10 Roles

Los **roles en Ansible** surgieron para dotar a la herramienta de la posibilidad de aportar **modularidad a los playbooks** y reutilizar partes (módulos) de estos de una forma muy simple. Por ejemplo, si se necesita instalar una plataforma compuesta por PHP, MySQL y Apache sobre un equipo remoto, se pueden identificar distintos roles que realizará el equipo:

- Servidor MySQL.
- Servidor Apache con PHP.

Cada uno de estos roles se programa de forma independiente. De este modo, si en otro momento se desea implementar una plataforma basada en *Nginx + PHP-FPM + MySQL*, se puede reutilizar el rol de MySQL desarrollado previamente y solo será necesario crear los roles de *Nginx* y *PHP-FPM*.

Además, los roles se pueden compartir en la plataforma **Ansible Galaxy**, haciéndolos disponibles para otros usuarios o utilizando los desarrollados por la comunidad.

Todos los roles de un proyecto Ansible deben estar definidos dentro de la carpeta `roles/`, que se busca en el directorio actual, en `~/.ansible` o en `/etc/ansible`.

Para **instanciar un rol** (por ejemplo, *apache*), se emplea la herramienta **ansible-galaxy**.

Como principales ventajas de usar roles externos destacamos:

- Son desarrollados por otras personas, lo que ahorra tiempo de desarrollo.
- Suelen estar mantenidos por el desarrollador o la comunidad.
- Generalmente están creados para funcionar en diferentes entornos o plataformas.

Como principales desventajas de usar roles externos podemos hablar de:

- El uso de un rol ajeno puede implicar la instalación de paquetes, servicios o dependencias que no sean necesarios en nuestro entorno siendo nuestra la responsabilidad de ver el funcionamiento del rol que queremos emplear.
- Puede ser preferible crear roles propios para optimizar los recursos a nivel de hardware y software, y mantener un mayor control sobre su comportamiento.

Ansible Galaxy ofrece una gran variedad de roles y módulos desarrollados por la comunidad, disponibles en la web oficial.
También permite crear, subir y compartir roles para que otros usuarios puedan utilizarlos y contribuir a su mejora.

Crear y compartir nuestros roles puede resultar interesante, ya que pueden ser compartidos entre proyectos o publicados en Ansible Galaxy para que otros usuarios los aprovechen y mejoren con el tiempo.

Estructura de un rol

Un **rol** posee una estructura de archivos bien definida, con los siguientes principales directorios:

- **defaults** (main.yml): almacena variables predefinidas de baja prioridad, que actúan como parámetros del rol. Estas variables pueden establecerse al invocar el rol y, en caso de no ser definidas, se usarán los valores por defecto. Se usan para tener un valor por defecto de variables que tienen que ser externas y en un momento concreto no existen, por ejemplo, podemos tener un usuario invitado con datos genericos para cuando no tenemos usuarios de una base de datos mysql.
- **vars** (main.yml): contiene variables específicas y sus valores, que en la mayoría de los casos actúan como constantes. Se definen de forma separada para facilitar su modificación y tener un mayor control. Aquí podemos definir los datos que va a tener nuestro rol, por ejemplo, si queremos instalar un servicio será aquí donde deberíamos tener las preguntas del debconf que vimos en secciones anteriores.
- **tasks** (main.yml): incluye la definición de tareas que el rol ejecuta de manera ordenada.
- **files**: almacena archivos que se utilizan en el rol y se pueden transferir a los sistemas remotos usando módulos como `copy` o `ansible.posix.synchronize`.
- **templates**: contiene plantillas Jinja2 que serán utilizadas con el rol. Es recomendable tener la misma estructura de directorios del servidor de destino en esta carpeta.
- **meta** (main.yml): alberga metadatos sobre el rol, tales como dependencias, plataformas soportadas, autor y otra información relevante.
- **handlers** (main.yml): define handlers del rol, es decir, tareas que se ejecutan tras ciertos eventos o notificaciones. Normalmente suelen emplearse para reiniciar servicios que se han instalado o modificado a la hora de emplear el rol.
- **tests**: directorio que contiene pequeños test e inventarios para comprobar el funcionamiento del rol.

Esta estructura permite organizar el contenido de los roles de manera clara y flexible, facilitando su mantenimiento y su reutilización en diferentes playbooks y proyectos.

