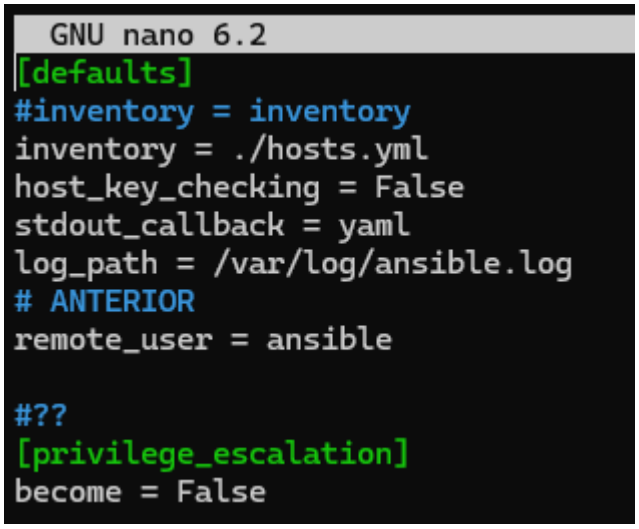
Optativa Tema 04 Ansible.

Índice

Ansible.cfg



Sección [defaults]

inventory = ./hosts.yml Indica dónde está la "agenda" de tus servidores. Al apuntar a ./hosts.yml, le dices a Ansible que use ese archivo específico

host_key_checking = False Por seguridad, SSH suele preguntar si confías en un servidor la primera vez que te conectas. Al ponerlo en False, Ansible no se detendrá a preguntar y aceptará la conexión automáticamente.

stdout_callback = yaml Cambia el formato en que ves los resultados en tu pantalla. En lugar de un bloque de texto denso (JSON), lo muestra en formato YAML

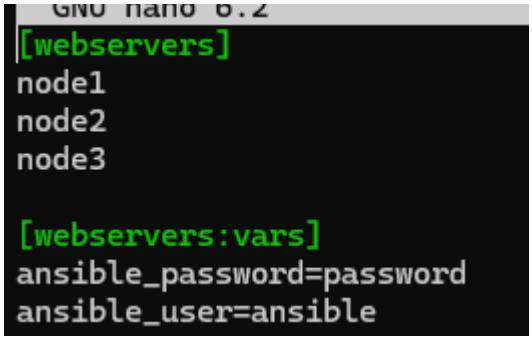
log_path = /var/log/ansible.log Guarda un registro histórico de todo lo que sucede durante las ejecuciones en ese archivo.

remote_user = ansible Define con qué nombre de usuario se conectará Ansible a los servidores remotos por defecto. En este caso, intentará entrar como el usuario llamado "ansible".

Sección [privilege_escalation]

become = False El parámetro become es el equivalente a usar sudo. Al estar en False, le estás diciendo a Ansible que, por defecto, **no intente subir de privilegios**. Ejecutará las tareas con los permisos limitados del remote_user (ansible).

Inventory



[webservers]: Definición del grupo y miembros que lo componen.

[webservers:vars]: Definición de variables para ese grupo.

Hosts.yml:

En Ansible, los grupos pueden tener "hijos" (**children**), lo que te permite organizar tus servidores por ubicación, función y entorno al mismo tiempo.

- **all**: Es el grupo raíz que contiene absolutamente todo.
- **madrid**: Es un grupo geográfico. Todo lo que esté debajo de él (db_servers, web_servers, etc.) pertenece a la ubicación "Madrid".

a) **Grupos Funcionales**:
   - db_servers: Contiene a db_master.
   - web_servers: Contiene a node2 y node3.

b) **Grupos de Entorno**:
   - dev: Solo contiene al node2.
   - prod: Contiene al db_master y al node3.

El uso de Alias (ansible_host): Aquí estás usando un **alias**. En tus playbooks te referirás a este servidor como db_master, pero Ansible sabe que para conectarse debe usar la dirección de red node1. Esto es excelente para que el nombre del servidor describa su función.

¿Cómo "ve" Ansible a tus nodos?
Gracias a esta estructura, un solo servidor puede pertenecer a múltiples grupos. Por ejemplo, si analizamos el node3:

- Es parte de web_servers.
- Es parte de prod.
- Es parte de madrid.
- Es parte de all.

