DO407-AUTOMATION WITH ANSIBLE I

DÍA 1	**1.- Introducción al curso** Introducción y revisión del curso. **2.- Introducción a Ansible** Descripción de la terminología y la arquitectura de Ansible. **3.- Implementando Ansible** Instalación de Ansible y ejecución de comandos ad-hoc.
DÍA 2	**4.- Implementación de playbooks** Redacción de ficheros y ejecución de un playbook de Ansible.
DÍA 3	**5.- Gestión de variables e inclusiones** Descripción del alcance y la prioridad de una variable, gestión de variables y datos en un fichero, y administración de inclusiones. **6.- Implementación de control de tareas** Administración de control de tareas, gestores y etiquetas en los playbooks de Ansible.

DÍA 4	**7.- Implementación de plantillas Jinja2** Implementación de una plantilla Jinja2.
DÍA 5	**8.- Implementación de roles** Creación y administración de roles.
DÍA 6	**9.- Optimizando Ansible** Configuración de tipos de conexiones, delegaciones y paralelismos. **10.- Implementación de Ansible Vault** Gestión de cifrado con Ansible Vault.

DÍA 7	**11.- Solución de problemas de Ansible** Solución de problemas de la máquina de control y los nodos administrados de Ansible. **12.- Implementación de Ansible Tower** Implementación de Ansible Tower.
DÍA 8 **DÍA 9**	**13.- Implementación de Ansible en un entorno DevOps** Implementación de Ansible en un entorno DevOps mediante el uso de Vagrant. Repaso completo Repaso de las tareas del curso Automation with Ansible.

1.- INTRODUCCIÓN AL CURSO

- ✓ Es un curso altamente práctico.
- ✓ Se basa en el temario de la certificación oficial de Ansible RedHat.
- ✓ Está orientado a Administradores de Sistemas.
- ✓ Utiliza VM de CentOS 7.5 en VirtualBox para realizar los ejemplos.
- ✓ Utiliza un workbook con los comandos a ejecutar para una rápida ejecución de los ejemplos.
- ✓ Se utiliza la última versión de Ansible.
- ✓ Los ejemplos se realizarán conectándonos entre distintas VM de la clase, para simular un entorno productivo real.

2.- INTRODUCCIÓN A ANSIBLE

- ✓ Lanzamiento inicial el 20 de febrero de 2012.
- ✓ Adquirido recientemente por Red Hat.
- ✓ Herramienta gratuita de código abierto para la gestión de configuración y orquestación de infraestructuras.
- ✓ "Arquitectura agentless" No require un agente en los servidores cliente, por lo que optimiza el rendimiento y evita abrir puertos especiales, sólo utiliza conexión por SSH.
- ✓ Sintaxis muy simple y fácil de aprender, no necesita conocimientos de programación.
- ✓ Hay más de 1000 módulos disponibles para activos diferentes para hablar el mismo idioma del dispositivo administrado y permite desarrollar modulos en pyton.

2.1 QUE ES ANSIBLE:

Ansible es una utilidad de orquestación y administración de configuración de código abierto. Puede automatizar y estandarizar la configuración de hosts remotos y máquinas virtuales. Su funcionalidad de orquestación permite a Ansible coordinar el lanzamiento y el apagado seguro de aplicaciones de varios niveles.

En lugar de escribir scripts personalizados e individualizados, los administradores del sistema crean un "play" o una jugada de alto nivel en Ansible. Un play realiza una serie de tareas en el host, o grupo de hosts, especificados en el play. Un archivo que contiene uno o más play's se denomina playbooks.

La arquitectura de Ansible es agentless "No tiene agente instalado en cada cliente". El trabajo se envía a los hosts remotos cuando Ansible se ejecuta. Los módulos son los programas que realizan el trabajo real de las tareas de un play. Ansible es muy útil porque viene con cientos de módulos que realizan un trabajo administrativo útil del sistema.

Ansible fue escrito originalmente por **Michael DeHaan**, el creador de la aplicación de aprovisionamiento Cobbler. Ansible se ha adoptado ampliamente porque es fácil de usar para los administradores de sistemas. Los desarrolladores facilitan el uso de Ansible porque está construido en Python.

Ansible es compatible con las herramientas DevOps, como Vagrant y Jenkins.

