [server1] => {
    "msg": "Mensaje del servidor"
}

TASK [Visualizar directorio principal]
*****************************************************
changed: [server1]

PLAY RECAP
*********************************************************************************
*
server1                    : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
server2                    : ok=0    changed=0    unreachable=1    failed=0
skipped=0    rescued=0    ignored=0
```

Determinar un caso concreto de fallo

La cláusula **failed_when** en Ansible es una condición que permite definir cuándo una tarea debe ser marcada como fallida, según criterios personalizados más allá de los errores estándar de retorno.

Por defecto, una tarea falla si el comando o módulo devuelve un código distinto de cero, pero con **failed_when** se puede especificar expresamente una condición que, si se cumple, marcará la tarea como fallida.

```
---
- name: Control de errores. Failed When
  hosts: server1

  tasks:
  - name: Crear el directorio
    ansible.builtin.file:
      path: /home/ansible/temporal
      state: directory

  - name: Localizar directorio temporal
    ansible.builtin.command:
      cmd: ls -l /home/ansible/temporal
    register: salida
    failed_when: salida.rc==0

  - name: visualizar salida
    ansible.builtin.debug:
      var: salida
```

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook failed_when.yaml

PLAY [Control de errores. Failed When]
*****************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [server1]

TASK [Crear el directorio]
*********************************************************************
changed: [server1]

TASK [Localizar directorio temporal]
*****************************************************
[ERROR]: Task failed: Action failed.
Origin: /home/ansible/practicas/pr8/failed_when.yaml:11:5

 9       state: directory
10
11   - name: Localizar directorio temporal
       ^ column 5
```

```
fatal: [server1]: FAILED! => {"changed": true, "cmd": ["ls", "-l",
"/home/ansible/temporal"], "delta": "0:00:00.004996", "end": "2025-10-02
18:32:41.108537", "failed_when_result": true, "msg": "", "rc": 0, "start": "2025-
10-02 18:32:41.103541", "stderr": "", "stderr_lines": [], "stdout": "total 0",
"stdout_lines": ["total 0"]}

PLAY RECAP
*********************************************************************************
*
server1                    : ok=2    changed=1    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

Logs en Ansible

La variable de entorno **ANSIBLE_LOG_PATH** permite definir la ruta del fichero donde Ansible guardará los logs de la ejecución en el controlador (la máquina desde donde se ejecutan los playbooks). Si se configura esta variable con una ruta válida, Ansible registrará en ese fichero toda la salida, facilitando auditoría y depuración. Esta variable tiene prioridad sobre la configuración del archivo ansible.cfg para la opción log_path. Se recomienda almacenar el archivo de log en `/var/log/ansible/ansible.log`.

Alternativamente, se puede configurar en la configuración *ansible.cfg* con la clave **log_path** bajo la sección [defaults], pero la variable entorno prevalece mientras esté exportada.

En resumen, **ANSIBLE_LOG_PATH** permite una configuración dinámica de la ruta del log de Ansible, ideal para gestionar logs relacionados con distintos ficheros de inventario o entornos específicos.

```
ansible@ControlNode:~/practicas/pr8$ cat /var/log/ansible/ansible.log
2025-10-02 18:53:30,854 p=43527 u=ansible n=ansible ERROR| [ERROR]: the playbook:
failed_when.yam could not be found

2025-10-02 18:53:38,439 p=43555 u=ansible n=ansible INFO| PLAY [Control de
errores. Failed When] ***************************************************
2025-10-02 18:53:38,444 p=43555 u=ansible n=ansible INFO| TASK [Gathering Facts]
*********************************************************************
2025-10-02 18:53:39,754 p=43555 u=ansible n=ansible WARNING| [WARNING]: Host
'server1' is using the discovered Python interpreter at '/usr/bin/python3.11', but
future installation of another Python interpreter could cause a different
interpreter to be discovered. See https://docs.ansible.com/ansible-
core/2.19/reference_appendices/interpreter_discovery.html for more information.

2025-10-02 18:53:39,754 p=43555 u=ansible n=ansible INFO| ok: [server1]
2025-10-02 18:53:39,756 p=43555 u=ansible n=ansible INFO| TASK [Crear el
directorio] *************************************************************
2025-10-02 18:53:40,171 p=43555 u=ansible n=ansible INFO| ok: [server1]
2025-10-02 18:53:40,174 p=43555 u=ansible n=ansible INFO| TASK [Localizar
directorio temporal] ****************************************************
2025-10-02 18:53:40,590 p=43555 u=ansible n=ansible ERROR| [ERROR]: Task failed:
Action failed.
Origin: /home/ansible/practicas/pr8/failed_when.yaml:11:5

 9       state: directory
```

```
10
11   - name: Localizar directorio temporal
       ^ column 5


2025-10-02 18:53:40,591 p=43555 u=ansible n=ansible INFO| fatal: [server1]:
FAILED! => {"changed": true, "cmd": ["ls", "-l", "/home/ansible/temporal"],
"delta": "0:00:00.004676", "end": "2025-10-02 18:53:40.582130",
"failed_when_result": true, "msg": "", "rc": 0, "start": "2025-10-02
18:53:40.577454", "stderr": "", "stderr_lines": [], "stdout": "total 0",
"stdout_lines": ["total 0"]}
2025-10-02 18:53:40,592 p=43555 u=ansible n=ansible INFO| PLAY RECAP
*********************************************************************
*
2025-10-02 18:53:40,592 p=43555 u=ansible n=ansible INFO| server1
: ok=2    changed=0    unreachable=0    failed=1    skipped=0    rescued=0
ignored=0
```

Manejadores o Handlers

Los *handlers* se utiliza para invocar una determinada tarea y solo se ejecutan una vez, aunque haya 2 tareas que los invoquen. Los manejadores se ejecutan despues de todas las tareas del PLAY y no pueden repetirse nombres en el playbook, ya que son globales. Para su invocación se usa el comando *notify*.

A continuación instalamos el servicio Apache2 y copiamos un fichero al servicio. Una vez realizada dicha acción reiniciamos Apache2.