Ejemplos de uso con tu archivo:

- Para lanzar un comando a todos los servidores de Madrid: **ansible madrid -m ping**

```
root@control:/ansible/proyecto2# ansible madrid -m ping
node2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
node3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
db_master | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

- Para lanzar algo solo a la base de datos: **ansible db_servers -m ping**

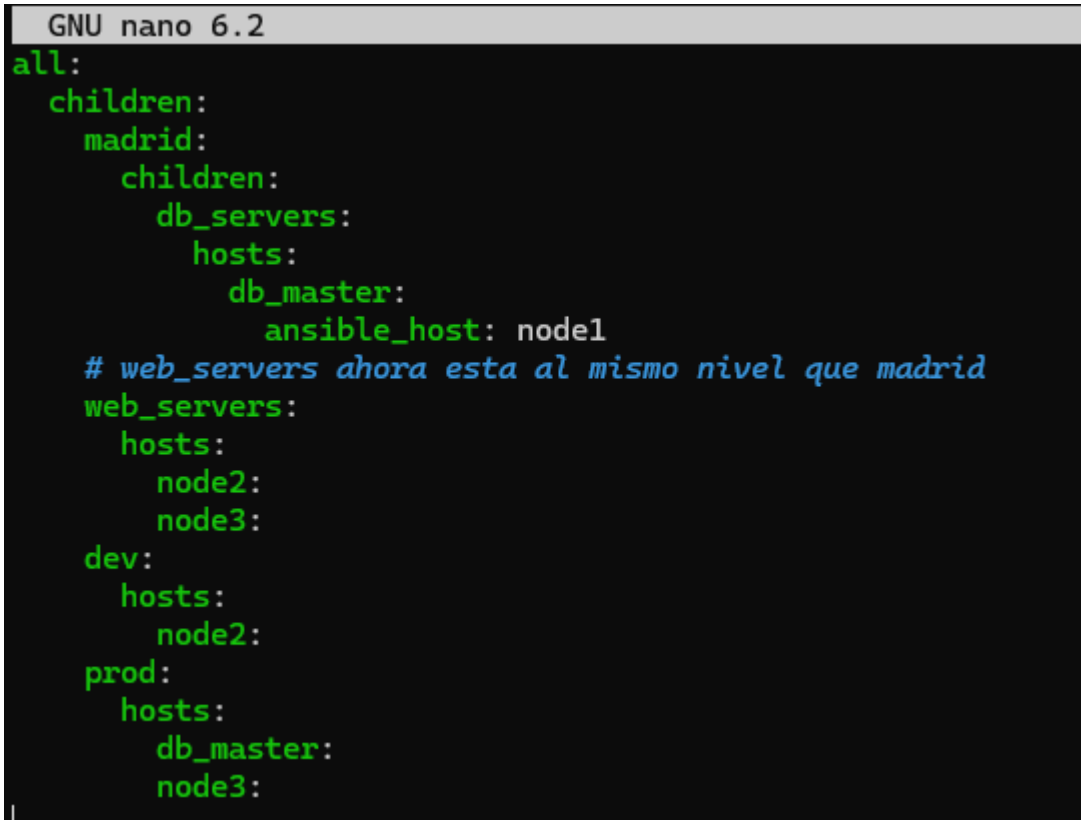
```
root@control:/ansible/proyecto2# ansible db_servers -m ping
db_master | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
root@control:/ansible/proyecto2#
```

Ping a todo el mundo: **ansible all -m ping**

```
root@control:/ansible/proyecto2# ansible all -m ping
node2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
node3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
db_master | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
root@control:/ansible/proyecto2#
```

Identación en Inventario:

Para que los web_servers dejen de ser hijos de madrid, debes ajustar la **indentación** (los espacios a la izquierda) en tu archivo YAML. En Ansible, el nivel de sangrado determina quién pertenece a quién.



```
all:
  children:
    madrid:
      children:
        db_servers:
          hosts:
            db_master:
              ansible_host: node1
    # web_servers ahora está al mismo nivel que madrid
    web_servers:
      hosts:
        node2:
        node3:
    dev:
      hosts:
        node2:
    prod:
      hosts:
        db_master:
        node3:
```

group_vars:

Ansible busca automáticamente una carpeta llamada group_vars en el mismo lugar donde tienes tu comando o tu inventario. Dentro de esa carpeta, creas archivos que se llamen **exactamente igual** que tus grupos del inventario.