DevOps es el acrónimo en ingles de Development Operations (Desarrollo y Operaciones)

Ansible no puede auditar los cambios realizados localmente por otros usuarios en un sistema. La siguiente lista proporciona algunos otros ejemplos de elementos que Ansible no puede realizar.

- ✓ Ansible puede agregar paquetes a una instalación, pero no realiza la **instalación mínima inicial del sistema**.
- ✓ Aunque Ansible puede modificar la configuración, no lo monitorea.
- ✓ Ansible no realiza un seguimiento de los cambios realizados en los archivos del sistema, ni realiza un seguimiento de qué usuario o proceso realizó dichos cambios.

WHAT DO YOU WANT TO AUTOMATE TODAY?

INFRASTRUCTURE	APPLICATIONS	NETWORKS
CONTAINERS	SECURITY + COMPLIANCE	CLOUD

SERVIDORES: Distintas versiones de Linux y Windows

CLOUD:

- ✓ AMAZON WEB SERVICES
- ✓ GOOGLE CLOUD PLATFORM
- ✓ OPENSTACK
- ✓ VMWARE

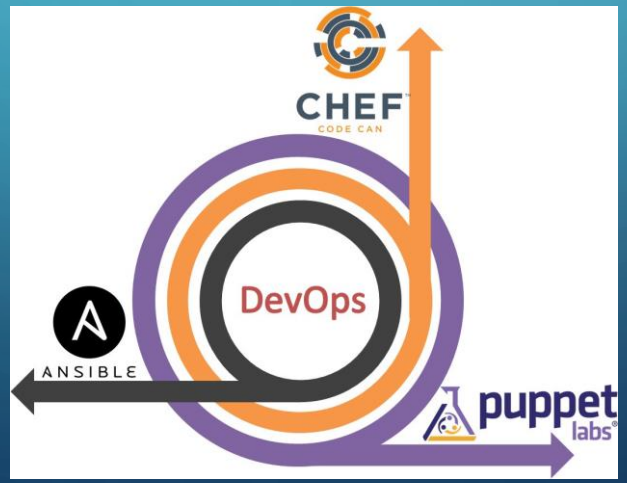
NETWORKS:

- ✓ SWITCHES
- ✓ ROUTERS

Puppet, Chef, Salt o Ansible

"Comparando herramientas de aprovisionamiento"

