Organización	Gradiant
Autor	Manuel Landín Gómez
Fecha	25-01-2025
Versión	0.1

Introducción a Ansible

¿Qué es Ansible?

Ansible es una plataforma de automatización de código abierto. Es un *lenguaje de automatización simple* que puede describir con precisión la infraestructura de aplicaciones de TI en Ansible Playbooks. También es un *motor de automatización* que ejecuta los Ansible Playbooks.

Ansible puede gestionar tareas de automatización potentes y adaptarse a diversos flujos de trabajo y entornos. Al mismo tiempo, los nuevos usuarios de Ansible pueden comenzar a usarlo rápidamente y volverse productivos.

Ansible es simple

Los Ansible Playbooks ofrecen una automatización fácil de leer para humanos. Esto significa que los playbooks son herramientas de automatización que también son fáciles de leer, comprender y modificar. No se necesitan habilidades de programación especializadas para escribirlos. Los playbooks ejecutan tareas en orden. La simplicidad en el diseño de los playbooks los hace utilizables por cualquier equipo, permitiendo que las personas nuevas en Ansible se vuelvan productivas rápidamente.

Ansible es potente

Puedes usar Ansible para desplegar aplicaciones, gestionar configuraciones, automatizar flujos de trabajo y redes. Ansible permite orquestar todo el ciclo de vida de una aplicación.

Ansible no necesita agentes

Ansible está diseñado con una arquitectura *sin agentes*. Normalmente, Ansible se conecta a los hosts que gestiona utilizando OpenSSH o WinRM y ejecuta tareas, frecuentemente (aunque no siempre), enviando pequeños programas llamados *módulos de Ansible* a esos hosts. Estos programas se utilizan para colocar el sistema en un estado específico deseado. Cualquier módulo que se envíe se elimina una vez que Ansible ha completado sus tareas. Puedes empezar a usar Ansible casi de inmediato porque no es necesario aprobar ni implementar agentes especiales en los hosts gestionados. Al no requerir agentes ni una infraestructura de seguridad personalizada adicional, Ansible es más eficiente y seguro que otras alternativas.

Ansible tiene varias fortalezas importantes:

- *Soporte multiplataforma*: Ansible ofrece soporte sin agentes para Linux, Windows, UNIX y dispositivos de red en entornos físicos, virtuales, en la nube y en contenedores.
- *Automatización fácil de leer para humanos*: Los Ansible Playbooks, escritos como archivos de texto en YAML, son fáciles de leer y ayudan a garantizar que todos comprendan lo que hacen.
- *Descripciones precisas de aplicaciones*: Cada cambio puede realizarse mediante Ansible Playbooks, y cada aspecto del entorno de tu aplicación puede describirse y documentarse.
- *Fácil de gestionar en sistemas de control de versiones*: Los Ansible Playbooks y proyectos son texto plano. Se pueden tratar como código fuente y colocarse en tu sistema de control de versiones existente.
- *Soporte para inventarios dinámicos*: La lista de máquinas que gestiona Ansible puede actualizarse dinámicamente desde fuentes externas para capturar siempre la lista correcta y actual de todos los servidores gestionados, independientemente de la infraestructura o la ubicación.

- *Orquestación que se integra fácilmente con otros sistemas*: HP SA, Puppet, Jenkins, Red Hat Satellite y otros sistemas existentes en tu entorno pueden aprovecharse e integrarse en tu flujo de trabajo con Ansible.

La comunicación es clave en DevOps. Ansible es el primer lenguaje de automatización que puede ser leído y escrito por cualquier equipo de TI.

Conceptos y Arquitectura de Ansible

La arquitectura de Ansible consta de dos tipos de máquinas: *nodos de control* y *hosts gestionados*. Ansible se instala y ejecuta desde un nodo de control, que también contiene copias de los archivos de tu proyecto Ansible.

Los hosts gestionados están listados en un *inventario*, el cual organiza estos sistemas en grupos para facilitar su gestión colectiva. Puedes definir el inventario de forma estática en un archivo de texto o dinámicamente mediante scripts que obtienen información de grupos y hosts desde fuentes externas.

En lugar de escribir scripts complejos, los usuarios de Ansible crean *plays* de alto nivel para garantizar que un host o grupo de hosts esté en un estado particular. Un play ejecuta una serie de *tareas* en los hosts, en el orden especificado por el play. Estos plays se expresan en formato YAML en un archivo de texto. Un archivo que contiene uno o más plays se denomina *playbook*.

Cada tarea ejecuta un *módulo*, un pequeño fragmento de código (escrito en Python, PowerShell u otro lenguaje), con argumentos específicos. Cada módulo es, esencialmente, una herramienta dentro de tu caja de herramientas. Ansible incluye cientos de módulos útiles que pueden realizar una amplia variedad de tareas de automatización. Estos pueden actuar sobre archivos del sistema, instalar software o realizar llamadas a APIs.

Cuando se utiliza en una tarea, un módulo generalmente asegura que un aspecto particular de la máquina esté en un estado específico. Por ejemplo:

- Una tarea con un módulo podría asegurar que un archivo exista y tenga ciertos permisos y contenido.
- Otra tarea con un módulo diferente podría asegurar que un sistema de archivos esté montado.