```
.
├── ansible.cfg
├── hosts.yml
├── group_vars/
│   ├── all.yml           # Variables para ABSOLUTAMENTE TODOS los nodos
│   ├── madrid.yml        # Variables solo para los nodos en el grupo madrid
│   └── web_servers.yml   # Variables solo para node2 y node3
└── playbook.yml
```

¿Qué se pone dentro de estos archivos?
Dentro de cada archivo escribes las variables en formato YAML puro, sin necesidad de la estructura de children o hosts.

Ejemplo de group_vars/web_servers.yml:

```
  GNU nano 6.2
http_port: 80
service_name: apache2
```

Ejemplo de group_vars/all.yml:

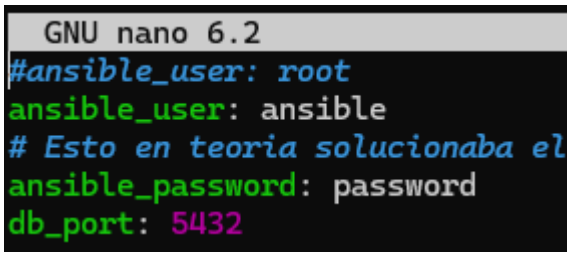
```
  GNU nano 6.2
ansible_user: ansible
ansible_password: password
empresa: TechCorp

# Ejercicio 3
log_path: /tmp/techcorp_audit.txt
audit_team:
  - auditor1
  - auditor2
  - auditor3
# Password de sudo para nodos
ansible_become_password: password
```

host_vars

Si group_vars sirve para configurar grupos enteros, **host_vars** sirve para la configuración "quirúrgica" de un **servidor específico**.

```
.
├── hosts.yml
├── group_vars/
│   └── web_servers.yml   # Configuración para node2 y node3
└── host_vars/
    ├── node1.yml         # Configuración ÚNICA para node1
    ├── node2.yml         # Configuración ÚNICA para node2
    └── node3.yml         # Configuración ÚNICA para node3
```

Ejemplo host_vars/db_master.yml:



La Jerarquía de Prioridad (Precedencia)

Ansible sigue un orden de importancia de menos a más específico. Si una variable llamada puerto aparece en varios sitios, Ansible elige la que esté más abajo en esta lista:

1. **all** en group_vars (lo más genérico).
2. **Grupos hijos** en group_vars (como madrid o web_servers).
3. **host_vars** (lo más específico, siempre gana a los grupos).

Patterns

En Ansible, los **Patterns** (patrones) son la forma en que decides sobre qué hosts o grupos quieres ejecutar un comando o un Playbook. Es, básicamente, el sistema de **filtrado** de tu inventario.

Cuando ejecutas ansible <pattern> -m ping, el "pattern" le dice a Ansible: "De todos los servidores que conozco, actúa solo en estos".

ansible web_servers -m debug -a "msg='El servicio {{ service_name }} corre en el puerto {{ http_port }}'"

- **ansible web_servers:** Le dice a Ansible que ejecute la tarea solo en los nodos que pertenecen al grupo web_servers.
- **-m debug**: Llama al **módulo debug**. Este módulo no cambia nada en el servidor remoto; su única función es imprimir información en tu pantalla.
- **{{ service_name }} y {{ http_port }}**: Son **Jinja2 templates**. Ansible buscará los valores de estas variables en tus archivos group_vars o host_vars.

```
root@control:/ansible/proyecto2# ansible web_servers -m debug -a
node2 | SUCCESS => {
    "msg": "El servicio apache2 corre en el puerto 80"
}
node3 | SUCCESS => {
    "msg": "El servicio apache2 corre en el puerto 80"
}
```

ansible "web_servers:&prod" -m ping

Para servidores del grupo web_servers que también estén en grupo prod.

La sintaxis es grupo1:&grupo2.