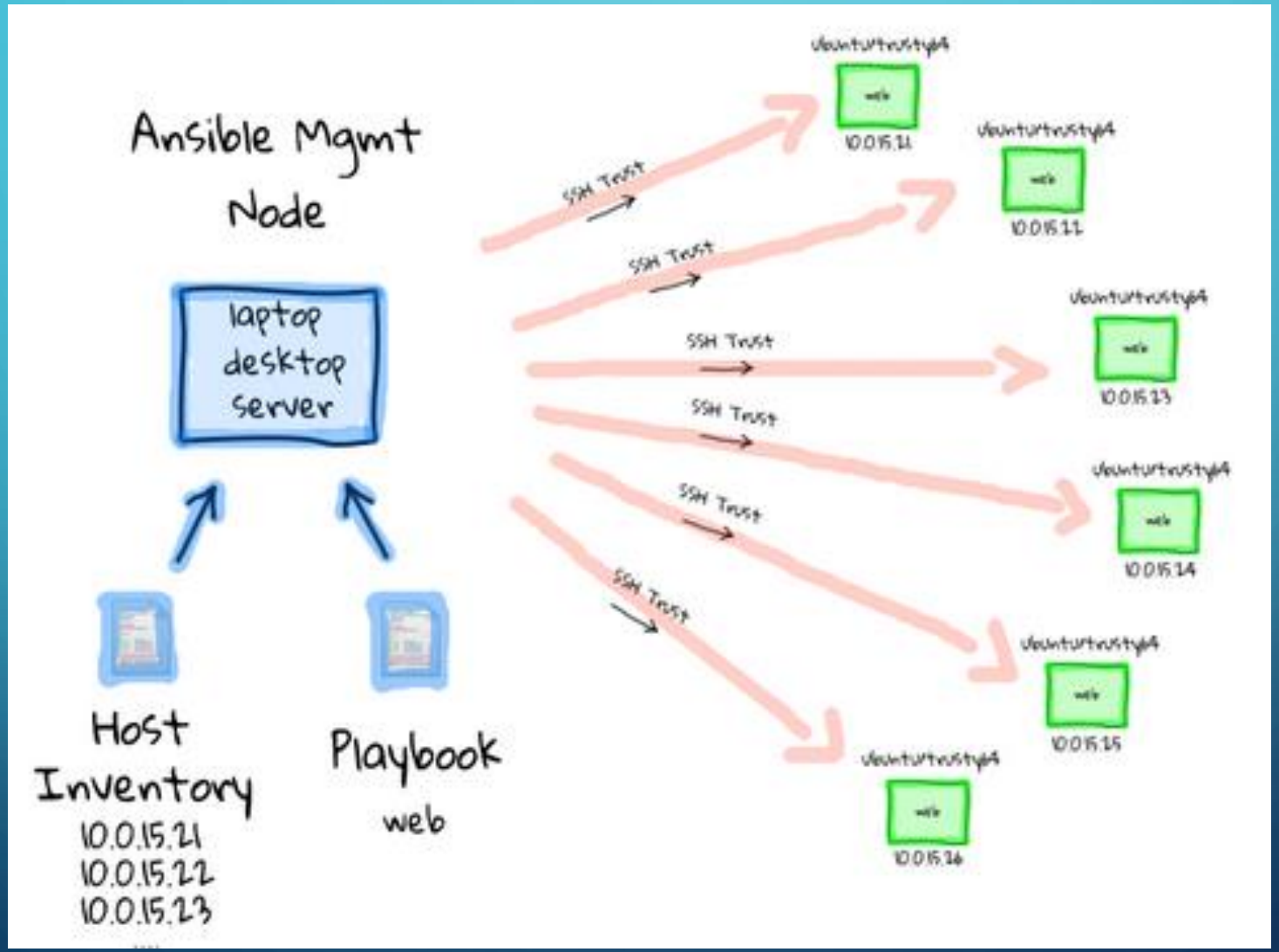
	Puppet	Chef	Ansible	Salt
Support	Puppet Labs	Opscode	Ansible Works	SaltStack
Control Interface	Manifest (DSL)	Recipes (DSL : Ruby 기반)	Playbook (YAML, JSON)	SLS (SoLt Sate/YAML)
Agent	Master/Agent Standalone	Server/Client Chef-solo (Standalone)	Standalone(Masterless)	Master/Agent(minions) Standalone(Masterless)
Core Technology	Ruby	Ruby	Python	Python
Communication	http, ssh / SSL	STOMP , rest / SSL	SSH / JSON	ZeroMQ
Remote Execution	Mcollective, challenging	Knife, challenging	Built-in, easy	Built-in, easy
Reference	Google, eBay, Twitter	Facebook, Ancestry, Splunk	Rackspace, Evernote	LinkedIn, HP Cloud
Since	2005	2009	2012	2011



2.2 CONCEPTOS Y ARQUITECTURA DE ANSIBLE

- Existen dos tipos de máquinas en la arquitectura de Ansible: el nodo de control y los hosts administrados.
- El software ansible está instalado en el nodo de control y todos sus componentes se mantienen en él. Los hosts administrados se enumeran en un inventario de host, un archivo de texto en el nodo de control que incluye una lista de nombres de host administrados FQDN (***fully qualified domain name)*** o direcciones IP.
- Ansible utiliza SSH como un transporte de red para comunicarse con los hosts administrados. Los módulos a los que se hace referencia en el playbook se copian en los hosts administrados. Luego se ejecutan, en orden, con los argumentos especificados en el playbook. Los usuarios de Ansible pueden escribir sus propios módulos personalizados, si es necesario, pero los módulos principales que vienen con Ansible pueden realizar la mayoría Tareas de administración del sistema.

ARQUITECTURA CLIENTE-SERVIDOR



2.3 MÉTODOS DE ORQUESTACIÓN ANSIBLE

Ansible se usa comúnmente para terminar de aprovisionar servidores de aplicaciones. Por ejemplo, se puede escribir un playbook para realizar los siguientes pasos en un sistema recién instalado:

- 1. Configurar repositorios de software.
- 2. Instalar la aplicación.
- 3. Ajustar los archivos de configuración. Opcionalmente descargando el contenido desde un sistema de control de versiones.
- 4. Abrir los puertos de servicio requeridos en el firewall.
- 5. Iniciar servicios relevantes.
- 6. Probar la aplicación y confirmar que está funcionando.