Uso de roles en playbooks

Los roles se pueden incluir en los playbooks de Ansible de varias formas:

- **Modo estático** (import_role): el rol se comporta igual que si se definiera directamente en el play.
- **Modo dinámico** (include_role): permite ejecutar el rol en el orden y lugar especificados dentro de las tareas.

Creación de roles

La estructura básica de un rol puede generarse automáticamente usando el comando **ansible-galaxy init nombre_de_role**. Esto crea todos los directorios y archivos necesarios para comenzar con la definición del rol.

```
ansible@debian64:~/practicas/pr05/roles$ ansible-galaxy init desarrollo
- Role desarrollo was created successfully
ansible@debian64:~/practicas/pr05/roles$ tree desarrollo/
desarrollo/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
```

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

9 directories, 8 files
```

Primer ejemplo de rol

Vamos a crear un rol básico para realizar la instalación de **mariabd**.

```
ansible@debian64:~/practicas/pr05/roles$ cat play.yaml
---
- name: Ejemplos de un role
  hosts: srv1
  roles:
    - mariadb

  tasks:
  - name: Última tarea
    ansible.builtin.debug:
      msg: "Primero se ejecutan las tareas del role"
```

```
ansible@ControlNode:~/practicas/pr10$ cat mariadb/tasks/main.yml
#SPDX-License-Identifier: MIT-0
---
# tasks file for mariadb
- name: Instalar MariaDB
  ansible.builtin.apt:
    name: mariadb-server
    state: present

- name: arrancar MariaDB
  ansible.builtin.service:
    name: mariadb
    state: started
```

Pudiera parecer que en este caso un rol es lo mismo que un archivo que contenga un conjunto de tareas, sin embargo, diferencia principal es que un conjunto de tasks en un archivo que solo contiene tareas agrupadas, mientras que un rol es una estructura más completa y modular.

Un rol puede incluir además variables, handlers, templates, archivos y meta-información, todo organizado de forma reutilizable y estandarizada. Un rol facilita la organización, reutilización y mantenimiento en

comparación a simples listas de tareas sueltas.

```
ansible@debian64:~/practicas/pr05/roles$ ansible-playbook play.yaml

PLAY [Ejemplos de un role] *****************************************************

TASK [Gathering Facts] *********************************************************
ok: [srv1]

TASK [mariadb : Instalar MariaDB] **********************************************
changed: [srv1]

TASK [mariadb : arrancar MariaDB] **********************************************
ok: [srv1]

TASK [Última tarea] ************************************************************
ok: [srv1] => {
    "msg": "Primero se ejecutan las tareas del role"
}

PLAY RECAP *********************************************************************
srv1                       : ok=4    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Variables en roles

Como vimos anteriormente podemos tener variables en diferentes lugares pero existe un orden de preferencia establecido. Este orden como sabemos es el siguiente:

En Ansible, las variables pueden definirse en varios lugares y cada ubicación tiene una prioridad específica a la hora de ser aplicada durante la ejecución, lo que se conoce como el orden de precedencia. Este orden determina qué valor tendrá efecto si la misma variable está definida en diferentes sitios.

El orden de menor a mayor prioridad es el siguiente:

- **Variables definidas en defaults del rol**: suelen emplearse para valores por defecto y son fácilmente sobreescribibles.
- **Variables en group_vars**: asociadas a grupos de hosts en el inventario.
- **Variables en host_vars**: asociadas a hosts concretos en el inventario.
- **Variables declaradas en el play**: definidas directamente en el playbook.
- **Variables en vars del rol**: poseen una prioridad superior a las anteriores dentro del contexto de un rol.
- **Variables establecidas desde línea de comandos (extra_vars)**: tienen la máxima prioridad y sobrescriben todas las demás.

Veamos el siguiente ejemplo, donde se instala el servicio *nginx*. Esto es así porque el orden de preferencia establece prioridad. De las variables definidas en `rol/vars/main.yml` sobre las definidas por la clausula *vars:* del propio playbook y sobre las definidas en `rol/vars/main.yml` respectivamente. De este modo el orden de prioridad de mayor a menor a la hora de instalar el servicio en el siguiente ejemplo es **nginx/apache2/mariadb**.