```
root@control:/ansible/proyecto2# ansible "web_servers:&prod" -m ping
node3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

ansible 'all:!db_servers' -m setup -a "filter=ansible_local"

1. El Patrón: 'all:!db_servers'

Como vimos antes, este es un patrón de **exclusión**:

- **all**: Selecciona todos los hosts de tu inventario.
- **:!db_servers**: El signo ! significa "NO".
- **Resultado**: Ansible ejecutará el comando en todos tus servidores (como node2, node3), **excepto** en los que pertenecen al grupo de bases de datos (db_master).

2. El Módulo: -m setup

El módulo setup es el encargado de "Gathering Facts" (recolectar hechos). Por defecto, cuando lo ejecutas, Ansible descarga muchísima información del servidor remoto: versión del kernel, memoria RAM, interfaces de red, discos, etc.

3. El Argumento: -a "filter=ansible_local"

Aquí es donde se vuelve específico. Sin este filtro, recibirías cientos de líneas de texto.

- **filter**: Le dice a Ansible que solo te muestre una parte específica de toda la información recolectada.
- **ansible_local**: Este filtro busca **Facts personalizados**.

"Ansible, ve a todos los servidores que NO sean bases de datos, ejecuta el módulo de recolección de datos, pero muéstrame únicamente los Facts personalizados que encuentres en sus carpetas locales."

Verificación de privilegios

ansible all -m shell -a "whoami"

Este comando es la "prueba de fuego" para verificar con qué identidad real está operando Ansible dentro de los servidores remotos.

Los componentes del comando

- **ansible all**: El patrón que indica que el comando se debe ejecutar en **absolutamente todos** los hosts definidos en tu inventario.
- **-m shell**: Llama al módulo shell. A diferencia del módulo command, este permite usar variables de entorno, redirecciones (>), tuberías (|) y operaciones complejas del terminal de Linux.
- **-a "whoami"**: Es el argumento o comando que quieres que la terminal remota ejecute. El comando whoami devuelve el nombre del usuario que está ejecutando la sesión actual.

Los administradores lo usan principalmente para **diagnóstico**:

1. **Validación de Conexión:** Si el comando responde en todos los nodos, sabes que el SSH y las llaves están funcionando bien.
2. **Verificación de Permisos:** Te confirma si Ansible está entrando con el usuario que tú crees. A veces, si no configuras bien el ansible.cfg, Ansible podría intentar usar tu nombre de usuario local por defecto.
3. **Prueba de Entorno:** Al ser el módulo shell, te asegura que el entorno de ejecución (bash, zsh, etc.) está listo para recibir comandos.

ansible db_master -m debug -a "var=ansible_user"

Este comando es la contraparte del que usamos antes con msg. Mientras que msg sirve para imprimir un texto personalizado, el parámetro **var** sirve para inspeccionar el contenido crudo de una variable específica.

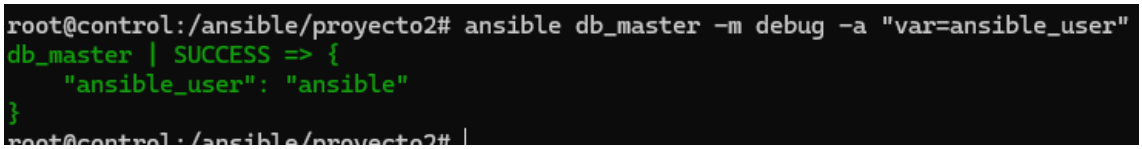
1. El objetivo del comando

- **ansible db_master**: Se dirige específicamente al host o alias llamado db_master.
- **-m debug**: Usa el módulo de depuración.
- **-a "var=ansible_user"**: Le pide a Ansible: "Búscame el valor de la variable interna ansible_user para este host y muéstramelo tal cual es".

2. ¿Qué información te está dando?

Este comando te confirmará **quién manda** en la jerarquía de configuración. Ansible buscará el valor de ansible_user siguiendo este orden:

1. ¿Está definido en host_vars/db_master.yml?
2. Si no, ¿está definido en el inventario (hosts.yml) al lado del host?
3. Si no, ¿está en group_vars/db_servers.yml o group_vars/madrid.yml?
4. Si no, ¿está en group_vars/all.yml?
5. Como último recurso, ¿cuál es el remote_user definido en el ansible.cfg?



