Variables

Ansible dispone de diferentes maneras de implementar variables como son las siguientes:

- **Variables de entorno**: Definen valores en el Sistema Operativo como es el caso de **ANSIBLE_CONFIG** que apunta al fichero de configuración a nivel global.
- **En los ficheros de inventario**: se incluyen en los ficheros de inventario y se pueden aplicar a nivel de hosts o de grupo de máquinas.
- **En los playbooks**: Donde podemos asociarlas a nivel de play e incluso de task (las más empleadas).
- **En los roles**: Se utilizan dentro de un rol (forma de organizar en plantillas los playbooks).
- **Variables especiales (magic, facts y conexión)**:
    - **Magic**: Variables que reflejan el estado interno del entorno y no son modificables.
    - **Facts**: Contiene información de un host, son devueltas por la máquina y nos dan información.
    - **Conexión**: Determinan como se ejecutan las acciones de un target, por ejemplo **ansible_become_user** que determina el usuario que ejecuta la tarea en el host remoto.
        - **ansible_host**: Cambia el nombre de la máquina.
        - **ansible_port**: Puerto para realizar la conexión, nomalmente el 22.
        - **ansible_user**: Usuario que realiza la conexión.
        - **ansible_become**: Permite escalar privilegios.
        - **ansible_become_user**: Usuario para la escalada de privilegios.
        - **ansible_python_interpreter**: Indica la ruta en la que se encuentra Python.

Variables en fichero de inventario

Lo normal en estos casos es que las variables se envien al playbook ya que este heredará las variables del fichero de inventario. En general lo que hacemos en el inventario es definir variables a utilizar en el playbook.

```
[servers]
srv1 ansible_host=192.168.10.6 ansible_user=ansible ansible_become=yes
srv2 ansible_host=192.168.10.5 ansible_user=ansible ansible_become=yes
```

```
all:
  children:
    servers:
      hosts:
        srv1:
          ansible_host: 192.168.10.6
          ansible_user: ansible
          ansible_become: yes
        srv2:
          ansible_host: 192.168.10.5
          ansible_user: ansible
          ansible_become: yes
```

```
ansible@debian64:~/practicas/pr02$ ansible -i hosts.yaml srv1 -m ping
```

Tambien puedo definir variables a nivel de grupo en el inventario de la siguiente forma.

```
[servers]
srv1 ansible_host=192.168.10.6
srv2 ansible_host=192.168.10.5

[servers:vars]
ansible_user=ansible
ansible_become=yes
```

```
all:
  children:
    servers:
      hosts:
        srv1:
          ansible_host: 192.168.10.6
        srv2:
          ansible_host: 192.168.10.5
      vars:
        ansible_user: ansible
        ansible_become: yes
```

Nota: No se recomienda, ni es válido, poner la sección [servers:vars] antes de definir el grupo [servers], porque las variables se asignan a los hosts definidos previamente, no a los que se definan más abajo.

Nota 2: Si queremos una variable para todos los grupos declaramos [all:vars].

Nota 3: En el inventario INI de Ansible, las variables de grupo se escriben una debajo de otra en la sección [grupo:vars], y las variables de host se pueden poner todas en una sola línea después del nombre del host.

Veamos unos ejemplos de ejecución partiendo del siguiente archivo de hosts en formato INI.

```
[servers]
srv1 ansible_host=192.168.10.6 ansible_user=martin
srv2 ansible_host=192.168.10.5

[servers:vars]
ansible_user=ansible
ansible_become=yes
```

Modificamos el fichero de inventario para que el usuario que realiza la conexión al host remoto sea otro usuario diferente a *ansible*, para ello usamos la variable de conexión **ansible_user**.

```
ansible@debian64:~/practicas/pr02$ ansible -i hosts2 servers -m ping
[ERROR]: Task failed: Failed to connect to the host via ssh: martin@192.168.10.6:
Permission denied (publickey,password).
Origin: <adhoc 'ping' task>

{'action': 'ping', 'args': {}, 'timeout': 0, 'async_val': 0, 'poll': 15}

srv1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Task failed: Failed to connect to the host via ssh:
martin@192.168.10.6: Permission denied (publickey,password).",
    "unreachable": true
}
srv2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false,
    "ping": "pong"
}
```

Fijemonos en que se produce un error en la conexión a *srv1* lo cual es normal ya que es el usuario *ansible* el usuario configurado para el par de claves.