Creamos un nuevo rol llamado variables con para este ejercicio en el que tenemos el siguiente contenido.

En el archivo `variables/defaults/main.yml`.

```
#SPDX-License-Identifier: MIT-0
---
# defaults file for variables
software: mariadb-server
servicio: mariadb
```

En el archivo `variables/vars/main.yml`.

```
#SPDX-License-Identifier: MIT-0
---
# vars file for variables
software: nginx
servicio: nginx
```

En el archivo `variables/tasks/main.yml`.

```
#SPDX-License-Identifier: MIT-0
---
# tasks file for variables
- name: Instalar {{software}}
  ansible.builtin.apt:
    name: "{{software}}"
    state: present
    update_cache: yes

- name: arrancar el servicio {{servicio}}
  ansible.builtin.service:
    name: "{{servicio}}"
    state: started
```

En el archivo `play.yaml`.

```
---
- name: Ejemplos de un role
  hosts: srv1
  vars:
    software: apache2
    servicio: apache2
  roles:
    - mariadb

  tasks:
```

```
  - name: Última tarea
    ansible.builtin.debug:
      msg: "Primero se ejecutan las tareas del role"
```

Si lanzamos el playbook podemos ver la ejecución del servicio **nginx**.

```
ansible@debian64:~/practicas/pr05/roles$ ansible-playbook play.yaml

PLAY [Ejemplos de un role]
*********************************************************************************
*********************************************************

TASK [Gathering Facts]
*********************************************************************************
*************************************************************
ok: [srv1]

TASK [mariadb : Instalar nginx]
*********************************************************************************
*************************************************
changed: [srv1]

TASK [mariadb : Arrancar nginx]
*********************************************************************************
*************************************************
ok: [srv1]

TASK [Última tarea]
*********************************************************************************
*************************************************************
ok: [srv1] => {
    "msg": "Primero se ejecutan las tareas del role"
}

PLAY RECAP
*********************************************************************************
*********************************************************************
srv1                       : ok=4    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Handlers o manejadores en roles

Los **handlers** en Ansible son tareas especiales que solo se ejecutan cuando otra tarea los notifica debido a que esta última ha provocado un cambio en el sistema. Es decir, un handler se activa solo si una tarea reporta que ha modificado algo, y su ejecución ocurre normalmente al final del playbook para evitar ejecuciones repetidas innecesarias. Esto es útil para acciones como reiniciar servicios solo cuando se han cambiado configuraciones, manteniendo la idempotencia del playbook y evitando acciones redundantes.

En los roles de Ansible, los handlers se definen dentro de la carpeta "handlers" en un archivo main.yml. Este archivo contiene las tareas que se ejecutarán únicamente cuando sean notificadas por tareas en la carpeta "tasks" u otras partes del playbook del rol. Es importante que el nombre de cada handler sea único para evitar conflictos, y que no se usen variables en los nombres. Los handlers ayudan a organizar y optimizar la ejecución al llamar operaciones condicionalmente, por ejemplo, reiniciar un servicio después de aplicar cambios específicos.

En el siguiente caso, vamos a instalar el servicio *mariadb* y como handler vamos a insertar la base de datos *usuarios_coches.sql* una vez concluida la instalación del servicio. Para ello tenemos los siguientes archivos.

Dentro de `files` tendremos el archivo de la base de datos.

```
CREATE DATABASE IF NOT EXISTS usuarios_coches;
USE usuarios_coches;

CREATE TABLE IF NOT EXISTS usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS coches (
  id INT AUTO_INCREMENT PRIMARY KEY,
  usuario_id INT NOT NULL,
  marca VARCHAR(50) NOT NULL,
  modelo VARCHAR(50) NOT NULL,
  año INT,
  FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE
);

INSERT INTO usuarios (nombre, email) VALUES
('Juan Pérez', 'juan@example.com'),
('Ana Gómez', 'ana@example.com');

INSERT INTO coches (usuario_id, marca, modelo, año) VALUES
(1, 'Toyota', 'Corolla', 2018),
(1, 'Honda', 'Civic', 2019),
(2, 'Ford', 'Focus', 2020);

-- Consulta para mostrar usuarios con sus coches
SELECT u.nombre, u.email, c.marca, c.modelo, c.año
FROM usuarios u
LEFT JOIN coches c ON u.id = c.usuario_id;
```

A mayores tendremos las variables para definir el servicio en `vars`.

