playbooks

2.11 introducción Playbooks

- lista de jugadas (tareas) en una lista de servidores
- configuraciones y variables
- formato YAML:

  playbook.yml

```
---
- name: Mi primer playbook
  hosts: all
  remote_user: <usuario>
  become: true # a nivel de playbook, se podría hacer a nivel de tarea
  tasks:
    - name: copiar ficheros hosts
      copy: src=/etc/hosts dest=/etc/host
    - name: ...
      service: ...
```

    - cada guión es un playbook, puede haber varios en un fichero
    - **name** es opcional, pero recomendado
- `ansible-playbook [-i inventario] [ociones] playbook.yml`

2.12 Esenciales

- hosts: lista de servidores a administrar (grupos o servidores)
    - si se quieren separar, usar **:**
    - se puede usar **&** para que estén en 2 grupos a la vez
    - se puede usar **!**

```
hosts: serweb # un grupo, afecta a todas las máquinas
hosts: serweb:&barcelona # dos grupos, solo los que pertenecen a los dos grupos a la vez se verán afectados
hosts: serweb:!madrid # 2 grupos, solo los que pertenezcan al primero y no estén en el segundo se verán afectados
hosts: serweb:dbweb # afecta a los dos grupos
```

```
# usamos el inventario del directorio inventarios
# usamos el fichero miPlaybook.yml
# usamos --lists-hosts para ver a que servidores afectará
ansible-playbook -i inventarios/ miPlaybook.yml --list-hosts

# solo serweb+barcelona
ansible-playbook -i --limit 'serweb:&barcelona' inventarios/ miPlaybook.yml --list-hosts
```

- remote_user: <usuario>
- become: [True|False] | [1|0]

- become_user: <usuario> (que queremos usar)
- become_method: sudo/su/pbrun/ksu
- check_mode: [True|False] → simulación

2.13 ansible-playbook

```
ansible-playbook [opiones] fichero.yml
```

- opciones:
  - `-i` : especificar un inventario (fuera de **/etc/ansible/hosts**)
  - `–syntax-check`
  - `–list-tasks`
  - `–list-hosts`
  - `–step` : confirmar cada una de las tareas, cada uno de los playbooks
  - `–start-at-task=<tarea>` : permite saltarse tareas, le indicamos a partir de cual
  - `–forks|[-f]=#` : número de tareas en paralelo a ejecutar (por defecto, 5)
  - `-v | -vv | -vvv` : más verbosidad

2.14 Variables

- dinamización de tareas
- pueden ser definidas en:
  - **facts** : obtenidos del servidor (sobre el que aplicamos la configuración)
  - **playbook**
  - **línea de comandos** : `-e` → clave=valor
- se pueden usar en:
  - tareas
  - plantillas (lógica)
  - otros (condiciones, bucles, roles, etc..)
- uso:''{{variable}}'' (con dobles comillas, dobles llaves)

ejemplo

template sencillo, definimos una variable propia y las otras dos las da el propio servidor

hosts.j2

```
{{ miip }} {{ ansible_hostname}} {{ ansible_fqdn }}
```

playbook.yml

```
- name: crear ficher usando variables
  hosts: localhost
  connection: local
  vars:
    - miip: "1.2.3.4"
  tasks:
    - name: Crear fichero hosts
      template: src=hosts.j2 dest=/tmp/hosts
```

ejecución

```
ansible-playbook playbook.yml
# el módulo template se encarga de buscar el fichero que necesita y hacer las sustituciones
```

lista de variables disponibles en el server

```
ansible localhost -m setup | grep -e ansible_hostname -e ansible_fqdn
```

sobreescribiendo las variables del playbook

```
ansible-playbook -e miip="5.6.7.8" playbook.yml
```

2.15 Sintaxis

- formato YAML:
  - simplifica definición playbooks
  - diferentes opciones para mejorar la legbilidad

ejemplos

- partir línea de configuración larga:

ejemplo línea larga

```
- name: copiar fichero
  copy: src=/etc/hosts dest=/etc/hosts owner=root group=root mode=0644
```

ejemplo mejorado

```
- name: copiar fichero
  copy:
    src: /etc/hosts
    dest: /etc/hosts
    owner: root
    group: root
    mode: 0644
```

