



PLAYBOOKS DE ANSIBLE

Publicado el [5 de julio de 2020](#) por atareao 13.3K [Deja un comentario](#)



! Este es **uno** de los capítulos del tutorial [Automatización con Ansible. Una introducción](#). Encontrarás los enlaces a todos los de capítulos, [al final de este artículo](#).

Hasta el momento, has podido utilizar Ansible para realizar operaciones mas o menos sencillas de forma aislada. Me refiero a que has podido borrar un archivo o crear un directorio, por ejemplo. Sin embargo, no has podido realizar varias tareas encadenadas. Esto no quita con que lo que has visto hasta el momento ya tiene gran atractivo, porque te permite realizar esas operaciones en tantas máquinas como necesites, en tantas máquinas como quieras. Imagina que tuvieras que apagar un centenar de máquinas y tuvieras que hacerlo entrando una a una... En este capítulo del tutorial, vas a dar un salto tanto cualitativo como cuantitativo, vamos a acometer los **playbooks de Ansible**.

Y ¿porque te digo que daremos un salto cualitativo y cuantitativo con los Playbooks de Ansible? Por el simple hecho de que, por un lado, vas a poder acometer varias tareas en un solo proceso, y por otro lado, vas a poder hacer realizar tareas mas complejas. Creo que este es el punto de partida de toda una revolución para lo que normalmente haces.

Pero además, otra de las grandes ventajas de los playbooks, es que te ofrecen la posibilidad de someterlos a control de versiones. Piénsalo, realizar una instalación de un stack como *Telegraf*, *InfluxDB* y *Grafana*, que comenté en el episodio 188 del podcast sobre los [paneles de control a medida](#), pero en un control de versiones. De esta forma, cualquier modificación al proceso de instalación, la tienes controlada, y documentada.

PLAYBOOKS DE ANSIBLE

Como te decía en la introducción, todas las operaciones que has realizado hasta el momento ha sido utilizando los [comandos ad-hoc](#). Ahora, verás una forma, completamente diferente de utilizar Ansible, de lo que has podido ver hasta el momento. Una manera de utilizar Ansible verdaderamente **potente**.

¿QUE ES UN PLAYBOOK?

Un **playbook** es una lista de **plays** o **actos** o **actos**. Y un play es una lista de tareas que se aplican sobre un determinado conjunto de hosts. Si solo tienes un conjunto de hosts, el playbook coincidirá con ese *play*. Cada tarea no es mas que **una llamada a un módulo de Ansible**. Digamos que cada tarea es como *un comando ad-hoc*.

De nuevo me enfrento a una traducción. Lo cierto, es que no he encontrado literatura al respecto, y tampoco, es que haya buscado, mucho. Sin embargo, me parece una muy buena traducción la de

play por **acto**. De esta manera, a partir de ahora, encontrarás indistintamente, *play* o *acto* para referirse a esa lista de tareas.

LO MAS BÁSICO

Para cada uno de los plays de tu playbook, deberías definir cuales son los host implicados y el usuario que va a realizar cada una de las tareas del *play*. Por ejemplo, un playbook básico, con un único *play*, tendría un aspecto como el que te muestro a continuación,

```
---
- hosts: servidores
  remote_user: root
```

Así, por ejemplo, tu primer playbook, eso si con un solo *acto* (*play*), podría tener el siguiente aspecto,

```
---
- hosts: all
  remote_user: root
  tasks:
    - name: hacer un ping
      ping:
```

Para ejecutarlo simplemente lanza la siguiente instrucción en un terminal,

```
ansible-playbook -i inv playbooks/ejemplo.yml
```

Indicarte que yo tengo el inventario en un directorio llamado `inv` y todos los playbooks en un directorio llamado `playbooks`.

Al ejecutarlo, obtendrás algo similar a lo que te muestro a continuación,

```
PLAY [all] *****

TASK [Gathering Facts] *****
ok: [io03]
ok: [io01]
ok: [do01]
ok: [co01]

TASK [hacer un ping] *****
ok: [io03]
ok: [io01]
ok: [co01]
ok: [do01]

PLAY RECAP *****
co01: ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
do01: ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
io01: ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
io03: ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

En tu *acto*, en tu *play*, solo habías puesto una tarea, la de hacer *ping*, sin embargo se han ejecutado

dos tareas. La primera de las tareas, que ya comenté en el capítulo de [comandos ad-hoc](#), se hace de forma automática, y simplemente recoge información del *host* remoto. Esto, podrías suprimirlo modificando tu *play* añadiendo la opción `gather_facts: no`, de forma que el playbook, te quedará con el siguiente aspecto,