Si el sistema no está en ese estado, la tarea debería colocarlo en ese estado o no hacer nada. Si una tarea falla, el comportamiento predeterminado de Ansible es abortar el resto del playbook para los hosts que tuvieron fallos y continuar con los hosts restantes.

Las tareas, los plays y los playbooks están diseñados para ser *idempotentes*. Esto significa que puedes ejecutar un playbook en los mismos hosts varias veces sin problemas. Cuando tus sistemas están en el estado correcto, el playbook no realiza cambios adicionales al ejecutarse. Aunque existen numerosos módulos disponibles para ejecutar comandos arbitrarios, debes usarlos con cuidado para asegurarte de que se ejecuten de manera idempotente.

Ansible también utiliza *plug-ins*. Los plug-ins son fragmentos de código que puedes agregar a Ansible para extenderlo y adaptarlo a nuevos usos y plataformas.

La arquitectura de Ansible es *sin agentes*. Normalmente, cuando un administrador ejecuta un Ansible Playbook, el nodo de control se conecta al host gestionado utilizando SSH (por defecto) o WinRM. Esto significa que no necesitas tener un agente específico de Ansible instalado en los hosts gestionados, ni es necesario permitir una comunicación adicional entre el nodo de control y los hosts gestionados.

Gestión de la Configuración de Ansible

Puedes crear un archivo `ansible.cfg` en el directorio de tu proyecto de Ansible para aplicar configuraciones que se usen en múltiples herramientas de Ansible.

El archivo de configuración de Ansible consta de varias secciones, donde cada sección contiene configuraciones definidas como pares clave-valor. Los títulos de las secciones están delimitados por corchetes. Para una operación básica, utiliza las siguientes dos secciones:

- `[defaults]`, que establece valores predeterminados para la operación de Ansible.
- `[privilege_escalation]`, que configura cómo Ansible realiza la escalada de privilegios en los hosts gestionados.

Por ejemplo, el siguiente es un archivo `ansible.cfg` típico:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

La siguiente tabla explica estos parámetros:

Directiva	Descripción
inventory	Especifica la ruta al archivo de inventario.
remote_user	Especifica el nombre de usuario que Ansible utiliza para conectarse a los hosts gestionados. Si no se especifica, se utiliza el nombre del usuario actual. (En un entorno de ejecución basado en contenedores gestionado por ansible-navigator, siempre será root).
ask_pass	Especifica si se debe solicitar una contraseña SSH. (Puede ser false, que es el valor predeterminado, si se utiliza autenticación con claves públicas SSH).
become	Especifica si se debe cambiar automáticamente de usuario en el host gestionado (generalmente a root) después de la conexión. Esto también puede especificarse en un play.
become_method	Especifica cómo cambiar de usuario (generalmente sudo, que es el predeterminado, aunque su es una opción).
become_user	Especifica a qué usuario cambiar en el host gestionado (generalmente root, que es el predeterminado).
become_ask_pass	Especifica si se debe solicitar una contraseña para el parámetro become_method. El valor predeterminado es false.

Construcción de un Inventario en Ansible

Definición del Inventario

Un *inventario* define una colección de hosts que Ansible gestiona. Estos hosts pueden asignarse a *grupos*, los cuales pueden gestionarse colectivamente. Los grupos pueden contener subgrupos, y los hosts pueden ser

miembros de múltiples grupos. El inventario también puede establecer variables que se aplican a los hosts y grupos que define.

Existen dos formas de definir inventarios de hosts:

- Usar un archivo de texto para definir un inventario de hosts *estático*.
- Usar un complemento (plug-in) de Ansible para generar un inventario de hosts *dinámico* según sea necesario, utilizando proveedores de información externos.

Especificación de Hosts Gestionados con un Inventario Estático

Un archivo de inventario estático es un archivo de texto que especifica los hosts gestionados que Ansible controla. Este archivo puede escribirse en varios formatos diferentes, incluyendo estilo INI o YAML. El formato estilo INI es muy común y se utiliza en la mayoría de los ejemplos de este texto.

En su forma más simple, un archivo de inventario estático estilo INI es una lista de nombres de host o direcciones IP de los hosts gestionados, cada uno en una línea separada:

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
192.0.2.42
```

Sin embargo, normalmente se organizan los hosts gestionados en *grupos de hosts*. Los grupos de hosts permiten ejecutar Ansible de manera más efectiva sobre una colección de sistemas. En este caso, cada sección comienza con el nombre del grupo de hosts entre corchetes (`[ ]`). Esto es seguido por el nombre de host o una dirección IP para cada host gestionado en el grupo, cada uno en una línea separada.

En el siguiente ejemplo, el inventario define dos grupos de hosts: `webservers` y `db-servers`.

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

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

Los hosts pueden pertenecer a múltiples grupos. De hecho, la práctica recomendada es organizar los hosts en múltiples grupos, posiblemente estructurados de diferentes maneras dependiendo del rol del host, su ubicación física, si está en producción o no, etc. Esto te permite aplicar fácilmente los plays de Ansible a conjuntos específicos de hosts según sus características, propósito o ubicación.

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

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

[east-datacenter]
web1.example.com
db1.example.com

[west-datacenter]
```

```
web2.example.com
db2.example.com

[production]
web1.example.com
web2.example.com
db1.example.com
db2.example.com

[development]
192.0.2.42
```