Instalación “manual” de sudo en los nodos administrados.

Los nodos administrados no tienen “sudo” y por lo tanto no podrán instalar paquetería. Tampoco pueden instalar el propio paquete de sudo los usuarios “ansible”. Para solucionarlo, usamos el siguiente comando:

- **Instalación de “sudo”. Desde un terminal del host usamos:**

docker exec -u 0 ansible-node1 apt-get update && docker exec -u 0 ansible-node1 apt-get install -y sudo

docker exec -u 0 ansible-node2 apt-get update && docker exec -u 0 ansible-node2 apt-get install -y sudo

docker exec -u 0 ansible-node3 apt-get update && docker exec -u 0 ansible-node3 apt-get install -y sudo

- **Añadir el usuario “ansible” a la lista de “sudoers” para que pueda usar el comando “sudo”:**

docker exec -u 0 ansible-node1 sh -c "echo 'ansible ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers"

docker exec -u 0 ansible-node2 sh -c "echo 'ansible ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers"

docker exec -u 0 ansible-node3 sh -c "echo 'ansible ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers"

Playbooks

ansible-playbook -i hosts.yml install_nginx.yml

-i: indica el fichero de inventarios

Ejemplo Básico de Playbook

A continuación, un ejemplo sencillo de un playbook (install_nginx.yml) que instala el servidor web Nginx y asegura que esté iniciado en un grupo de servidores web:

```
---
- name: Instalar y configurar Nginx
  hosts: webservers
  become: yes
  tasks:
    - name: Actualizar caché de apt e instalar Nginx
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: yes
    - name: Iniciar y habilitar Nginx
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: yes
```

Estructura Principal del Play (La cabecera)

- **---**: Es el inicio estándar de cualquier archivo YAML.
- **name: Instalar y configurar Nginx**: Una descripción humana de lo que hace este conjunto de tareas. Aparecerá en tu terminal cuando lo ejecutes.
- **hosts: webservers**: Define el **Pattern**. Este Playbook solo se ejecutará en las máquinas que pertenezcan al grupo webservers de tu inventario.
- **become: yes**: Esta es la clave de los privilegios. Le dice a Ansible que ejecute todas las tareas como **root** (usando sudo). Es necesario porque instalar software o manejar servicios requiere permisos de administrador.

Las Tareas (tasks)

Ansible ejecuta estas tareas en orden, de arriba hacia abajo.

1. Instalación del software

- **ansible.builtin.apt**: Es el módulo para gestionar paquetes en sistemas basados en Debian/Ubuntu.
- **name: nginx**: El nombre del paquete.
- **state: present**: El concepto más importante. No le dices "instala", le dices "asegúrate de que esté presente". Si ya está instalado, Ansible no hará nada (**Idempotencia**).
- **update_cache: yes**: Equivale a ejecutar apt update antes de instalar, para asegurarse de bajar la versión más reciente.

2. Gestión del servicio

- **ansible.builtin.service**: El módulo para controlar servicios del sistema (systemd).
- **state: started**: Se asegura de que el servicio esté corriendo ahora mismo.
- **enabled: yes**: Se asegura de que el servicio se inicie automáticamente cada vez que el servidor se reinicie.

Controladores (Handlers)

Los handlers son tareas especiales que **solo se ejecutan si han sido "notificadas"** por otra tarea que haya realizado un cambio real (changed).

- **Uso típico:** Reiniciar un servicio (Nginx, MySQL) solo si su archivo de configuración fue modificado.
- **Directiva:** Se activan usando la palabra clave notify.

```
handlers:
  - name: Reiniciar SSH
    service:
      name: ssh
      state: restarted
```

Resumen de tu código:

- **name: Reiniciar SSH**: Es el identificador. Es vital que el nombre sea exacto al que uses en el notify.
- **service**: Es el módulo que gestiona servicios.
- **state: restarted**: La acción que realizará cuando sea notificado.

Condicionales (when)

Permiten que una tarea se ejecute solo si se cumple una condición específica. Esto es vital para manejar diferentes sistemas operativos o estados.