```
---
- hosts: all
  remote_user: root
  gather_facts: no
  tasks:
    - name: hacer un ping
      ping:
```

Otra cuestión, es que la ejecución de las *tareas* en cada uno de los *actos* (plays), se realiza en el orden indicando en el inventario. Este orden de ejecución lo puedes modificar, añadiendo la opción `order: sorted`. Otros posibles valores son `inventory`, `reverse_inventory`, `sorted`, `reverse_sorted` y `shuffle` (aleatorio). Esto no afecta al acto de resumen `PLAY RECAP` donde se muestran los resultados y que por lo general lo hará en orden alfabético.

Si solo lo quisieras ejecutar en uno de los hosts, en lugar de ejecutarlo en todos simplemente tienes que añadir la opción `limit`. Por ejemplo,

```
ansible-playbook -i inv playbooks/sample01.yml --limit co01
```

Con la anterior instrucción solo ejecutará el `playbook` en el host `co01`. Esto mismo lo puedes ejecutar de una forma mas simplificada con,

```
ansible-playbook -i inv playbooks/sample01.yml -l co01
```

ALGUNAS CUESTIONES BÁSICAS

Lo primero es comentar el `remote_user`, que no es mas que la cuenta de usuario. En general, creo una cuenta de usuario para el usuario `ansible`, y realizo todas las operaciones con este usuario. Sin embargo, es posible indicar el `remote_user` tanto por playbook como por tarea. De la misma forma, también puedes ganar derechos utilizando `become`, tanto por playbook como por tarea.

Otra cuestión interesante es el uso de variables. Y es que esto te va a permitir reutilizar los playbooks de forma sencilla, y sin necesidad de modificarlos, sobre todo si las variables las tienes en un archivo separado. ¿Como utilizar las variables? Un ejemplo,

```
---
- hosts: all
  gather_facts: no
  vars:
    dname: test
    dname2: casa
  tasks:
    - name: create a directory
      become: yes
      file:
```

```
path: /etc/{{ dname }}
state: directory
```

SOBRE VARIABLES

A la hora de buscar nombres de variables, indicarte que no todos son válidos, sino que tienen que cumplir unos requisitos básicos. Así pueden estar contruidos con una combinación de números, letras y guión bajo, con la condición de que siempre deben empezar con una letra y no pueden contener espacios.

También es posible que una variable sea un *array*, como en el siguiente ejemplo,

```
---
- hosts: all
  gather_facts: no
  vars:
    cars:
      - Citroen
      - Seat
  task:
    - name: lists
      debug:
        msg: "{{ cars[1] }}"
```

Igual que tienes *array* también puedes utilizar variables que sean diccionarios. Por ejemplo,

```
---
- hosts: all
  gather_facts: no
  vars:
    users:
      - user1:
          name : Jose
          apellido1: Garcia
      - user2:
          name: Juan
          apellido1: Fernández
```

Esto, lo puedes utilizar posteriormente como `user1.name` o también con el formato `user1['name']`, como tu prefieras.

Por otro lado, otra opción que tienes para la definición de variables es utilizar un archivo externo, lo cual siempre quedará mas limpio y seguro. Digo seguro, en tanto en cuanto, en el caso de modificaciones, no necesitas *tocar* el cuerpo de tu *playbook*.

Así, para definir las variables en un archivo externo, lo puedes hacer de la siguiente forma,

```
---
- hosts: all
  gather_facts: no
  var_files:
    - /vars/external_vars.yml
```

Y el contenido de ese archivo es similar a lo que has visto hasta el momento,

```
---
# variables
var1: variable1
var2: variable2
var3:
  - Citroen
  - Seat
  - Renault
```

CONCLUSIÓN

Con esto ya tienes una idea de como crear tus propias listas de tareas y como utilizar variables para simplificar el trabajo. Sin embargo, esto de los playbooks todavía tiene recorrido, en tanto en cuanto puedes, definir bucles y utilizar condicionales para mejorar ese flujo de trabajo.

Listado de capítulos

Estos son todos los capítulos del tutorial [Automatización con Ansible. Una introducción.](#),

[1.- Instalar Ansible. Conceptos básicos y primeros pasos](#)

[2.- YAML el reemplazo de JSON](#)

[3.- El inventario de Ansible](#)

[4.- Comandos ad-hoc](#)

[5.- Comandos básicos de Ansible](#)

6.- Playbooks de Ansible

[7.- Bucles y condicionales en Ansible](#)

[8.- Bloques en Ansible](#)

[9.- Handlers en Ansible](#)

[Comandos básicos de Ansible](#)

[Bucles y condicionales en Ansible](#)



COMANDOS BÁSICOS DE ANSIBLE

En este capítulo encontrarás algunos comandos básicos de Ansible que te facilitarán enormemente el trabajo en tu infraestructura.



BUCLES Y CONDICIONALES EN ANSIBLE

Ansible no solo te permite realizar tareas de forma secuencial, sino que pone a tu disposición bucles y condicionales para ejecutar solo las necesarias