```
---
- name: Ejemplo con un handler
  hosts: server1
  become: yes

  tasks:
  - name: Instalar apache2
    ansible.builtin.apt:
      name: apache2
      update_cache: yes

  - name: Copiar index1.html a /var/www/html
    ansible.builtin.copy:
      src: index1.html
      dest: /var/www/html
    notify:
      - rebotar_apache

  handlers:
  - name: rebotar_apache
    ansible.builtin.service:
      name: apache2
      state: restarted
```

Bloques de tareas

Un *bloque* en Ansible es una forma de agrupar varias tareas para aplicarles propiedades o condiciones comunes, como cuando ejecutar esas tareas o manejar errores. Esto permite organizar mejor el playbook y manejar situaciones de forma conjunta para todas las tareas dentro del bloque.

Ejemplo sencillo de bloque en un playbook donde usamos *rescue* para ejecutar una tarea cuando falla el bloque o *always* que se ejecuta siempre con independencia de la salida del bloque.

```
---
- name: Ejemplo avanzado con múltiples bloques
  hosts: server1

  tasks:

    # Primer bloque con fallo simulado
    - name: Primer bloque con error
      block:
        - name: Comando que falla
          ansible.builtin.shell: ls /tmp/no_existe.txt

        - name: Esta tarea no se ejecuta
          ansible.builtin.debug:
            msg: "No me verás porque falla antes"

      rescue:
        - name: Captura error en primer bloque
          ansible.builtin.debug:
            msg: "Error detectado, ejecutando rescue en bloque 1"

      always:
        - name: Siempre se ejecuta en bloque 1
          ansible.builtin.debug:
            msg: "Siempre ejecuto, pase lo que pase en bloque 1"

    # Segundo bloque sin errores
    - name: Segundo bloque exitoso
      block:
        - name: Mostrar mensaje 1
          ansible.builtin.debug:
            msg: "Tarea 1 del bloque 2"

        - name: Mostrar mensaje 2
          ansible.builtin.debug:
            msg: "Tarea 2 del bloque 2"

      rescue:
        - name: Esto no se ejecuta
          ansible.builtin.debug:
            msg: "No hay errores en bloque 2"

      always:
```

```
        - name: Siempre se ejecuta en bloque 2
          ansible.builtin.debug:
            msg: "Siempre ejecuto en bloque 2"

    # Tercer bloque con condición
    - name: Tercer bloque con condición
      block:
        - name: Mostrar mensaje condicional
          ansible.builtin.debug:
            msg: "Ejecutando bloque 3 porque condición es verdadera"
      when: ansible_os_family == "Debian"

      rescue:
        - name: Capturar error en bloque 3
          ansible.builtin.debug:
            msg: "Error en bloque 3"

      always:
        - name: Siempre se ejecuta en bloque 3
          ansible.builtin.debug:
            msg: "Siempre ejecuto en bloque 3"
```

```
ansible@ControlNode:~/practicas/pr8$ ansible-playbook control_errores.yaml

PLAY [Ejemplo avanzado con múltiples bloques]
*********************************************

TASK [Gathering Facts]
*********************************************************************
ok: [server1]

TASK [Comando que falla]
*********************************************************************
[ERROR]: Task failed: Module failed: non-zero return code
Origin: /home/ansible/practicas/pr8/control_errores.yaml:10:11

 8     - name: Primer bloque con error
 9       block:
10         - name: Comando que falla
             ^ column 11

fatal: [server1]: FAILED! => {"changed": true, "cmd": "ls /tmp/no_existe.txt",
"delta": "0:00:00.006277", "end": "2025-10-03 01:11:08.635956", "msg": "non-zero
return code", "rc": 2, "start": "2025-10-03 01:11:08.629679", "stderr": "ls: no se
puede acceder a '/tmp/no_existe.txt': No existe el fichero o el directorio",
"stderr_lines": ["ls: no se puede acceder a '/tmp/no_existe.txt': No existe el
fichero o el directorio"], "stdout": "", "stdout_lines": []}

TASK [Captura error en primer bloque]
*****************************************************
ok: [server1] => {
```

```
    "msg": "Error detectado, ejecutando rescue en bloque 1"
}

TASK [Siempre se ejecuta en bloque 1]
*********************************************************
ok: [server1] => {
    "msg": "Siempre ejecuto, pase lo que pase en bloque 1"
}

TASK [Mostrar mensaje 1]
*********************************************************************
ok: [server1] => {
    "msg": "Tarea 1 del bloque 2"
}

TASK [Mostrar mensaje 2]
*********************************************************************
ok: [server1] => {
    "msg": "Tarea 2 del bloque 2"
}

TASK [Siempre se ejecuta en bloque 2]
*********************************************************
ok: [server1] => {
    "msg": "Siempre ejecuto en bloque 2"
}

TASK [Mostrar mensaje condicional]
*********************************************************
ok: [server1] => {
    "msg": "Ejecutando bloque 3 porque condición es verdadera"
}

TASK [Siempre se ejecuta en bloque 3]
*********************************************************
ok: [server1] => {
    "msg": "Siempre ejecuto en bloque 3"
}

PLAY RECAP
*********************************************************************************
*
server1                    : ok=8    changed=0    unreachable=0    failed=0
skipped=0    rescued=1    ignored=0
```