Nota: Tambien podemos grupos hijos de otro y por lo tanto variables.

```
# Servidores Linux
[aula1]
server1
server2

[aula2]
server3

[taller]
server4

[plata1:children]
aula1
aula2

[planta1:vars]
ansible_user=antonio

[all:vars]
ansible_user=martin
```

- Los hosts se agrupan en aula1, aula2 y taller.
- El grupo planta1 incluye los subgrupos aula1 y aula2 como hijos.

- La variable ansible_user=antonio se asigna a todo planta1.
- La variable global ansible_user=martin aplica a todos los hosts, pero queda sobreescrita en planta1 por la variable de grupo.

Su equivalencia en yaml sería la siguiente:

```
all:
  children:
    aula1:
      hosts:
        server1:
        server2:
    aula2:
      hosts:
        server3:
    taller:
      hosts:
        server4:
    planta1:
      children:
        aula1:
        aula2:
      vars:
        ansible_user: antonio
  vars:
    ansible_user: martin
```

Variables en ficheros externos a nivel de inventario

Un punto importante es que podemos agrupar las variables en ficheros externos y a partir de estos ficheros podemos seleccionarlas. Estas variables se buscan a la misma altura que el fichero de inventario y están agrupadas en ficheros muy concretos. Estas variables estarán en un fichero *.yaml* dentro de un directorio llamado **host_vars**. Tambien podemos tener las variables agrupadas en un fichero *.yaml* dentro de un directorio llamado **group_vars**.

```
ansible@debian64:~/practicas/pr02$ tree
.
├── ansible.cfg
├── hosts
└── host_vars
    └── srv1.yaml
├── group_vars
│   └── servers.yaml

ansible@debian64:~/practicas/pr02$ cat hosts3
[servers]
srv1 ansible_host=192.168.10.6
srv2 ansible_host=192.168.10.5
```

```
ansible@debian64:~/practicas/pr02$ cat group_vars/servers.yaml
ansible_user: ansible
ansible_become: yes

ansible@debian64:~/practicas/pr02$ cat host_vars/srv1.yaml
ansible_user: martin
```

```
ansible@debian64:~/practicas/pr02$ ansible -i hosts3 all -m ping
[ERROR]: Task failed: Failed to connect to the host via ssh: martin@192.168.10.6:
Permission denied (publickey,password).
Origin: <adhoc 'ping' task>

{'action': 'ping', 'args': {}, 'timeout': 0, 'async_val': 0, 'poll': 15}

srv1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Task failed: Failed to connect to the host via ssh:
martin@192.168.10.6: Permission denied (publickey,password).",
    "unreachable": true
}
srv2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false,
    "ping": "pong"
}
```

Vemos que las variables definidas a nivel de host son más restrictivas y por tanto tienen una mayor prioridad a la hora de lanzar un módulo o playbook.

Como podemos ver, esta forma de definir variables nos permite tener un inventario mucho más limpio y ordenado.

Variables Facts

Son propiedades o datos que Ansible obtiene automáticamente de los sistemas que administra, conocidas como facts. Estos facts contienen información sobre el hardware, software, configuración y estado del sistema, como nombre del host, direcciones IP, versión del sistema operativo, interfaces de red, y más.

Además de los facts, Ansible trabaja con variables, que son valores que podemos definir para personalizar y controlar la ejecución de las tareas. Estas variables pueden adoptar varias estructuras:

- **Clave-valor simples**: pares donde una clave se asocia a un valor único.
- **Diccionarios (mapas)**, que se representan entre llaves {} y contienen pares clave-valor relacionados.
- **Listas o arrays**, que se representan entre corchetes [] y contienen colecciones ordenadas de elementos.

```
ansible@debian64:~/practicas/pr02$ ansible srv1 -m gather_facts
srv1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.10.6"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:fe69:d0fa"
        ],
        "ansible_apparmor": {
            "status": "enabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_vendor": "innotek GmbH",
        "ansible_bios_version": "VirtualBox",
        "ansible_board_asset_tag": "NA",
        "ansible_board_name": "VirtualBox",
        "ansible_board_serial": "NA",
        "ansible_board_vendor": "Oracle Corporation",
        "ansible_board_version": "1.2",
        "ansible_chassis_asset_tag": "NA",
        "ansible_chassis_serial": "NA",
        "ansible_chassis_vendor": "Oracle Corporation",
        "ansible_chassis_version": "NA",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-6.1.0-25-amd64",
            "quiet": true,
            "ro": true,
            "root": "UUID=af1a4503-9053-4546-b5c1-dcc1de1d4dc3"
        }
        ...
```

Ejemplo de par clave-valor.

```
ansible@debian64:~/practicas/pr02$ ansible srv1 -m gather_facts -a
"filter=ansible_system,ansible_os_family"
srv1 | SUCCESS => {
    "ansible_facts": {
        "ansible_os_family": "Debian",
        "ansible_system": "Linux",
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false
}
```

Ejemplo de diccionario.

```
ansible@debian64:~/practicas/pr02$ ansible srv1 -m gather_facts -a "filter
=ansible_proc_cmdline"
srv1 | SUCCESS => {
    "ansible_facts": {
        "ansible_proc_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-6.1.0-39-amd64",
            "quiet": true,
            "ro": true,
            "root": "UUID=af1a4503-9053-4546-b5c1-dcc1de1d4dc3"
        },
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false
}
```

Ejemplo de array.

```
ansible@debian64:~/practicas/pr02$ ansible srv1 -m gather_facts -a
"filter=ansible_processor"
srv1 | SUCCESS => {
    "ansible_facts": {
        "ansible_processor": [
            "0",
            "AuthenticAMD",
            "AMD Ryzen 5 5600H with Radeon Graphics"
        ],
        "discovered_interpreter_python": "/usr/bin/python3.11"
    },
    "changed": false
}
```

Variables dentro de playbooks

En esta parte vamos a hacer uso de un nuevo módulo que es *debug*.

Documentación módulo debug

El módulo debug en Ansible se utiliza para mostrar mensajes o valores de variables durante la ejecución de un playbook, lo que ayuda a la depuración y a entender el flujo o estado de la ejecución sin detener el playbook. Puede imprimir mensajes personalizados mediante el parámetro msg o mostrar el contenido de variables específicas usando el parámetro var.

Podemos tener un playbook que simplemente muestra un mensaje.

```
---
- name: Variables dentro de playbook
  hosts: srv1
```

```
  tasks:
    - name: Ver las variables
      ansible.builtin.debug:
        msg: Esto es una prueba nueva
```

O definir el mensaje haciendo uso de variables.

```
---
- name: Variables dentro de playbook
  hosts: srv1
  vars:
    mensaje: "Esto es un mensaje de Ansible"
    curso: "Curso de platega"

  tasks:
    - name: Ver las Variables
      debug:
        msg: "Texto de las variables: {{mensaje}} para el {{curso}}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook playbook.yaml

PLAY [Variables dentro de playbook] ********************************************

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

TASK [Ver las Variables] *******************************************************
ok: [srv1] => {
    "msg": "Texto de las variables: Esto es un mensaje de Ansible para el Curso de
platega"
}

PLAY RECAP *********************************************************************
srv1                       : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Hasta este punto vemos variables clave-valor pero podemos tener variables más complejas como diccionarios o arrays.

```
---
- name: Variables dentro de playbook
  hosts: srv1
  vars:
    mensaje: "Esto es un mensaje de Ansible"
    curso: "Curso de platega"
    entornos:
```

```
      - "desarrollo"
      - "producción"
      - "testing"
    equipos:
      tipo: "Linux"
      memoria: "8GB de RAM"
      almacenamiento: "250GB"

  tasks:
    - name: Variables clave-valor
      ansible.builtin.debug:
        msg: "Texto de las variables: {{mensaje}} para el {{curso}}"

    - name: Variables array
      ansible.builtin.debug:
        msg: "Entornos disponibles {{ entornos }}. El primer entorno es es {{
entornos[0] }} y los otros entornos son {{ entornos[1:3] }}"