- **Ejemplo:** Instalar un paquete solo si la variable is_production es verdadera o si el sistema es de la familia "Debian".

```
- name: Instalar herramienta solo en Producción
  ansible.builtin.apt:
    name: vim
  when: inventory_hostname in groups['prod']
```

En tu ejemplo, estás condicionando la ejecución a la pertenencia de un host a un grupo específico del inventario.

- **inventory_hostname**: Es una variable interna de Ansible que contiene el nombre del servidor en el que se está ejecutando la tarea en ese momento (ej. node3).
- **groups['prod']**: Es un diccionario que contiene la lista de todos los servidores que pertenecen al grupo prod en tu archivo hosts.yml.
- **in**: Es un operador de comparación de Python que verifica si el nombre del servidor actual está dentro de esa lista de producción.

Bucles (Loops)

Sirven para repetir una misma tarea sobre una lista de elementos, evitando repetir código innecesariamente.

- **Directiva:** Se usa loop. Antes se usaba with_items, pero loop es el estándar actual.
- **Uso típico:** Crear múltiples usuarios o instalar una lista de paquetes de una sola vez.

```yaml
# 2. Gestión de usuarios masiva (Loop)
- name: Crear usuarios del equipo de auditoría
  user:
    name: "{{ item }}"
    groups: sudo
    append: yes
    state: present
  loop: "{{ audit_team }}"
```

1. ¿Cómo funciona la mecánica?

Cuando usas loop, Ansible toma la lista de datos que le proporciones y ejecuta la tarea tantas veces como elementos haya en esa lista.

- **loop: "{{ audit_team }}"**: Aquí le indicas a Ansible que busque una variable llamada audit_team. Esta variable debe ser una **lista** (por ejemplo, definida en group_vars o en el mismo playbook).
- **"{{ item }}"**: Esta es la variable mágica. Durante cada vuelta del bucle, Ansible mete el valor actual de la lista dentro de item.

2. Parámetros clave de la tarea user

- **groups: sudo**: Define a qué grupo se añadirá el usuario.
- **append: yes**: Esto es **vital**. Significa: "añade al usuario a este grupo *sin quitarlo* de los grupos en los que ya esté". Si pusieras append: no, el usuario perdería todos sus grupos actuales y solo se quedaría en sudo.
- **state: present**: Asegura que la cuenta de usuario exista.

Registros (register)

El registro permite **capturar la salida** de una tarea para usarla más adelante en el mismo playbook.

- Cuando ejecutas una tarea, Ansible genera un objeto JSON con el resultado (si falló, si cambió algo, qué devolvió el comando).
- Al "registrar" esa salida en una variable, puedes usarla en un condicional when de la siguiente tarea.

```
# 1. Auditoría de tiempo de actividad
- name: Ejecutar comando uptime
  command: uptime
  register: uptime_result
  changed_when: false
```

- **command: uptime**: Ansible ejecuta el comando en el servidor remoto.
- **register: uptime_result**: Ansible crea una variable llamada uptime_result (puedes ponerle el nombre que quieras) y mete dentro toda la información de la ejecución en formato JSON.
- **changed_when: false**: Como uptime es un comando de solo lectura (no cambia nada en el sistema), esta línea le dice a Ansible que siempre marque la tarea en **verde (ok)** en lugar de **amarillo (changed)**.

¿Cómo usas esa información después?

```
# 6. Registro de Log con variable dinámica
- name: Escribir resultado de uptime en log
  copy:
    content: "{{ uptime_result.stdout }}"
    dest: "{{ log_path }}"
  when: uptime_result.rc == 0
```

1. ¿De dónde vienen los datos?

- **content: "{{ uptime_result.stdout }}"**: Aquí es donde usas la variable que registraste antes.
    - uptime_result es el objeto completo.
    - .stdout es la "propiedad" que contiene solo el texto que verías en la consola (la salida estándar).
    - Ansible escribirá ese texto exacto dentro del archivo de destino.

- **dest: "{{ log_path }}"**: Esta es una variable (que probablemente definiste en tu ansible.cfg o group_vars). Indica la ruta del archivo donde se guardará el contenido (por ejemplo: /var/log/ansible_uptime.log).