- definir lista de variables:

lista de variables

```
- instalar:
  - apache
  - php5
  - mariadb
```

```
- instalar: ["apache","php5","mariadb"]
```

- definir diccionario:

diccionario

```
- instalar:
  web: apache2
  bd: mariadb
  script: php5
-instalar: {web:"apache", bd:"mariadb",script:"php5"}
```

para hacer referencia, se usará `instalar.web` o `instalar.script`

- texto / líneas de texto largas

texto largo

```
- texto_largo: |
    primera linea
    segunda linea
- linea_larga: >
    primera parte
    segunda parte
```

- módulo **debug** : muestra un texto o un valor de una variable

debug

```
- debug: var=miip
- debug: var=texto_largo
- debug: linea_larga
- debug: instalar.web
```

2.16 Handlers

es una tarea que se ejecuta solo cuando la llama otra tarea

```
- tasks:
  - name: configurar sshd.config
    copy: src=sshd_config dest=/etc/sshd.config
    notify: reiniciar_sshd
- handlers:
  - name: reinicar_sshd
    service: name=sshd state=restarted
```

los **handlers** se ejecutan al final de las tareas del **playbook**

2.17 Include y Roles

include

- dividir el distintas partes para facilitar edición y tratado
- **include** permite añadir otro fichero que contiene un playbook o una tarea

ejemplos

- inclusión de una tarea:

```
- name: primer playbook
  hosts: servweb
  tasks:
    - name: tarea de este fichero
      apt: name=apache2 state=latest
    - include: otra_tarea.yml
```

otra_tarea.yml

```
- name: tarea en fichero anexo
  service: name=nombre state=started
```

- inclusión de otro playbook:

```
- name: primer playbook
  hosts: servweb
  tasks:
    - name: tarea de este fichero
      apt: name=apache2 state=latest
- name: Segundo playbook
  include: segundo_playbook.yml
```

segundo_playbook.yml

```
- hosts: servweb
  tasks:
    - name: tarea de este fichero
      apt: name=vim state=latest
```

roles

- estructura de ficheros y directorios para separar los elementos
- permite ser reusados facilmente
- es posible descargar roles predefinidos : Ansible Galaxy
- roles/
    - <nombre>/
        - files/

        - templates/
        - tasks/main.yml ←- único obligatorio
        - handlers/main.yml
        - vars/main.yml
        - defaults/main.yml
        - meta/main.yml
- uso en el playbook:

```
- roles:
  - rol1
  - rol2
```

paso de variables

```
- roles:
  - {role: rol1, clave: valor}
  - rol2
```

2.18 Templates

- instrucciones en tamplates:
    - expresiones:
                                                variable
    - control: {% ... %}
        - condicional:

```
{% if ansible_distribution == "Debian" %}
si se cumple la condición
{% endif %}
```

        - bucle:

```
{% for usuario in lista_usuarios %}
{{ usuario }}
{% endfor %}
```

roles/apache2/defaults/main.yml

```
lista_usuarios: ["root","alberto","www-data"]
```

```
roles:
  - { role: apache2, lista_usuarios: ["usuario1","root1"] }
```

        - comentarios: {# comentario #}

2.19 Prioridad variables

en orden de menos a mas

1. **Defaults** dentro de un **rol**

2. variables de grupo (inventario→group_vars/all→group_vars/<grupo>)
3. variables de servidor (inventario→host_vars/<servidor>)
4. **facts** del servidor
5. variables del **playbook** (→vars_prompt→vars_files)
6. variables del **rol** (definidas en /roles/<rol>/vars/main.yml)
7. variables de bloque → variables de tareas
8. parámetros **rol** → include_parvars → include_vars
9. set_facts / registered_vars
10. extra vars → `-e`

2.20 Condiciones

se puede condicionar:

- ejecución de una tarea
- inclusión de un fichero
- el uso de un rol

usando la expresión when

```
- name: instalar apache2
  apt: name=apache2 state=latest
  when: ansible_distribution == "Debian"
```

```
- name: instalar apache2
  apt: name=apache2 state=latest
  when: ansible_distribution == "Debian" or ansible_distribution == "Ubuntu"
```

archivo según versión

```
- name: Instalar apache2
  include: instalar-apache2.yml
  when: ansible_distribution == "Debian" or ansible_distribution == "Ubuntu"

- name: Instalar httpd
  include: instalar-httpd.yml
  when: ansible_distribution == "CentOS"
```

filtrado a través de rol

```
- { role: apache2, when: ansible_distribution == "Debian" or
ansible_distribution == "Ubuntu" }
- { role: apache2, when: ansible_distribution == "CentOS" }
```

2.21 Bucles

ya vimos como usarlo en las plantillas ()

ahora en tareas (palabra clave): `with_items`

listas

```
- name: Instalar soft necesario
  apt: name={{ item }} state=latest
  with_items:
    - mariadb
    - php5
    - phpmyadmin
```

diccionarios

```
- name: Crear usuarios
  user: name={{ item.nombre }} state=present groups={{ item.grupo }}
  with_items:
    - {nombre: usuario1, grupo: www-data}
    - {nombre: usuario2, grupo: www-data}
```

con variables (en roles/<rol

```
/defaults/main.yml)>
- name: Instalar paquetes
  apt: {{ item }}
  with_items: "{{ lista_paquetes }}"
- name: Crar usuarios
  user: {{ item }}
  with_items: "{{ dic_usuarios }}"
```

variables en main.yml

```
lista_paquetes: ["php5","phpmyadmin","mysql-server"]
dic_usuarios:
  - {nombre: "usuario1", grupo: "www-data"}
  - {nombre: "usuario2", grupo: "www-data"}
```

2.22 Register

`register` nos permite guardar en una varibale el resultado de la acción realizada por un módulo en una tarea

```
- name: ejecutar comando
  command: uptime
  register: salida_uptime
- name: mostrar uptime
  debug: var=salida[.stdout]
```

los valores devueltos son:

- changed
- failed
- skipped
- rc

    - == 0 → OK
    - != 0 → KO
- stdout / stderr / stdout_lines / stderr_lines
    - en lines se podría rehusar como bucle

y se pueden usar esos estados para filtrar la ejecución de otras tareas

solo ejecutará la segunda tarea si ha habído un changed en la primera

```
- name: copiar fichero index.html
  template: src=index.html.j2 dest=/var/www/html/index.html
  register: out_copia
- name: mostrar contenido
  command: cat /var/www/html/index.html
  register: out_salida
  when: out_copia|changed
- debug: var=out_copia.stdout
  when: out_copia|changed
```

2.23 Ignore Errors

las tareas erróneas paran el playbook, usando `ignore_errrors = True` evitamos que la tarea errónea pare la ejecución

```
- name: comprobar si fichero existe
  command: ls /noexiste.conf
  registar: existe
  ignore_errors = true
- name: mostrar errores
  debug: var=existe.stderr_lines
```

se puede usar a nivel de tarea o de playbook ← CUIDADO! todos los errores serán ignorados

se pueden usar condicionales `not <var_register>|failed` para realizar tareas complementarias en caso de error

```
- name: mostrar salida
  debug: var=existe.stdout_lines
  when: not salida|failed
[  when: salida|failed != False ] # expresión equivalente
- name: mostrar errores
  debug: var=existe.stderr_lines
  when: salida|failed
```

2.24 Failed When

`failed_when` o `changed_when` permiten especificar las condiciones para marcar una tarea como fallida o cambiada, respectivamente

un comando es marcado como erróneo si su **rc** (return code) es != 0 → $?

siempre retornará changed

```
- name: ejecutar comando
  command: ip a
```

da error si no existe la interfaz eth2

```
- name: ejecutar comando
  command: ip a
  register: salida
  failed_when: "'eth2' not in salida.stdout"
```

no la marca como cambiada

```
- name: no marcar nunca como cambiado
  command: uptime
  changed_when: False
```

se pueden combinar en la misma tarea juntas

From:
https://matebcn.eu/ - **miguel angel torres egea**

Permanent link:
https://matebcn.eu/info:cursos:udemy:ansible:playbooks

Last update: **19/09/2018 09:54**