    - name: Variables diccionario
      ansible.builtin.debug:
        msg: "Las características de los ordenadores de desarrollo son: {{ equipos
}} y tienen {{ equipos.almacenamiento }} de almacenamiento"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook playbook.yaml

PLAY [Variables dentro de playbook] ********************************************

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

TASK [Variables clave-valor] ***************************************************
ok: [srv1] => {
    "msg": "Texto de las variables: Esto es un mensaje de Ansible para el Curso de
platega"
}

TASK [Variables array] *********************************************************
ok: [srv1] => {
    "msg": "Entornos disponibles ['desarrollo', 'producción', 'testing']. El
primer entorno es es desarrollo y los otros entornos son ['producción',
'testing']"
}

TASK [Variables diccionario] ***************************************************
ok: [srv1] => {
    "msg": "Las características de los ordenadores de desarrollo son: {'tipo':
'Linux', 'memoria': '8GB de RAM', 'almacenamiento': '250GB'} y tienen 250GB de
almacenamiento"
}

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

```
srv1                       : ok=4    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Práctica de variables. Uso del fichero /etc/motd y variables en Ansible

En esta práctica vamos a trabajar con distintos tipos de variables. Vamos a usar el fichero **/etc/motd** de los sistemas Linux, que permite visualizar un mensaje de bienvenida al entrar en el sistema. Para ello, utilizaremos:

- El módulo **copy** de Ansible (módulo copy).
- La propiedad **content**, que permite copiar contenido directamente en lugar de copiar un fichero.
- Crear un playbook llamado **practicas.yaml**.
- Haz uso de variables para esta práctica.

```
---
- name: Práctica de variables
  hosts: srv1
  become: yes
  tasks:
    - name: Bienvenida
      ansible.builtin.copy:
        content: "Buenos días amigo, espero que estés bien"
        dest: /etc/motd
```

Probamos a ejecutar el playbook y nos conectamos por ssh al servidor de destino. Vemos el mensaje de bienvenida.

```
ansible@debian64:~$ ssh 192.168.10.6
Linux srv1 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26)
x86_64
Buenos días amigo, espero que estés bien
Last login: Tue Sep 23 19:17:45 2025 from 192.168.122.4
```

A continuación vamos a hacer uso de una variable que sea saludo, esta variable almacenará el mensaje de bienvenida *Buenos días amigo, soy una variable y espero que estés bien* para el srv1. Por este motivo, es necesario modificar el playbook.

```
---
- name: Practica de variables
  hosts: srv1
  become: yes
  vars:
    saludo: "Buenos días amigo, soy una variable y espero que estés bien"

  tasks:
    - name: Bienvenida
      ansible.builtin.copy:
```

```
        content: "{{saludo}}"
        dest: /etc/motd
```

```
ansible@debian64:~$ ssh 192.168.10.6
Linux srv1 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26)
x86_64
Buenos días amigo, soy una variable y espero que estés bien
Last login: Tue Sep 23 19:24:55 2025 from 192.168.122.4
```

Ahora la idea es trabajar con un diccionario con la siguiente formación.

- **Curso**: Administrar servidores con Ansible.
- **Plataforma**: Platega.
- **Destinatarios**: Docentes de la Consellería de Educación.
- **Responsables**: Martín, Antonio, Carlos.

Crea una nueva variable que sea información y al mensaje añade la información contenida en *Destinatarios* y el primer *responsable*.

```
---
- name: Practica de variables
  hosts: srv1
  become: yes
  vars:
    saludo:
      - "Buenos días amigo"
      - "soy una variable y espero que estés bien"
    informacion:
      curso: "Administrar servidores con Ansible"
      plataforma: "Platega"
      destinatarios: "Docentes de la Consellería de Educación"
      responsables: [Martín, Antonio, Carlos]

  tasks:
    - name: Bienvenida
      ansible.builtin.copy:
        content: "{{saludo[0]}}, {{saludo[1]}} y los destinatarios son
{{informacion.destinatarios}} siendo el responsable
{{informacion.responsables[0]}}"
        dest: /etc/motd
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook practica.yaml
...
ansible@debian64:~/practicas/pr03$ ssh server1
Linux server1 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26)
x86_64
Buenos días amigo, soy una variable y espero que estés bien y los destinatarios
```

```
son Docentes de la Consellería de Educación siendo el responsable Martín
Last login: Tue Sep 23 19:41:57 2025 from 192.168.10.6
ansible@server1:~$
```

Variables en tasks

Es interesante crear variables a nivel local, es decir, que únicamente funcionen en una tarea o task y no en todo el playbook. Un ejemplo de esto es lo siguiente.