Definición de Grupos Anidados

Los inventarios de hosts en Ansible pueden incluir grupos que contengan otros grupos de hosts. Esto se logra creando un nombre de grupo con el sufijo `:children`. En el siguiente ejemplo, se crea un nuevo grupo llamado `galicia`, que incluye todos los hosts de los grupos `pontevedra` y `acoruna`.

```
[pontevedra]
vigo01.example.com
vigo02.example.com

[acoruna]
arteixo01.example.com
arteixo02.example.com

[galicia:children]
pontevedra
acoruna
```

Playbooks de Ansible

La potencia de Ansible radica en que puedes usar playbooks para ejecutar múltiples tareas complejas contra un conjunto de hosts seleccionados de manera fácilmente repetible.

Una *tarea* es la aplicación de un módulo para realizar una unidad de trabajo específica. Un *play* es una secuencia de tareas que se aplican, en orden, a uno o más hosts seleccionados de tu inventario. Un *playbook* es un archivo de texto que contiene una lista de uno o más plays que se ejecutan en un orden específico.

Los plays permiten convertir un conjunto largo y complejo de tareas administrativas manuales en una rutina fácilmente repetible con resultados predecibles y exitosos. En un playbook, puedes guardar la secuencia de tareas de un play en un formato legible para humanos y listo para ejecutarse de inmediato. Las propias tareas, gracias a la forma en que están escritas, documentan los pasos necesarios para implementar tu aplicación o infraestructura.

Formato de un Playbook de Ansible

El siguiente ejemplo es un playbook que contiene un play con una sola tarea:

```
---
- name: Configurar un usuario importante de forma consistente
  hosts: servera.lab.example.com
  tasks:

    - name: El usuario newbie existe con UID 4000
      ansible.builtin.user:
```

```
    name: newbie
    uid: 4000
    state: present
```

Un playbook es un archivo de texto escrito en formato YAML y normalmente se guarda con la extensión `.yml`. El playbook usa sangría con espacios para indicar la estructura de sus datos. YAML no requiere un número exacto de espacios para la sangría, pero se aplican dos reglas básicas:

- Los elementos de datos en el mismo nivel de jerarquía (como elementos de una misma lista) deben tener la misma sangría.
- Los elementos que son hijos de otro elemento deben tener más sangría que sus padres.

También puedes agregar líneas en blanco para mejorar la legibilidad.

Un playbook generalmente comienza con una línea que consta de tres guiones (`---`) para indicar el inicio del documento. Puede terminar con tres puntos (`...`) para indicar el final del documento, aunque en la práctica esto suele omitirse.

Entre estos marcadores, el playbook se define como una lista de plays. Un elemento en una lista YAML comienza con un guion seguido de un espacio. Por ejemplo, una lista YAML podría verse así:

```
- apple
- orange
- grape
```

El play en sí es una colección de pares clave-valor. Las claves en el mismo play deben tener la misma sangría. Comentemos el siguiente play:

```
---
- name: Configurar un usuario importante de forma consistente
  hosts: servera.lab.example.com
  tasks:

    - name: El usuario newbie existe con UID 4000
      ansible.builtin.user:
        name: newbie
        uid: 4000
        state: present
```

La primera línea del play comienza con un guion y un espacio (indicando que el play es el primer elemento de una lista), seguido por la primera clave, `name`. La clave `name` asocia una cadena arbitraria con el play como una etiqueta que identifica su propósito. La clave `name` es opcional, pero se recomienda porque ayuda a documentar tu playbook, especialmente cuando contiene múltiples plays.

La segunda clave en el play es `hosts`, que especifica los hosts contra los que se ejecutarán las tareas del play. La clave `hosts` toma como valor un patrón de hosts, como nombres de hosts gestionados o grupos en el inventario.

Finalmente, la última clave en el play es `tasks`, cuyo valor especifica una lista de tareas a ejecutar para este play. Este ejemplo tiene una sola tarea, que ejecuta el módulo `ansible.builtin.user` con argumentos específicos (para asegurar que el usuario `newbie` existe y tiene UID 4000).

La clave `tasks` es la parte del play que lista, en orden, las tareas que se ejecutarán en los hosts gestionados. Cada tarea en la lista es, a su vez, una colección de pares clave-valor.

En este ejemplo, la única tarea del play tiene dos claves:

- `name`: una etiqueta opcional que documenta el propósito de la tarea. Es una buena práctica nombrar todas las tareas para ayudar a documentar cada paso del proceso de automatización.
- `ansible.builtin.user`: el módulo que se ejecutará para esta tarea. Sus argumentos se pasan como una colección de pares clave-valor, que son hijos del módulo (`name`, `uid` y `state`).

Trabajando con Variables

Introducción a las Variables en Ansible

Ansible admite variables que se pueden usar para almacenar valores reutilizables en los archivos de un proyecto Ansible. Esto puede simplificar la creación y el mantenimiento del proyecto, además de reducir la cantidad de errores.

Las variables proporcionan una forma conveniente de gestionar valores dinámicos para un entorno específico en tu proyecto Ansible. Ejemplos de valores que pueden contener las variables incluyen:

- Usuarios a crear
- Paquetes a instalar
- Servicios a reiniciar
- Archivos a eliminar
- Archivos comprimidos a recuperar desde internet

Nombres de Variables

Los nombres de las variables deben comenzar con una letra y solo pueden contener letras, números y guiones bajos (_).