Orquestación de actualizaciones sucesivas de tiempo de inactividad cero

Ansible también es una herramienta sencilla para actualizar aplicaciones en paralelo. Por ejemplo, se puede desarrollar un playbook para ejecutar los siguientes pasos en los servidores de aplicaciones:

1. Detener el monitoreo del sistema y la aplicación.
2. Elimine el servidor del balanceo de carga.
3. Detener los servicios pertinentes.
4. Implementar, o actualizar, la aplicación.
5. Iniciar servicios relevantes.
6. Confirme que los servicios estén disponibles y vuelva a agregar el servidor al equilibrio de carga.
7. Iniciar el monitoreo del sistema y la aplicación.

2.4 INVENTARIOS DE ANSIBLE

Un inventario es la lista de hosts que va a gestionar Ansible. Los hosts pueden pertenecer a grupos que identifican la función de los hosts en la empresa. Un host puede ser miembro de más de un grupo.

Hay dos formas en que se pueden definir los inventarios de host. Un inventario de host estático se puede definir mediante un archivo de texto, o un inventario de host dinámico se puede generar desde proveedores externos.

```
[webservers]
localhost              ansible_connection=local
web1.example.com
web2.example.com:1234 ansible_connection=ssh ansible_user=ftaylor
192.168.3.7

[db-servers]
web1.example.com
db1.example.com
```

2.4.1 Inventario de host estático

Un inventario de host estático se define en un archivo de texto similar a INI, en el que cada sección define un grupo de hosts. **Cada sección comienza con un nombre de grupo de host incluido entre corchetes ([]).** Luego se listan las entradas para cada host administrado en el grupo, cada una en una sola línea.

La ubicación predeterminada para el archivo de inventario del host es /etc/ansible/hosts. Los comandos de ansible utilizarán un archivo de inventario de host diferente cuando se usen con la opción
-- inventory *PATHNAME*, -i *PATHNAME* **para abreviar.**

✓ Los inventarios de hosts ansibles pueden incluir grupos de grupos de hosts. Esto se logra con el sufijo **:children**

```
## db-[99:101]-node.example.com
localhost ansible_connection=local
[CENTOS]
172.20.20.29
[DEBIAN]
172.20.20.30

[LINUX:children]
DEBIAN
CENTOS

[LINUX:vars]
ansible_become=True
```

ansible_connection Tipo de conexión al host.
ansible_host El nombre del host al que se conectará, si es diferente del alias que desea darle.
ansible_port El número de puerto ssh, si no 22
ansible_user El nombre de usuario ssh predeterminado a usar.
ansible_ssh_pass La contraseña ssh a usar (nunca en texto plano, siempre use un vault. Vea Variables and Vaults)
ansible_become Equivalente a ansible_sudo o ansible_su, permite forzar la escalada de privilegios
ansible_become_method Permite establecer el método de escalado de privilegios.
ansible_become_user Equivalente a ansible_sudo_user o ansible_su_user, permite configurar el usuario en el que se convierte a través de la escalada de privilegios

✓Simplificación de inventarios de host con rangos.

Los inventarios de hosts ansibles se pueden simplificar especificando rangos en los nombres de host o direcciones IP. Se pueden especificar rangos numéricos, pero también se admiten rangos alfabéticos. Los rangos tienen la siguiente sintaxis:

```
[START:END]
```

Los rangos coinciden con todos los valores entre START y END, inclusive.

192.168.[4:7].[0:255]: Todas las direcciones IP en la red 192.168.4.0/22 (192.168.4.0 hasta 192.168.7.255).

server[01:20].example.com: Todos los hosts llamados server01.example.com a través de server20.example.com.

2.4.2 Inventario de host dinámico:

Las fuentes de información de inventario dinámico incluyen proveedores de nube pública/privada, información del sistema Cobbler, una base de datos LDAP o una base de datos de administración de configuración (CMDB).