```
---
- name: Variables locales a tasks
  hosts: srv1

  tasks:
  - name: Variable locales
    vars:
      variable_local: "local"
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_local }}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_locales.yaml

PLAY [Variables locales a tasks] ***********************************************

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

TASK [Variable locales] ********************************************************
ok: [srv1] => {
    "msg": "Esta es una variable local"
}

PLAY RECAP *********************************************************************
srv1                       : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Si nosotros invocamos a esta variable en otra tarea tendremos un error.

```
---
- name: Variables locales a tasks
  hosts: srv1

  tasks:
  - name: Variable locales
    vars:
      variable_local: "local"
    ansible.builtin.debug:
```

```
      msg: "Esta es una variable {{ variable_local }}"

  - name: Variable locales dos
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_local }}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_locales.yaml

PLAY [Variables locales a tasks] ***********************************************

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

TASK [Variable locales] ********************************************************
ok: [srv1] => {
    "msg": "Esta es una variable local"
}

TASK [Variable locales dos] ****************************************************
[ERROR]: Task failed: Finalization of task args for 'ansible.builtin.debug'
failed: Error while resolving value for 'msg': 'variable_local' is undefined

Task failed.
Origin: /home/ansible/practicas/pr03/variables_locales.yaml:12:5

10       msg: "Esta es una variable {{ variable_local }}"
11
12   - name: Variable locales dos
       ^ column 5

<<< caused by >>>

Finalization of task args for 'ansible.builtin.debug' failed.
Origin: /home/ansible/practicas/pr03/variables_locales.yaml:13:5

11
12   - name: Variable locales dos
13     ansible.builtin.debug:
       ^ column 5

<<< caused by >>>

Error while resolving value for 'msg': 'variable_local' is undefined
Origin: /home/ansible/practicas/pr03/variables_locales.yaml:14:12

12   - name: Variable locales dos
13     ansible.builtin.debug:
14       msg: "Esta es una variable {{ variable_local }}"
              ^ column 12

fatal: [srv1]: FAILED! => {"msg": "Task failed: Finalization of task args for
```

```
'ansible.builtin.debug' failed: Error while resolving value for 'msg':
'variable_local' is undefined"}

PLAY RECAP *********************************************************************
srv1                       : ok=2    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

Por otra parte, en caso de que exista una variable local y otra global con el mismo nombre, tiene preferencia la variable local sobre la global.

```
---
- name: Variables locales a tasks
  hosts: srv1
  vars:
    variable_local: global

  tasks:
  - name: Variable locales
    vars:
      variable_local: "local"
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_local }}"

  - name: Variable locales dos
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_local }}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_locales.yaml

PLAY [Variables locales a tasks] ***********************************************

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

TASK [Variable locales] ********************************************************
ok: [srv1] => {
    "msg": "Esta es una variable local"
}

TASK [Variable locales dos] ****************************************************
ok: [srv1] => {
    "msg": "Esta es una variable global"
}

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

Nota: En caso de tener variables definidas en un fichero externo estas tienen menor preferencia sobre las globales del propio playbook y sobre las locales a una tarea.