2. El Condicional de Seguridad

- **when: uptime_result.rc == 0**: Esta línea es fundamental por seguridad.
    - rc significa **Return Code** (Código de Retorno).
    - En sistemas Linux/Unix, un comando que termina con éxito siempre devuelve un **0**. Si el comando falla (por ejemplo, si uptime no estuviera instalado), devolvería un número distinto de 0.
    - **Resultado:** La tarea de copiar solo se ejecutará si el comando anterior fue exitoso. Si el comando uptime falló, Ansible saltará esta tarea para evitar escribir basura o errores en tu archivo de log.

Roles de Ansible

- Los Roles de Ansible son la forma más efectiva de organizar y estructurar los playbooks.
- Un rol es un conjunto de tareas, controladores (handlers), variables, plantillas y archivos que se agrupan para automatizar una función específica.

Estructura de un Rol de Ansible

tasks/

Contiene el conjunto principal de tareas que ejecuta el rol. Se suelen dividir en archivos como main.yml, install.yml, configure.yml, etc.

handlers/

Almacena los controladores que solo se ejecutan cuando son "notificados"

vars/

Define variables específicas para el rol. Pueden estar en main.yml o en archivos específicos del sistema operativo, como Debian.yml o RedHat.yml.

templates/

Guarda archivos de plantilla Jinja2 que se procesan y personalizan antes de ser copiados a los nodos administrados.

files/

Contiene archivos estáticos que se copian directamente a los nodos administrados sin ninguna modificación, por ejemplo, scripts o claves SSH.

meta/

Incluye metadatos sobre el rol, como su descripción, autor, licencias y dependencias de otros roles de Ansible.

Uso de Roles en Playbooks

Para utilizar un rol, simplemente se referencia en un playbook. Ansible buscará automáticamente la estructura de directorios del rol y ejecutará sus tareas y controladores según sea necesario.

```
---
- name: Despliegue de servidor web
  hosts: webservers
  become: yes
  roles:
    - webserver_nginx
    - common_security
```

En este ejemplo, el playbook aplica dos roles: webserver_nginx para configurar el servidor web Nginx y common_security para aplicar configuraciones de seguridad comunes a los servidores del grupo webservers.

Instalar Roles:

Nos vamos a https://galaxy.ansible.com/ui/standalone/roles/

Instalas con: ansible-galaxy role install + nombre_rol

Ejemplo: **ansible-galaxy role install geerlingguy.apache**

Se instala en el ansible que ejerce de administrador:

```
root@control:/ansible/proyecto2# ansible-galaxy role install geerlingguy.apache
Starting galaxy role install process
- downloading role 'apache', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-apache/archive/4.2.0.tar.gz
- extracting geerlingguy.apache to /root/.ansible/roles/geerlingguy.apache
- geerlingguy.apache (4.2.0) was installed successfully
```

Para ver los roles instalados:

ansible-galaxy list

ansible-galaxy role list

```
root@control:/ansible/proyecto2# ansible-galaxy list
# /root/.ansible/roles
- geerlingguy.apache, 4.2.0
[WARNING]: - the configured path /usr/share/ansible/roles does not exist.
[WARNING]: - the configured path /etc/ansible/roles does not exist.
```

Para instalar el rol en un directorio específico:

ansible-galaxy install geerlingguy.apache -p ./roles

Borrar un rol:

ansible-galaxy role remove geerlingguy.apache

¿Dónde se borra el rol?

Cuando ejecutas ese comando, Ansible busca el rol en las rutas configuradas en tu sistema (por defecto suelen estar en ~/.ansible/roles o /etc/ansible/roles). Si el rol está ahí, borrará la carpeta completa.

Borrar un rol instalado en una carpeta específica:

- La forma "oficial" (vía comando):

ansible-galaxy role remove geerlingguy.apache -p ./roles

- La forma manual (borrando la carpeta)

rm -rf ./roles/geerlingguy.apache

- Forzar la reinstalación (actualizar):

ansible-galaxy role install geerlingguy.apache --force

- **¿Por qué es importante el parámetro -p?**

Ansible busca roles en un orden de prioridad. Al usar -p ./roles, creaste una estructura de proyecto "aislada".