Ansible incluye scripts que manejan información dinámica de host, grupo y variable de los proveedores más comunes, como Amazon EC2, Cobbler, Rackspace Cloud y OpenStack. Para los proveedores de la nube, la autenticación y la información de acceso se deben definir en los archivos a los que pueden acceder los scripts.

```
[cobbler]

# Set Cobbler's hostname or IP address
host = http://127.0.0.1/cobbler_api

# API calls to Cobbler can be slow. For this reason, we cache the results of an API
# call. Set this to the path you want cache files to be written to. Two files
# will be written to this directory:
#   - ansible-cobbler.cache
#   - ansible-cobbler.index

cache_path = /tmp

# The number of seconds a cache file is considered valid. After this many
# seconds, a new API call will be made, and the cache file will be updated.

cache_max_age = 900
```

3.- IMPLEMENTANDO ANSIBLE

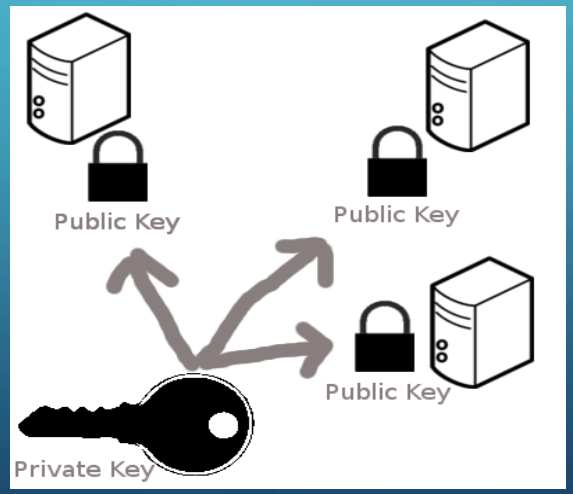
- ✓ A partir de la versión 2.4 de Ansible se agregó la compatibilidad con Pyton3.
- ✓ A partir del CentOS 7.4 ya no es necesario agregar el repositorio EPEL.

3.1 PREREQUISITOS DE ANSIBLE

- ✓ Para el nodo de control requiere la instalación de Python 2.6 o posterior.
- ✓ Para el nodo cliente requiere tener instalado Python 2.4 o posterior.

EXCEPTO los routers, ya que estos utilizan un modulo raw de ansible.

- ✓ Configurar la autentificación basada en clave SSH “SSH-KEYGEN” Esto crea una clave pública / privada que permite la conexión entre ambos servidores por SSH sin necesidad de introducir contraseña.



3.2 SINTAXIS

- Se debe indicar el inventario "opción -i" que defina la lista de hosts que debe administrar el nodo de control.

```
[student@controlnode ~]$ ansible <host-pattern> -i /path/to/inventory/file [options]
```

- Opción --list-hosts para identificar el host al que se hace referencia.

```
[student@controlnode ~]$ ansible 192.168.2.1 -i myinventory --list-hosts
  hosts (1):
    192.168.2.1
```

- Podemos utilizar grupos de hosts.

```
[student@controlnode ~]$ ansible lab -i myinventory --list-hosts
  hosts (2):
    labhost1.example.com
    labhost2.example.com
```

- Otra opción es utilizar los "*"

```
[student@controlnode ~]$ ansible '192.168.2.*' -i myinventory --list-hosts
  hosts (2):
    192.168.2.1
    192.168.2.2
```

3.3 CONFIGURANDO ANSIBLE

- Es posible cambiar la configuración desde el fichero */etc/ansible/ansible.cfg* que se define por defecto en la instalación.

Si existe un archivo ansible.cfg en el directorio donde se ejecuta el comando ansible, se usará en lugar del archivo global. Esto permite a los administradores crear una estructura de directorios donde los diferentes entornos o proyectos se alojan en directorios separados y cada directorio contiene un archivo de configuración personalizado, con un conjunto único de configuraciones

```
[student@workstation ~]$ touch /home/student/.ansible.cfg
[student@workstation ~]$ ansible --version
ansible 2.0.1.0
  config file = /home/student/.ansible.cfg
...output omitted...
```

- Si se define la variable **$ANSIBLE_CONFIG** con la variable de entorno, ansible usa el archivo de configuración que la variable indica.

3.3.1 Fichero de configuración de Ansible:

```
[student@controlnode ~]$ grep "^\[" /etc/ansible/ansible.cfg
[defaults]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[accelerate]
[selinux]
```

- ✓ **[defaults]** : La mayoría de configuraciones se agrupan aquí.
- ✓ **[privilege_escalation]:** Configuracion de como se ejecutan las operaciones que requieren escalado de privilegios.
- ✓ **[paramiko_connection], [ssh_connection] y [accelerate]** contienen configuraciones para optimizar las conexiones a los hosts administrados.
- ✓ **[selinux]:** Configuración de interacciones de SELinux

AJUSTES DE ANSIBLE:

Setting	Decription
inventory	Ubicación del archivo de inventario Ansible.
remote_user	La cuenta de usuario remoto utilizada para establecer conexiones a hosts administrados.
become	Habilita o inhabilita la escalada de privilegios para operaciones en hosts administrados.
become_method	Define el método de escalado de privilegios en hosts gestionados.
become_user	La cuenta de usuario para escalar privilegios en hosts gestionados.
become_ask_pass	Define si el escalado de privilegios en los hosts administrados debe solicitar una contraseña.