```
#SPDX-License-Identifier: MIT-0
---
# vars file for handlers
```

```
software: mariadb-server
servicio: mariadb
```

Por otra parte, tendremos el archivo propio del handler en la carpeta `handlers`.

```
#SPDX-License-Identifier: MIT-0
---
# handlers file for mariadb2
- name: copiar_script_sql
  ansible.builtin.copy:
    src: files/usuarios_coches.sql
    dest: /tmp/usuarios_coches.sql

- name: crear_bd
  ansible.builtin.shell:
    cmd: mariadb < /tmp/usuarios_coches.sql
```

La propia tarea de la instalación del servicio que notifica a los manejadores correspondientes y se encuentra en la carpeta `tasks`.

```
#SPDX-License-Identifier: MIT-0
---
# tasks file for mariadb2
- name: Instalar {{ software }}
  ansible.builtin.apt:
    name: "{{ software }}"
    state: present
    update_cache: yes

- name: Arrancar {{ servicio }}
  ansible.builtin.service:
    name: "{{ servicio }}"
    state: restarted
  notify:
    - copiar_script_sql
    - crear_bd
```

El playbook que hace uso del rol correspondiente.

```
---
- name: Ejemplos de un role
  hosts: srv1
  roles:
    - mariadb

  tasks:
  - name: Última tarea
```

```
    ansible.builtin.debug:
      msg: "Se ha instalado el software {{software}}"
```

Ejecutamos el playbook y comprobamos la existencia de la base de datos.

```
ansible@debian64:~/practicas/pr05/roles$ ansible-playbook play.yaml

PLAY [Ejemplos de un role]
**********************************************************************************
************************************************************

TASK [Gathering Facts]
**********************************************************************************
************************************************************
ok: [srv1]

TASK [mariadb2 : Instalar mariadb-server]
**********************************************************************************
*****************************************
changed: [srv1]

TASK [mariadb2 : Arrancar mariadb]
**********************************************************************************
************************************************
changed: [srv1]

TASK [Última tarea]
**********************************************************************************
*****************************************************************
ok: [srv1] => {
    "msg": "Se ha instalado el software mariadb-server"
}

RUNNING HANDLER [mariadb2 : copiar_script_sql]
**********************************************************************************
************************************
ok: [srv1]

RUNNING HANDLER [mariadb2 : crear_bd]
**********************************************************************************
*********************************************
changed: [srv1]

PLAY RECAP
**********************************************************************************
**************************************************************************
srv1                       : ok=6    changed=3    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Ejecución de tareas en un playbook

La ejecución de un **play** en Ansible sigue un orden muy específico. Las tareas definidas dentro de la sección `roles:` de tu *playbook* se ejecutan **antes** que las tareas definidas en la sección **tasks:** principal del *play*.

1. `pre_tasks`: Tareas que quieres que se ejecuten antes de cualquier rol.
2. `roles`: Se ejecutan todas las tareas definidas en los roles listados, en el orden en que se especifican.
3. `tasks`: Tareas principales definidas directamente en el cuerpo del *playbook*.
4. `post_tasks`: Tareas que quieres que se ejecuten después de los roles y las tareas principales.
5. `handlers`: Se ejecutan todos los *handlers* que hayan sido notificados por cualquiera de las tareas anteriores (`pre_tasks`, `roles`, `tasks`, `post_tasks`).

En Ansible, **pre_tasks** y **post_tasks** son directivas que permiten ejecutar tareas específicas antes o después de las tareas principales de un playbook o rol, facilitando así la organización y control del flujo de trabajo.

Pre_tasks

- Se ejecutan **antes** de las tareas principales, incluyendo la recopilación de hechos (fact gathering) y la ejecución de roles.
- Son útiles para realizar tareas de preparación, como configurar variables, verificar condiciones previas o ejecutar comandos que deben ocurrir antes de las tareas principales.
- Por ejemplo, se pueden usar para actualizar sistema, obtener información del entorno o detener servicios antes de modificarlos.