La siguiente tabla ilustra la diferencia entre nombres de variables no válidos y válidos.

Nombres de variables no válidos	Nombres de variables válidos
`web server`	`web_server`
`remote.file`	`remote_file`
`1st file`	`file_1 file1`
`remoteserver$1`	`remote_server_1 remote_server1`

Definición de Variables

Las variables pueden definirse en varios lugares dentro de un proyecto Ansible. Si una variable se define con el mismo nombre en dos lugares y esos valores son diferentes, la *precedencia* determina qué valor se utiliza.

Puedes definir una variable que afecte a un grupo de hosts o a hosts individuales. Algunas variables son *facts* que Ansible establece basándose en la configuración de un sistema. Otras variables pueden definirse dentro del playbook y afectar únicamente a un play o a una tarea específica en ese play. También puedes definir *variables adicionales* en la línea de comandos de `ansible-playbook` usando la opción `--extra-vars` o `-e`. Estas variables adicionales sobrescriben todos los demás valores para esa variable.

Precedencia completa de las variables en Ansible

1. **Valores predeterminados de los roles** (`defaults/main.yml` en un rol).
   - Son los valores por defecto definidos dentro de un rol.
2. **Variables definidas en el inventario estático o dinámico**, dentro de:
   - Archivos de inventario.
   - Scripts o plug-ins de inventario dinámico.
3. **Variables de grupo en el directorio `group_vars`:**
   - Definidas en un archivo dentro de `group_vars` correspondiente a un grupo del inventario.
4. **Variables de host en el directorio `host_vars`:**
   - Definidas en un archivo dentro de `host_vars` correspondiente a un host específico.
5. ***Facts* del host:**
   - Descubiertos en tiempo de ejecución mediante el módulo `setup` (habilitado por defecto en la recopilación de *facts*).
6. **Variables de rol (`vars/main.yml` dentro de un rol):**

    - Variables específicas definidas directamente dentro de un rol.
7. **Variables de inclusión de archivos (`include_vars`):**
    - Archivos de variables incluidas mediante el módulo `include_vars`.
8. **Variables de tarea (`vars` en una tarea):**
    - Definidas dentro de una tarea específica.
9. **Variables definidas en playbooks:**
    - **Bloques `vars` dentro de un play.**
    - **Archivos externos cargados mediante `vars_files`.**
10. **Variables registradas:**
    - Variables creadas durante la ejecución de tareas con el uso de `register`.
11. **Variables adicionales en la línea de comandos:**
    - Pasadas mediante `--extra-vars` o `-e` en la ejecución de `ansible-playbook` o `ansible-navigator`.

Notas importantes sobre esta precedencia

- Las variables definidas más abajo en la lista sobrescriben a las anteriores.
- Variables adicionales (`--extra-vars`) siempre tienen la precedencia más alta.
- Los *facts* del host pueden sobrescribirse si se redefinen en otros niveles más altos (por ejemplo, en un playbook o con `--extra-vars`).
- Valores predeterminados de los roles (`defaults`) son útiles para establecer configuraciones iniciales que los usuarios pueden sobrescribir fácilmente en niveles más altos.

Una práctica recomendada es elegir nombres de variables únicos a nivel global, para no tener que preocuparte por las reglas de precedencia. Sin embargo, en algunos casos puede ser útil aprovechar la precedencia para aplicar configuraciones diferentes a ciertos hosts o grupos de hosts en lugar de usar los valores predeterminados.

Si el mismo nombre de variable se define en más de un nivel, el nivel con la precedencia más alta prevalecerá. Un alcance más específico, como una variable de host o de tarea, tiene prioridad sobre un alcance más amplio, como una variable de grupo o de play. Las variables definidas en el inventario son reemplazadas por variables definidas en el playbook. Las variables adicionales definidas en la línea de comandos con las opciones `--extra-vars` o `-e` tienen la mayor precedencia.

Una discusión más detallada y precisa sobre la precedencia de variables está disponible en la documentación de Ansible en: Variable Precedence - Where should I put a variable

Variables en Playbooks

Las variables juegan un papel importante en los playbooks de Ansible porque facilitan la gestión de datos variables en un playbook.

Definición de Variables en Playbooks

Al escribir plays, puedes definir tus propias variables y luego usarlas en una tarea. Por ejemplo, puedes definir una variable llamada `web_package` con un valor de `httpd`. Una tarea puede entonces llamar a la variable utilizando el módulo `ansible.builtin.dnf` para instalar el paquete `httpd`.

Puedes definir variables en playbooks de varias maneras. Un método común es colocarlas en un bloque `vars` al inicio de un play:

```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

También es posible definir variables en archivos externos. En este caso, en lugar de usar un bloque `vars` en el playbook, puedes usar la directiva `vars_files` seguida de una lista de archivos de variables externos relativos a la ubicación del playbook:

```yaml
- hosts: all
  vars_files:
    - vars/users.yml
```

Las variables del playbook se definen luego en esos archivos en formato YAML:

```yaml
user: joe
home: /home/joe
```

Uso de Variables en Playbooks

Después de declarar variables, puedes utilizarlas en las tareas. Las variables se hacen referencia colocando el nombre de la variable entre dobles llaves (`{{ }}`). Ansible sustituye la variable por su valor al ejecutar la tarea.