3.4 COMANDOS AD-HOC

- Es la ejecución de tareas bajo demanda en los host administrados.
- Es una introducción a funciones de Ansible más avanzadas, como módulos, tareas, playbooks.
- Para fines de configuración, se pueden usar para realizar cambios de forma rápida en muchos hosts administrados, como realizar cambios en los archivos de configuración y realizar tareas de administración de software.

```
ansible host-pattern -m module -a 'argument1 argument2' [-i inventory]
```

- Un módulo es una herramienta para realizar una tarea específica.

- El *module_name* definido en el fichero */etc/ansible/ansible.cfg*

```
# default module name for /usr/bin/ansible
#module_name = command
```

Aunque esté comentado, se utiliza como modulo por defecto.

```
ansible host-pattern -m command -a 'module arguments'
```

```
ansible host-pattern -a 'module arguments'
```

El módulo ***command*** no es procesado por el shell en los hosts administrados. No pueden acceder a las variables de entorno del shell o realizar operaciones de shell como la redirección y tuberias. Para esto usar el módulo de ***shell***

```
[student@demo ~]$ ansible localhost -m command -a set
localhost | FAILED | rc=2 >>
[Errno 2] No such file or directory
[student@demo ~]$ ansible localhost -m shell -a set
localhost | SUCCESS | rc=0 >>
BASH=/bin/sh
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interact
ive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
...output omitted...
```

3.5 MÓDULOS DE ANSIBLE

- Hay muchos módulos disponibles, el comando *ansible-doc -l* lista todos:

```
[ansible@ansible_server ~]$ ansible-doc  -l | grep shell
shell                                   Execute commands in nodes.
vmware_vm_shell                         Execute a process in VM
win_psmodule                            Adds or removes a Powershell Module.
win_shell                               Execute shell commands on target hosts.
[ansible@ansible_server ~]$
```

- Modulo copy

```
[ansible@ansible_server ~]$ ansible-doc  -l | grep copy
ce_file_copy                            Copy a file to a remote cloudengine device over SCP on HUAWEI CloudEngine switches.
copy                                    Copies files to remote locations
ec2_ami_copy                            copies AMI between AWS regions, return new image id
ec2_snapshot_copy                       copies an EC2 snapshot and returns the new Snapshot ID.
netapp_e_volume_copy                    Create volume copy pairs
nxos_file_copy                          Copy a file to a remote NXOS device over SCP.
unarchive                               Unpacks an archive after (optionally) copying it from the local machine.
vsphere_copy                            Copy a file to a vCenter datastore
win_copy                                Copies files to remote locations on windows hosts
win_robocopy                            Synchronizes the contents of two directories using Robocopy
```

- Hay muchos módulos disponibles en la documentación oficial:

https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html

4.- IMPLEMENTACIÓN DE PLAYBOOKS

- Los playbooks se escriben utilizando el lenguaje YAML.
- YAML fue diseñado principalmente para la representación de estructuras de datos en un formato fácil de leer y legible .
- Utiliza espacios para la sangría de contorno (identación).
- IMPORTANTE NO USAR TABULACIONES.
- Opcionalmente se usa --- para indicar el principio y … para indicar el final de un fichero YAML.
- Se ejecutan en orden secuencial.
- Deben asegurar **idempotencia** (capacidad para ejecutarse varias veces sin alterar ni hacer cambios en los sistemas)

```yaml
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:

  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache

  - name: ensure apache is running (and enable it at boot)
    service: >
      name=httpd
      state=started
      enabled=yes

  - block:
    - yum: name={{ item }} state=installed
      with_items:
      - httpd
      - memcached
    - template: src=templates/src.j2 dest=/etc/foo.conf
    - service: name=bar state=started enabled=True
    rescue:
    - debug: msg="Shit's on fire yo"
    always:
    - shell: echo "I'm always going to happen, I'm neat"
    when: ansible_distribution == 'CentOS'
    become: true
    become_user: root
  handlers:
    - name: restart apache
```