- Si ejecutas ansible-galaxy role list, es posible que **no veas** el rol a menos que añadas el path: ansible-galaxy role list -p ./roles
- Si intentas borrarlo sin el -p, lo más probable es que recibas un mensaje diciendo: geerlingguy.apache is not installed, skipping. (aunque veas la carpeta ahí mismo).

Crear nuestro propio rol:

ansible-galaxy role init security_hardening

ansible-galaxy role init security_hardening --init-path ./prueba

¿Cómo se usa entonces un rol?

El rol se ejecuta cuando usas un Playbook. En este caso hemos creado el archivo "site.yml" en el directorio del proyecto.

```
root@control:/ansible/proyecto2# ls -l
total 48
-rwxr-xr-x 1 root root 2419 Jan 18 17:16 BACKUP_security_hardening.yml
-rwxr-xr-x 1 root root  401 Jan 18 16:32 HOSTS.BACKUP2
-rwxr-xr-x 1 root root  227 Dec 21 16:11 ansible.cfg
-rwxr-xr-x 1 root root  246 Dec 10 17:10 desktop.ini
drwxr-xr-x 2 root root 4096 Jan 21 13:30 group_vars
drwxr-xr-x 2 root root 4096 Jan 21 13:30 host_vars
-rwxr-xr-x 1 root root  337 Jan 18 16:28 hosts.BACKUP
-rwxr-xr-x 1 root root  337 Jan 18 16:32 hosts.yml
-rwxr-xr-x 1 root root  103 Dec 10 18:04 inventory
drwxr-xr-x 4 root root 4096 Jan 21 13:30 roles
-rwxr-xr-x 1 root root 1747 Jan 18 17:16 security_hardening.yml
-rwxr-xr-x 1 root root  283 Jan 21 12:50 site.yml
```

El archivo solo tiene que llamar a los roles y ellos se encargarán de hacer el trabajo. En nuestro caso hemos establecido los "hosts" a los que afectan, el uso de "sudo" y en el último la variable del puerto de apache.

```
---
- name: Configuraci  n Global de Seguridad
  hosts: all
  become: yes
  roles:
    - security_hardening

- name: Despliegue de Servidores Web
  hosts: web_servers
  become: yes
  vars:
    apache_listen_port: 8080  # Cambio de pu
  roles:
    - geerlingguy.apache
```

- ¿Cómo se ejecuta entonces?

ansible-playbook site.yml

Traspasar ficheros del host al contenedor:

- Del Host al contenedor

docker cp ./ansible_portable/proyecto2 ansible-control:/ansible

ansible-control: Es el nombre del contenedor Docker que debe estar corriendo

:/ansible: Es la ruta dentro del contenedor donde se pegarán los archivos.

- Copiar de dentro del docker a el host

docker cp ansible-control:/ansible ./ansible_actividad_3_files/

Obtener IP de los contenedores

docker inspect -f "{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}" ansible-control

docker network inspect bridge

docker exec ansible-control hostname -I

Instalar Ansible

Nodo Administrador

- Instalación de Python

$ sudo apt-get update

$ sudo apt-get install -y python-minimal

- Instalación de Ansible

$ sudo apt-get update

$ sudo apt-get install software-properties-common

$ sudo apt-add-repository --yes --update ppa:ansible/ansible

$ sudo apt-get install ansible

- Comprobación de instalación

python --version

ansible –version

Nodo Administrado

- Instalación de Python

$ sudo apt-get update

$ sudo apt-get install -y python-minimal

Variables más usadas en Ansible

Las variables que definas son las que te van a permitir controlar como te conectas a las distintas máquinas de tu entorno. Así, algunas de estas variables son las siguientes,

- ansible_host en caso de que el nombre de la máquina a la que te conectas sea distinto del alias que tu le has asignado.
- ansible_port para indicar al puerto en que al que te conectas, en el caso de que sea distinto del 22.
- ansible_user el nombre del usuario con el que te conectas
- ansible_password la contraseña del usuario con el que te conectas. Como sabes, yo no soy muy partidario de utilizar contraseña, sino mas bien, clave público privada.
- ansible_private_key_file se refiere a la clave público privada.
- ansible_become permite escalar privilegios
- ansible_become_user igual que el anterior, pero además para un usuario concreto.
- ansible_python_interpreter te permite indicar la ruta en la que se encuentra Python.