Post_tasks

- Se ejecutan **después** de todas las tareas, incluyendo roles y tareas principales.
- Se utilizan generalmente para tareas de limpieza, notificaciones, enviar alertas o archivar información después de completar las tareas principales.
- Por ejemplo, enviar notificaciones a Slack, archivar informes o ejecutar tareas que dependen de la finalización exitosa de todas las operaciones anteriores.

```
---
- name: Ejemplos de un role
  hosts: srv1
  roles:
    # - mariadb
    - mariadb2

  pre_tasks:
    - name: Actualización de la cache
      ansible.builtin.apt:
        update_cache: yes

  tasks:
  - name: Última tarea del play
    ansible.builtin.debug:
      msg: "Se ha instalado el software {{software}}"

  post_tasks:
    - name: Información de finalización
```

```
      ansible.builtin.debug:
        msg: "Finalizamos todo el proceso de ejecución del playbook"
```

Si ejecutamos el playbook podemos ver la ejecución de las tareas.

```
ansible@debian64:~/practicas/pr05/roles$ ansible-playbook play.yaml

PLAY [Ejemplos de un role]
*********************************************************************************
*************************************************************

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

TASK [Actualización de la caché]
*********************************************************************************
************************************************
changed: [srv1]

TASK [mariadb2 : Instalar mariadb-server]
*********************************************************************************
****************************************
changed: [srv1]

TASK [mariadb2 : Arrancar mariadb]
*********************************************************************************
************************************************
changed: [srv1]

TASK [Última tarea del play]
*********************************************************************************
*************************************************************
ok: [srv1] => {
    "msg": "Se ha instalado el software mariadb-server"
}

RUNNING HANDLER [mariadb2 : copiar_script_sql]
*********************************************************************************
************************************
changed: [srv1]

RUNNING HANDLER [mariadb2 : crear_bd]
*********************************************************************************
*******************************************
changed: [srv1]

TASK [Información de finalización]
*********************************************************************************
************************************************
ok: [srv1] => {
    "msg": "Finalizamos todo el proceso de ejecución del playbook"
```

```
}

PLAY RECAP
*********************************************************************************
*********************************************************************
srv1                       : ok=8    changed=5    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Roles a nivel de tarea

Como sabemos el orden de ejecución de un rol es el siguiente definido.

- Playbook comienza.
- Por cada play:
  - Se ejecutan roles en el orden listados.
  - Dentro de cada rol, se ejecutan tareas y handlers.
  - Luego siguen las tareas definidas directamente en el play.
- Playbook continúa con próximos plays o finaliza.

Sin embargo podemos definir roles a nivel de tarea y que la ejecución se lleve a cabo según se ha definido en el playbook.

```
---
- name: Ejemplos de un role
  hosts: srv1
  tasks:
  - name: Primera tarea del play del PLAY
    ansible.builtin.debug:
      msg: "Comienzo el juego"

  - name: Incluir el role
    import_role:
      name: mariadb2

  - name: Última tarea del PLAY
    ansible.builtin.debug:
      msg: "Se ha terminado el proceso {{software}}"
```

```
ansible@debian64:~/practicas/pr05/roles$ ansible-playbook play2.yaml

PLAY [Ejemplos de un role]
*********************************************************************************
*****************************************************

TASK [Gathering Facts]
*********************************************************************************
*************************************************************
[WARNING]: Host 'srv1' 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.
ok: [srv1]

TASK [Primera tarea del play del PLAY]
*********************************************************************************
*****************************************
ok: [srv1] => {
    "msg": "Comienzo el juego"
}

TASK [mariadb2 : Instalar mariadb-server]
*********************************************************************************
*****************************************
changed: [srv1]

TASK [mariadb2 : Arrancar mariadb]
*********************************************************************************
************************************************
changed: [srv1]

TASK [Última tarea del PLAY]
*********************************************************************************
************************************************************
ok: [srv1] => {
    "msg": "Se ha terminado el proceso mariadb-server"
}

RUNNING HANDLER [mariadb2 : copiar_script_sql]
*********************************************************************************
************************************
ok: [srv1]

RUNNING HANDLER [mariadb2 : crear_bd]
*********************************************************************************
*********************************************
changed: [srv1]

PLAY RECAP
*********************************************************************************
*********************************************************************
srv1                       : ok=7    changed=3    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```