N playbook.yml unix | utf-8 | ansible 22% 8:1

"roles/playbook.yml" 36L, 849C

4.1 STRINGS

- No requieren comillas:

```
this is a string
```

```
'this is another string'
```

```
"this is yet another a string"
```

- Se puede utilizar "|" o ">" para escribir strings de varias líneas.

```
include_newlines: |
          Example Company
          123 Main Street
          Atlanta, GA 30303
```

```
fold_newlines: >
          This is
          a very long,
          long, long, long
          sentence.
```

4.2 DICCIONARIOS:

- Los pares de datos **clave /valor** en formato sangría son utilizados en YAML.

```
---
  name: Automation using Ansible
  code: DO407
```

4.3 LISTAS:

- Son como arrays en otros lenguajes de programación.
- Se usa con un solo guión seguido de un espacio o en formato de línea donde los elementos se encierran entre corchetes y se separan por comas y espacio.

```
---
 - red
 - green
 - blue
```

```
---
fruits:
  [red, green, blue]
```

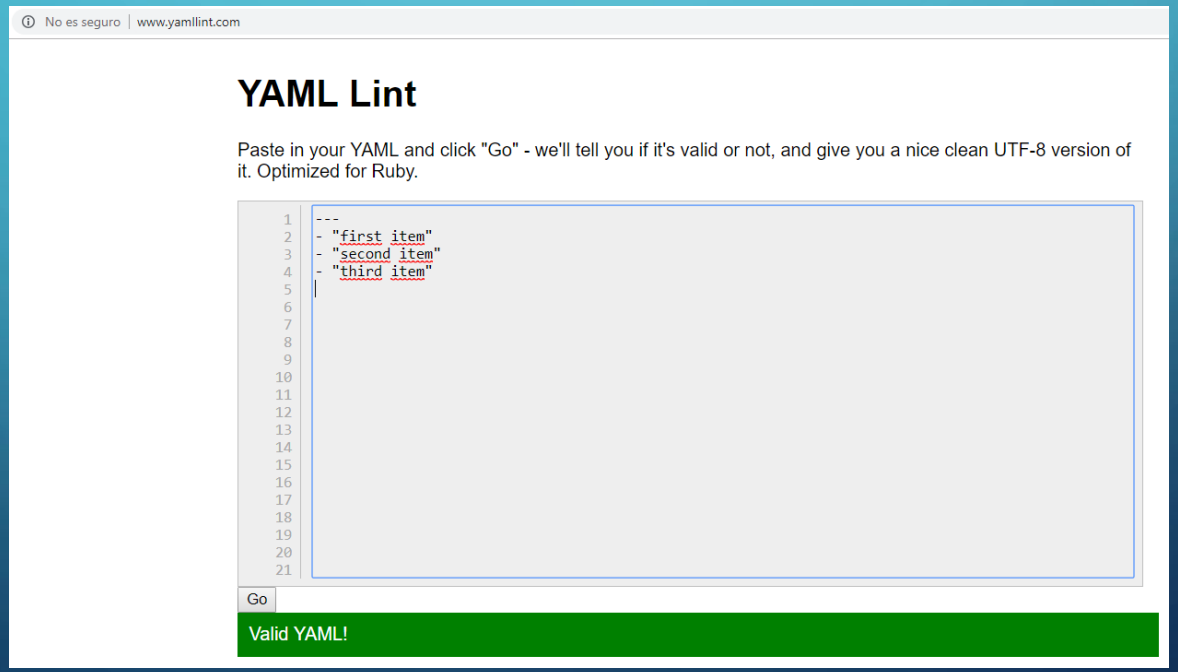
4.3 VERIFICANDO SINTAXIS YAML

- Es aconsejable que se verifique la sintaxis YAML de un playbook antes de su ejecución.

4.3.1 Con Python:

```
python -c 'import yaml, sys; print yaml.load(sys.stdin)' < myyaml.yml
```

4.3.2 Con YAML Lint: Herramienta de verificación de sintaxis online.



4.3.3 Con los comandos de ansible:

- Ansible ofrece una función nativa para validar la sintaxis de YAML en un playbook. El comando *ansible-playbook*, utilizado para ejecutar playbooks, incluye una opción *--syntax-check* que verifica errores de sintaxis.
- También se puede usar la opción *-C* o *--check* que simula la ejecución del playbook aunque realmente no realiza ningún cambio.
- Ansible también ofrece la opción *--step* de ejecutar paso a paso el playbook.