```yaml
vars:
  user: joe

tasks:
  # Esta línea se leerá: Crea el usuario joe
  - name: Crea el usuario {{ user }}
    user:
      # Esta línea creará el usuario llamado Joe
      name: "{{ user }}"
```

Variables de Host y Variables de Grupo

Las variables de inventario que se aplican directamente a los hosts se dividen en dos categorías principales: las *variables de host*, que se aplican a un host específico, y las *variables de grupo*, que se aplican a todos los hosts de un grupo o de un grupo de grupos. Las variables de host tienen precedencia sobre las de grupo, pero las variables definidas en un playbook tienen precedencia sobre ambas.

Una forma de definir variables de host y de grupo es hacerlo directamente en el archivo de inventario.

Definición de la variable de host `ansible_user` para `demo.example.com`:

```
[servers]
demo.example.com ansible_user=joe
```

Definición de una variable de grupo para el grupo de hosts `servers`:

```
[servers]
demo1.example.com
demo2.example.com

[servers:vars]
user=joe
```

Definición de una variable de grupo para el grupo `servers`, que consta de dos grupos de hosts con dos servidores cada uno:

```
[servers1]
demo1.example.com
demo2.example.com

[servers2]
demo3.example.com
demo4.example.com
```

```
[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

Algunas desventajas de este enfoque son que hace que el archivo de inventario sea más difícil de manejar, mezcla información sobre hosts y variables en un mismo archivo y utiliza una sintaxis obsoleta.

Uso de Directorios para poblar Variables de Host y Grupo

Puedes definir variables para hosts y grupos de hosts creando dos directorios, `group_vars` y `host_vars`, en el mismo directorio de trabajo que el archivo de inventario o el playbook. Estos directorios contienen archivos que definen variables de grupo y de host, respectivamente.

Para definir variables de grupo para el grupo `servers`, crea un archivo YAML llamado `group_vars/servers`, cuyo contenido establecerá las variables con la misma sintaxis que en un playbook:

```
user: joe
```

De manera similar, para definir variables de host para un host específico, crea un archivo con un nombre que coincida con el host en el directorio `host_vars` y define las variables en su interior.

Ejemplos que ilustran este enfoque:

Escenario: Gestionar dos centros de datos cuyos hosts están definidos en el archivo `~/project/inventory`:

```
[datacenter1]
demo1.example.com
demo2.example.com

[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

Definir un valor general para todos los servidores en ambos centros de datos:

```
[admin@station project]$ cat ~/project/group_vars/datacenters
package: httpd
```

Definir valores diferentes para cada centro de datos:

```
[admin@station project]$ cat ~/project/group_vars/datacenter1
package: httpd
[admin@station project]$ cat ~/project/group_vars/datacenter2
package: apache
```

Definir valores diferentes para cada host gestionado:

```
[admin@station project]$ cat ~/project/host_vars/demo1.example.com
package: httpd
[admin@station project]$ cat ~/project/host_vars/demo2.example.com
package: apache
[admin@station project]$ cat ~/project/host_vars/demo3.example.com
package: mariadb-server
```

```
[admin@station project]$ cat ~/project/host_vars/demo4.example.com
package: mysql-server
```

Estructura del directorio del proyecto que contiene todos los archivos del ejemplo:

```
project
  ansible.cfg
  group_vars
    datacenters
    datacenter1
    datacenter2
  host_vars
    demo1.example.com
    demo2.example.com
    demo3.example.com
    demo4.example.com
  inventory
  playbook.yml
```

Roles en Ansible

Un **rol en Ansible** es una forma de organizar y reutilizar el código en un proyecto de automatización. Los roles dividen las tareas relacionadas en archivos y directorios bien estructurados, facilitando la gestión, el mantenimiento y la reutilización de las automatizaciones en diferentes proyectos y entornos.

Los roles son especialmente útiles para automatizaciones más complejas, ya que permiten agrupar configuraciones relacionadas, como las de un servicio, una aplicación o un componente del sistema.

Estructura de un Rol en Ansible

La estructura de un rol sigue un formato predefinido, que facilita la organización de las tareas, variables, plantillas y otros recursos necesarios. Cuando creas un rol, Ansible genera una estructura de directorios como la siguiente:

```
rol/
tasks/          Define las tareas principales que ejecutará el rol.
    main.yml    Archivo principal donde se definen las tareas del rol.
handlers/       Define handlers que pueden ser invocados desde las tareas.
    main.yml    Archivo principal con los handlers del rol.
vars/           Contiene variables predefinidas para el rol.
    main.yml    Archivo principal de variables.
defaults/       Define valores por defecto para las variables del rol.
files/          Archivos estáticos que se copian a los hosts gestionados.
templates/      Plantillas (generalmente en formato Jinja2) que se renderizan y aplican.
meta/           Metadatos sobre el rol (dependencias, etc.).
    main.yml    Archivo principal con los metadatos.
tests/          Contiene pruebas para verificar el rol.
    inventory   Archivo de inventario para pruebas.
    test.yml    Playbook de prueba.
README.md       Documentación del rol.
```

Explicación de los Componentes de un Rol

1. **`tasks/`:** Este directorio contiene el archivo `main.yml`, que define las tareas principales del rol. Es aquí donde describes qué hará el rol (instalar paquetes, configurar servicios, etc.).

2. **`handlers/`:** Los handlers son tareas especiales que se ejecutan solo cuando son notificadas, por ejemplo, para reiniciar un servicio después de cambiar su configuración.
3. **`vars/`:** Contiene las variables definidas explícitamente para el rol, las cuales tienen prioridad más alta que las variables por defecto.
4. **`defaults/`:** Almacena las variables con valores por defecto, que pueden ser sobrescritas por otras variables definidas en el inventario o playbook.
5. **`files/`:** Archivos estáticos que se copian directamente a los hosts gestionados, como binarios o configuraciones predefinidas.
6. **`templates/`:** Plantillas dinámicas que se generan en los hosts gestionados, utilizando el motor de plantillas Jinja2. Son útiles para archivos de configuración personalizados.
7. **`meta/`:** Contiene metadatos sobre el rol, como dependencias de otros roles o información adicional para catalogar el rol.
8. **`tests/`:** Proporciona un entorno de pruebas para verificar que el rol funciona como se espera.

Creación de un Rol

Puedes crear un rol en Ansible utilizando el comando `ansible-galaxy init`. Esto genera automáticamente la estructura básica del rol:

```
ansible-galaxy init nombre_del_rol
```

Esto generará una carpeta `nombre_del_rol/` con la estructura estándar descrita anteriormente.

Uso de un Rol en un Playbook

Una vez que tienes un rol, puedes incluirlo en un playbook utilizando la directiva `roles`:

```yaml
---
- name: Nombre del play
  hosts: webservers
  roles:
    - nombre_del_rol
```

Beneficios de Usar Roles

- **Reutilización:** Los roles permiten definir configuraciones y tareas reutilizables para diferentes proyectos y entornos.
- **Organización:** La estructura predefinida de los roles mejora la claridad y el mantenimiento del código.
- **Colaboración:** Los roles facilitan la colaboración entre equipos, ya que tienen una estructura estándar y pueden ser versionados.
- **Flexibilidad:** Puedes usar variables y plantillas para adaptar los roles a diferentes configuraciones.
- **Escalabilidad:** Los roles son ideales para gestionar configuraciones complejas y aplicar cambios consistentes en grandes entornos.

Colecciones de Contenido de Ansible

Cuando se desarrolló Ansible inicialmente, todos los módulos que utilizaba formaban parte del paquete de software principal. A medida que aumentó el número de módulos, se volvió más difícil para el proyecto gestionar todos estos módulos, ya que cada módulo requería un nombre único y había que sincronizar las actualizaciones de los módulos con las del código central de Ansible.

Con las *Colecciones de Contenido de Ansible*, las actualizaciones del código de Ansible se separan de las actualizaciones de los módulos y complementos. Una colección de contenido de Ansible proporciona un conjunto de módulos, roles y otros complementos relacionados que puedes usar en tus playbooks. Este enfoque permite a los proveedores y desarrolladores mantener y distribuir sus colecciones a su propio ritmo, independientemente de las versiones de Ansible.

Por ejemplo:

- La colección de contenido `redhat.insights` proporciona módulos y roles que puedes usar para registrar un sistema con Red Hat Insights en Red Hat Enterprise Linux.
- La colección `cisco.ios`, mantenida por Cisco, ofrece módulos y complementos para gestionar dispositivos de red Cisco IOS.
- La colección `community.crypto` proporciona módulos para crear certificados SSL/TLS.

Las colecciones de contenido de Ansible también ofrecen flexibilidad. Puedes instalar solo el contenido que necesitas, en lugar de instalar todos los módulos compatibles.

También puedes seleccionar una versión específica de una colección (ya sea una versión anterior o más reciente) o elegir entre una colección respaldada por Red Hat o proveedores, o una proporcionada por la comunidad.

Ansible 2.9 y versiones posteriores admiten las colecciones de contenido de Ansible. A partir de Ansible Base 2.10 y Ansible Core 2.11, la mayoría de los módulos se separaron del código principal y se colocaron en colecciones. Red Hat Ansible Automation Platform 2.2 proporciona entornos de ejecución de automatización basados en Ansible Core 2.13 que heredan esta funcionalidad.

Espacios de Nombres para las Colecciones de Contenido de Ansible

Para facilitar la especificación de colecciones y sus contenidos por nombre, los nombres de las colecciones se organizan en *espacios de nombres*. Los proveedores, socios, desarrolladores y creadores de contenido pueden usar espacios de nombres para asignar nombres únicos a sus colecciones sin conflictos con otras colecciones.

El espacio de nombres es la primera parte del nombre de una colección. Por ejemplo:

- Todas las colecciones mantenidas por la comunidad de Ansible están en el espacio de nombres `community` e incluyen nombres como `community.crypto`, `community.postgresql` y `community.rabbitmq`.
- Las colecciones mantenidas directamente por Red Hat pueden usar el espacio de nombres `redhat`, como `redhat.rhv`, `redhat.satellite` y `redhat.insights`.

Selección de Fuentes de las Colecciones de Contenido de Ansible

Independientemente de si usas `ansible-navigator` con el entorno de ejecución mínimo o `ansible-playbook` en un Ansible Core instalado en metal, siempre tienes disponible al menos una colección de contenido de Ansible: `ansible.builtin`.

Además, tu entorno de ejecución de automatización puede incluir colecciones adicionales integradas, como el entorno de ejecución predeterminado de Red Hat Ansible Automation Platform 2.2, `ee-supported-rhel8`.

Si necesitas colecciones adicionales, puedes agregarlas al subdirectorio `collections` de tu proyecto de Ansible. Las colecciones de contenido de Ansible se pueden obtener de varias fuentes:

Automation Hub
Automation Hub es un servicio proporcionado por Red Hat para distribuir colecciones de contenido certificadas por Red Hat, respaldadas por Red Hat y sus socios. Como cliente final, puedes abrir un ticket de soporte con Red Hat para estas colecciones, y Red Hat y sus socios abordarán el problema.

Se requiere una suscripción válida a Red Hat Ansible Automation Platform para acceder a Automation Hub. Usa la interfaz web en https://console.redhat.com/ansible/automation-hub/ para explorar estas colecciones.

Private Automation Hub
Tu organización puede tener su propio hub de automatización privado en el sitio, y también usar ese hub para distribuir sus propias colecciones de contenido de Ansible. El hub de automatización privado está incluido con Red Hat Ansible Automation Platform.

Ansible Galaxy
Ansible Galaxy es un sitio web respaldado por la comunidad que aloja colecciones de contenido de Ansible

enviadas por desarrolladores y usuarios. Ansible Galaxy es una biblioteca pública que no ofrece garantías formales de soporte ni está gestionada por Red Hat. Por ejemplo, las colecciones `community.crypto`, `community.postgresql` y `community.rabbitmq` están disponibles en esta plataforma.

Usa la interfaz web de Ansible Galaxy en https://galaxy.ansible.com/ para buscar colecciones.

Repositorio Git de Terceros o Archivo de Tar
También puedes descargar colecciones de contenido de Ansible desde un repositorio Git o un archivo tar local o remoto, de manera similar a como descargas roles.

Handlers en Ansible

Los módulos de Ansible están diseñados para ser *idempotentes*. Esto significa que si ejecutas un playbook varias veces, el resultado siempre será el mismo. Puedes ejecutar plays y sus tareas múltiples veces, pero los hosts gestionados solo se modificarán si esos cambios son necesarios para alcanzar el estado deseado.

Sin embargo, a veces, cuando una tarea realiza un cambio en el sistema, podría ser necesario ejecutar una tarea adicional. Por ejemplo, un cambio en el archivo de configuración de un servicio podría requerir que el servicio se recargue para que la nueva configuración surta efecto.

Los *handlers* son tareas que responden a una notificación activada por otras tareas. Las tareas solo notifican a sus handlers cuando realizan un cambio en un host gestionado. Cada handler se activa por su nombre después de que el bloque de tareas del play haya finalizado.

Si ninguna tarea notifica al handler por su nombre, este no se ejecuta. Si una o más tareas notifican al handler, este se ejecuta una vez después de que todas las tareas del play se hayan completado. Dado que los handlers son tareas, los administradores pueden usar en ellos los mismos módulos que utilizarían en cualquier otra tarea.

Normalmente, los handlers se utilizan para reiniciar servicios o reiniciar hosts.

Los handlers pueden considerarse como tareas *inactivas* que solo se ejecutan cuando son invocadas explícitamente mediante una declaración `notify`. El siguiente fragmento muestra cómo el servidor Apache solo se reinicia mediante el handler `restart apache` cuando se actualiza un archivo de configuración y se le notifica:

```
tasks:

- name: copiar la plantilla de configuración demo.example.conf
  ansible.builtin.template:
    src: /var/lib/templates/demo.example.conf.template
    dest: /etc/httpd/conf.d/demo.example.conf
  notify:
  - restart apache

handlers:

- name: restart apache
  ansible.builtin.service:
    name: httpd
```

En el ejemplo anterior, el handler `restart apache` se activa cuando es notificado por la tarea `template` de que ocurrió un cambio. Una tarea puede llamar a más de un handler en su sección `notify`. Ansible trata la declaración `notify` como un array e itera sobre los nombres de los handlers:

```
tasks:

- name: copiar la plantilla de configuración demo.example.conf
  ansible.builtin.template:
```

```
    src: /var/lib/templates/demo.example.conf.template
    dest: /etc/httpd/conf.d/demo.example.conf
  notify:
    - restart mysql
    - restart apache

handlers:

- name: restart mysql
  ansible.builtin.service:
    name: mariadb
    state: restarted