Variables de inventario en playbook

Podemos invocar a una variable del inventario como vimos anteriormente y siempre teniendo en cuenta que estas son aún más globales que una variable a nivel de playbook. Como ejemplo podemos ver el siguiente.

```
[servers]
srv1 ansible_host=192.168.10.6 ansible_user=ansible ansible_become=yes
variable_playbook="Variable en playbook"
srv2 ansible_host=192.168.10.5 ansible_user=ansible ansible_become=yes
```

```
---
- name: Variables locales a tasks
  hosts: srv1
  vars:
    variable_local: global

  tasks:
  - name: Variable locales
    vars:
      variable_local: "local"
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_local }}"

  - name: Variable locales dos
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_local }}"

  - name: Variable locales tres
    ansible.builtin.debug:
      msg: "Esta es una variable {{ variable_playbook }}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_locales.yaml

PLAY [Variables locales a tasks] ***********************************************

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

TASK [Variable locales] ********************************************************
ok: [srv1] => {
    "msg": "Esta es una variable local"
}

TASK [Variable locales dos] ****************************************************
ok: [srv1] => {
```

```
    "msg": "Esta es una variable global"
}

TASK [Variable locales tres] ***************************************************
ok: [srv1] => {
    "msg": "Esta es una variable Variable en playbook"
}

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

Variables externas al playbook

A diferencia de los inventarios donde tengo que declarar las variables de una forma concreta en unos directorios concretos, aquí puedo realizarlo de cualquier manera aunque es recomendable crear una carpeta de *recursos* y dentro los ficheros *.yaml* que contengan dichas variables.

```
---
- name: Variables externas al playbook
  hosts: srv1
  vars_files:
    - recursos/f1.yaml
    - recursos/f2.yaml

  tasks:
    - name: Variable del fichero f1.yaml
      ansible.builtin.debug:
        msg: "Esta es una variable {{ variable1 }}"

    - name: Variable del fichero f2.yaml
      ansible.builtin.debug:
        msg: "Esta es una variable {{ variable2 }}"
```

Fichero `./recursos/f1.yaml`.

```
variable1: "Variable 1"
```

Fichero `./recursos/f2.yaml`.

```
variable2: "Variable 2"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_externas.yaml

PLAY [Variables externas al playbook] ************************************

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

TASK [Variable del fichero f1.yaml] **************************************
ok: [srv1] => {
    "msg": "Esta es una variable Variable 1"
}

TASK [Variable del fichero f2.yaml] **************************************
ok: [srv1] => {
    "msg": "Esta es una variable Variable 2"
}

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

Nota: Igualmente válido es el uso de **vars_files** que hacer uso del módulo **include_vars** cuya documentación se adjunta en el siguiente enlace.

Documentación módulo include_vars

```
---
- name: Variables externas al playbook
  hosts: srv1

  tasks:
    # - name: Cargar variables desde f1.yaml
    #   ansible.builtin.include_vars:
    #     file: recursos/f1.yaml

    # - name: Cargar variables desde f2.yaml
    #   ansible.builtin.include_vars:
    #     file: recursos/f2.yaml

    - name: Cargar variables desde directorio
      ansible.builtin.include_vars:
        dir: recursos/

    - name: Variable del fichero f1.yaml
      ansible.builtin.debug:
        msg: "Esta es una variable {{ variable1 }}"

    - name: Variable del fichero f2.yaml
      ansible.builtin.debug:
        msg: "Esta es una variable {{ variable2 }}"
```

Nota: Más adelante veremos como podemos realizar este mismo ejemplo pero utilizando bucles.

Variables por linea de comandos

Podemos enviar variables desde CLI al propio playbook. Para ello tenemos que hacer uso en la ejecución de *--extra-vars* y la variable a utilizar.

```
---
- name: Variables por linea de comandos
  hosts: srv1
  # vars:
  #   mensaje: "MENSAJE VAR INTERNO"

  tasks:
  - name: Variable de línea de comandos
    ansible.builtin.debug:
      msg: "Mostramos la variable pasada desde CLI: {{mensaje}}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_por_cli.yaml --
extra-vars "mensaje='esta es la variable'"

PLAY [Variables por linea de comandos] *************************************

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

TASK [Variable de línea de comandos] ***************************************
ok: [srv1] => {
    "msg": "Mostramos la variable pasada desde CLI: esta es la variable"
}

PLAY RECAP *****************************************************************
srv1                       : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Variables de tipo Fact

Como dijimos estas son variables que nos dan información de las propiedades de los nodos a gestionar. Son variables muy interesantes y más adelante cuando queramos hacer ciertas tareas en unos nodos y no en otro serán de ayuda. Si nos fijamos, siempre que hacemos la ejecución de un playbook, aunque nosotros no creemos la tarea o invoquemos al módulo concreto se realiza una tarea de **[Gathering Facts]** como podemos ver a continuación. Esto permite dentro del propio playbook invocar a dichas variables para realizar cualquier acción en base a las características de los nodos a gestionar.

```
...
TASK [Gathering Facts]
```

```
**********************************************************************************
********
ok: [srv1]
...
```

Podemos hacer uso de estas propiedades de la siguiente forma.

```
---
- name: Variables de tipo FACT
  hosts: srv1

  tasks:
  - name: Ver varaibles de tipo FACT
    ansible.builtin.debug:
      msg: "La arquitectura es {{ansible_facts['architecture']}} o
{{ansible_facts.architecture}} y la ip es {{ansible_facts['all_ipv4_addresses']
[0]}} "

  - name: Usar diccionarios FACT
    ansible.builtin.debug:
      msg: "La información temporal es {{ansible_facts.date_time.day}} del
{{ansible_facts.date_time.month}}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_fact.yaml

PLAY [Variables de tipo FACT] **************************************************

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

TASK [Ver varaibles de tipo FACT] **********************************************
ok: [srv1] => {
    "msg": "La arquitectura es x86_64 o x86_64 y la ip es 192.168.10.6 "
}

TASK [Usar diccionarios FACT] **************************************************
ok: [srv1] => {
    "msg": "La información temporal es 11 del 11"
}

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

Nota: Hay veces que no queremos recopilar las variables FACTS, sobre todo si no las necesitamos y tenemos muchos entornos. Para ello podemo susar la opción **gather_facts=no**.

Ejemplo

```
- name: Proceso de instalación de Apache
  hosts: all
  gather_facts: no

  tasks:
  - name: Instalar Apache
    apt:
      name: apache2
      state: present
```

Variable debug_enabled y módulo include_tasks

Supongamos el siguiente ejemplo de playbook. En este playbook vamos a incluir un fichero llamado *config.yaml* para poder declarar variables externas a nuestro playbook, estas serán una variable **debug_enabled** para realizar las tareas de debug si así lo deseamos y que muestre por pantalla en la ejecución del playbook una información concreta. Por otra parte, hacemos uso del módulo **include_tasks** que permite incluir tareas definidas a parte en otro fichero que únicamente contiene tareas, estas están en *debian.yaml*

Fichero *config.yaml*:

```
ansible@debian64:~/practicas/pr03$ cat config.yaml
debug_enabled: yes
var1: "Soy una variable de un fichero externo"
```

Fichero *debian.yaml*:

```
ansible@debian64:~/practicas/pr03$ cat debian.yaml
- name: Listar archivos
  ansible.builtin.shell: ls -la
  register: resultado_ls

- name: Mostrar salida de ls con stdout_
  ansible.builtin.debug: var=resultado_ls.stdout      # ← cadena única

- name: Mostrar salida de ls con stdout_lines
  ansible.builtin.debug: var=resultado_ls.stdout_lines # ← lista (una línea por
elemento)
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook include_tasks.yaml

PLAY [Uso en playbook de when y variables del nodo a administrar para un Debian12]
***

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

```
ok: [srv1]

TASK [Leer configuración] ******************************************************
ok: [srv1]

TASK [debug var1] **************************************************************
ok: [srv1] => {
    "var1": "Soy una variable de un fichero externo"
}

TASK [debug ansible_os_family] *************************************************
ok: [srv1] => {
    "ansible_os_family": "Debian"
}

TASK [Listar directorios sobre Debian12] ***************************************
included: /home/ansible/practicas/pr03/debian.yaml for srv1

TASK [Listar archivos] *********************************************************
changed: [srv1]

TASK [Mostrar salida de ls con stdout_] ****************************************
ok: [srv1] => {
    "resultado_ls.stdout": "total 48\ndrwxr-xr-x 4 ansible ansible 4096 nov  6
14:18 .\ndrwxr-xr-x 5 root    root    4096 nov  6 13:48 ..\ndrwx------ 3 ansible
ansible 4096 nov  4 09:18 .ansible\n-rw------- 1 ansible ansible  433 nov  6 13:54
.bash_history\n-rw-r--r-- 1 ansible ansible  220 ago  4  2021 .bash_logout\n-rw-r-
-r-- 1 ansible ansible 3526 ago  4  2021 .bashrc\n-rw-r--r-- 1 root    root
53 nov  6 14:18 copia.txt\n-rw------- 1 ansible ansible   20 nov  5 23:36
.lesshst\n-rwxr-xr-x 1 ansible ansible   22 nov  6 14:20 playbook.txt\n-rw-r--r--
1 ansible ansible  807 ago  4  2021 .profile\n-rw-r--r-- 1 root    root       5
nov  6 14:18 saludo.txt\ndrwx------ 2 ansible ansible 4096 nov  4 08:57 .ssh"
}