- name: restart apache
  ansible.builtin.service:
    name: httpd
    state: restarted
```

Ventajas de Usar Handlers

Como se menciona en la documentación de Ansible, hay aspectos importantes a considerar al usar handlers:

- Los handlers siempre se ejecutan en el orden especificado en la sección `handlers` del play. No se ejecutan en el orden en que son listados en las declaraciones `notify` de una tarea ni en el orden en que las tareas los notifican.
- Los handlers normalmente se ejecutan después de que todas las demás tareas del play hayan finalizado. Un handler llamado por una tarea en la sección `tasks` del playbook no se ejecuta hasta que *todas* las tareas bajo `tasks` hayan sido procesadas. (Existen algunas excepciones menores a esta regla).
- Los nombres de los handlers existen en un espacio de nombres por cada play. Si dos handlers tienen el mismo nombre por error, solo uno de ellos se ejecutará.
- Incluso si más de una tarea notifica a un handler, este se ejecuta solo una vez. Si ninguna tarea lo notifica, el handler no se ejecuta.
- Si una tarea que incluye una declaración `notify` no reporta un resultado de `changed` (por ejemplo, si un paquete ya está instalado y la tarea reporta `ok`), el handler no es notificado. Ansible solo notifica a los handlers si la tarea reporta el estado `changed`.

Ejecución Condicional de Tareas

Ansible puede usar *condiciones* para ejecutar tareas o plays cuando se cumplen ciertos criterios. Por ejemplo, puedes usar una condición para determinar la memoria disponible en un host gestionado antes de que Ansible instale o configure un servicio.

Las condiciones te ayudan a diferenciar entre hosts gestionados y asignarles roles funcionales según las condiciones que cumplan. Las variables de playbooks, las variables registradas y los *facts* de Ansible (`facts`) pueden ser evaluados con condiciones. Hay operadores disponibles para comparar cadenas de texto, datos numéricos y valores booleanos.

Los siguientes escenarios ilustran el uso de condiciones en Ansible:

- Definir un límite en una variable (por ejemplo, `min_memory`) y compararlo con la memoria disponible en un host gestionado.
- Capturar la salida de un comando y evaluarla para determinar si una tarea se completó antes de realizar una acción adicional. Por ejemplo, si un programa falla, omitir un lote.
- Usar los *facts* de Ansible para determinar la configuración de red de un host gestionado y decidir qué archivo de plantilla enviar (por ejemplo, para bonding o trunking de red).

- Evaluar el número de CPUs para determinar cómo ajustar adecuadamente un servidor web.
- Comparar una variable registrada con una variable predefinida para verificar si un servicio cambió. Por ejemplo, comprobar la suma MD5 de un archivo de configuración para determinar si el servicio cambió.

Sintaxis de Tareas Condicionales

La declaración `when` se utiliza para ejecutar una tarea de forma condicional. Su valor es la condición a probar. Si se cumple la condición, la tarea se ejecuta. Si no se cumple, la tarea se omite.

Una de las condiciones más simples que se pueden evaluar es si una variable booleana es verdadera o falsa. En el siguiente ejemplo, la declaración `when` hace que la tarea se ejecute solo si `run_my_task` es verdadero.

```
---
- name: Ejemplo de Tarea Booleana Simple
  hosts: all
  vars:
    run_my_task: true

  tasks:

    - name: El paquete httpd está instalado
      ansible.builtin.dnf:
        name: httpd
      when: run_my_task
```

El siguiente ejemplo es un poco más sofisticado y prueba si la variable `my_service` tiene un valor. Si lo tiene, se usa el valor de `my_service` como el nombre del paquete a instalar. Si la variable `my_service` no está definida, la tarea se omite sin generar un error.

```
---
- name: Demostración de Variable Definida
  hosts: all
  vars:
    my_service: httpd

  tasks:

    - name: El paquete {{ my_service }} está instalado
      ansible.builtin.dnf:
        name: "{{ my_service }}"
      when: my_service is defined
```

La siguiente tabla muestra algunas operaciones que puedes usar con condiciones:

Operación	Ejemplo
Igualdad (valor es una cadena)	`ansible_facts['machine'] == "x86_64"`
Igualdad (valor es numérico)	`max_memory == 512`
Menor que	`min_memory < 128`
Mayor que	`min_memory > 256`
Menor o igual que	`min_memory <= 256`
Mayor o igual que	`min_memory >= 512`
Diferente de	`min_memory != 512`
Variable existe	`min_memory is defined`
Variable no existe	`min_memory is not defined`
Variable booleana es verdadera (1, True, o yes se evalúan como true).	`memory_available`

Operación	Ejemplo
Variable booleana es falsa (0, False, o no se evalúan como false).	`not memory_available`
El valor de la primera variable está en la lista de la segunda variable.	`ansible_facts['distribution'] in supported_distros`

La última entrada de la tabla puede ser confusa al principio. El siguiente ejemplo ilustra cómo funciona.

En este ejemplo, la variable `ansible_facts['distribution']` es un *fact* determinado durante la tarea `Gathering Facts`, e identifica la distribución del sistema operativo del host gestionado. La variable `supported_distros` fue creada por el autor del playbook y contiene una lista de distribuciones de sistemas operativos que el playbook admite. Si el valor de `ansible_facts['distribution']` está en la lista `supported_distros`, la condición se cumple y la tarea se ejecuta.

```
---
- name: Demostración del uso de "in"
  hosts: all
  gather_facts: yes
  vars:
    supported_distros:
      - RedHat
      - Fedora

  tasks:

    - name: Instalar httpd usando dnf, donde esté soportado
      ansible.builtin.dnf:
        name: httpd
        state: present
      when: ansible_facts['distribution'] in supported_distros
```