TASK [Mostrar salida de ls con stdout_lines] ***********************************
ok: [srv1] => {
    "resultado_ls.stdout_lines": [
        "total 48",
        "drwxr-xr-x 4 ansible ansible 4096 nov  6 14:18 .",
        "drwxr-xr-x 5 root    root    4096 nov  6 13:48 ..",
        "drwx------ 3 ansible ansible 4096 nov  4 09:18 .ansible",
        "-rw------- 1 ansible ansible  433 nov  6 13:54 .bash_history",
        "-rw-r--r-- 1 ansible ansible  220 ago  4  2021 .bash_logout",
        "-rw-r--r-- 1 ansible ansible 3526 ago  4  2021 .bashrc",
        "-rw-r--r-- 1 root    root      53 nov  6 14:18 copia.txt",
        "-rw------- 1 ansible ansible   20 nov  5 23:36 .lesshst",
        "-rwxr-xr-x 1 ansible ansible   22 nov  6 14:20 playbook.txt",
        "-rw-r--r-- 1 ansible ansible  807 ago  4  2021 .profile",
        "-rw-r--r-- 1 root    root       5 nov  6 14:18 saludo.txt",
        "drwx------ 2 ansible ansible 4096 nov  4 08:57 .ssh"
    ]
}

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

```
srv1                       : ok=8    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Variables registradas

Nos servirán para obtener una variable y hacer uso de ella en el propio playbook. Este playbook ejecuta el comando `date` en el servidor **server1** y guarda (con `register`) el resultado en la variable `fecha`. Luego usa `debug` de dos formas:

- `var: fecha` muestra toda la estructura de datos que Ansible almacenó (stdout, stderr, rc, etc.).
- `msg: "{{ fecha }} y {{ fecha.stdout }}"` imprime la variable completa y, además, solo la salida estándar (la fecha exacta).

```
---
- name: Prueba con Variables Registradas
  hosts: srv1

  tasks:
  - name: Capturar fecha
    ansible.builtin.shell:
      cmd: date
    register: fecha

  - name: Visualizar fecha
    ansible.builtin.debug:
      var: fecha

  - name: Visualizar fecha
    ansible.builtin.debug:
      msg: "{{fecha}}} y {{fecha.stdout}}"
```

```
ansible@debian64:~/practicas/pr03$ ansible-playbook variables_registradas.yaml

PLAY [Prueba con Variables Registradas] ************************************

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

TASK [Capturar fecha] ******************************************************
changed: [srv1]

TASK [Visualizar fecha] ****************************************************
ok: [srv1] => {
    "fecha": {
        "changed": true,
        "cmd": "date",
        "delta": "0:00:00.005172",
        "end": "2025-11-11 08:18:17.177872",
```

```
        "failed": false,
        "msg": "",
        "rc": 0,
        "start": "2025-11-11 08:18:17.172700",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "mar 11 nov 2025 08:18:17 CET",
        "stdout_lines": [
            "mar 11 nov 2025 08:18:17 CET"
        ]
    }
}

TASK [Visualizar fecha] ********************************************************
ok: [srv1] => {
    "msg": "{'changed': True, 'stdout': 'mar 11 nov 2025 08:18:17 CET', 'stderr':
'', 'rc': 0, 'cmd': 'date', 'start': '2025-11-11 08:18:17.172700', 'end': '2025-
11-11 08:18:17.177872', 'delta': '0:00:00.005172', 'msg': '', 'stdout_lines':
['mar 11 nov 2025 08:18:17 CET'], 'stderr_lines': [], 'failed': False}} y mar 11
nov 2025 08:18:17 CET"
}

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