Para cada task solicita *(N) o/ (Y) es/(C)ontinue*

- Si queremos listar las tareas de un playbooks podemos utilizar *--list-tasks*
- Para empezar la ejecución desde una tarea: *--start-at-task="nombre"*
- Se puede aumentar la información de ejecución utilizando *-v (hasta 3 -vvv)*
- Hay muchas más disponibles en: *ansible-playbook --help*

4.4 IMPLEMENTANDO PLAYBOOKS

- Los comandos ad hoc solo pueden invocar un módulo y un conjunto de argumentos a la vez.
- Los playbooks pueden convertir tareas administrativas largas y complejas en rutinas fácilmente repetibles con resultados predecibles y exitosos.
- Los playbooks contienen play's, donde cada play sirve para definir un conjunto de operaciones a realizar en un conjunto específico de hosts administrados. Estas operaciones se denominan tasks. Las tasks se realizan invocando módulos Ansible y pasándoles los argumentos necesarios para realizar la operación deseada.

Importante!!!
El orden del contenido dentro de un playbook es importante, ya que Ansible ejecuta los plays y los tasks en el orden en que se presentan.

4.5 ESCRIBIENDO UN PLAYBOOK

- En un fichero de playbook, los play's se expresan en un contexto de lista donde cada play se define utilizando una estructura de datos del diccionario YAML.
- El guión en la primera entrada marca el comienzo de un play usando la sintaxis del elemento de la lista. La segunda entrada tiene espacio en el mismo nivel de sangría que la primera entrada para indicar que los dos atributos se refieren al mismo play principal.

```
---
- attribute1: value1
  attribute2: value2
```

- El atributo *name* es un atributo opcional, pero es una práctica recomendada nombrar las tareas con una etiqueta.

```
name: my first play
```

- El atributo de *hosts* debe definirse en cada juego.

```
hosts: managedhost.example.com
```

- Atributo de usuario:

```
remote_user: remoteuser
```

- Atributos de escalada de privilegios:

```
become: True/False
```

```
become_method: sudo
```

```
become_user: privileged_user
```

- Atributo de *task*: Es el componente principal de un play, contiene las operaciones que se ejecutarán en los hosts.

```
tasks:
- name: first task
  service: name=httpd enabled=true
```

4.5 FORMATO DE PLAYBOOKS

- Los argumentos largos del módulo se pueden dividir utilizando un formato multilínea y debe separarse con un espacio.

```
tasks:
- name: first task
  service: name=httpd enabled=true state=started
```

```
tasks:
- name: first task
  service: name=httpd
           enabled=true
           state=started
```

- Formato estructura del diccionario de YAML.

```
tasks:
- name: first task
  service:
    name: httpd
    enabled: true
    state: started
```

4.6 PATRONES EN PLAYBOOKS:

- Podemos limitar la ejecución de los playbooks a los hosts o grupos que queremos.
- Para todos los hosts podemos utilizar "*all*" o "*"
- Para limitar excluir grupos:

```
ansible-playbook --limit 'servweb:&madrid'
```

Ejemplo: Todos los host del grupo **webservers** que no estén en **Phoenix.**

```
webservers:!phoenix
```

- **También se puede utilizar la intersección de dos grupos:**

Ejemplo: Los hosts tiene que estar en ambos servidores.

```
webservers:&staging
```

- O combinarlos:

```
webservers:dbservers:&staging:!phoenix
```

5.- GESTIÓN DE VARIABLES E INCLUSIONES

- Las variables pueden utilizarse para almacenar valores que pueden reutilizarse en todos los archivos de todo un proyecto.
- Las variables deben comenzar con una letra y solo puede contener letras, números y guiones bajos.

Ansible variable names	
Invalid variable names	**Valid variable names**
`web server`	`web_server`
`remote.file`	`remote_file`
`1st file`	`file_1` or `file1`
`remoteserver$1`	`remote_server_1` or `remote_server1`

Las variables pueden estar dentro de los playbooks o fuera de ellos.

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

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

YAML format:

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

Cualquiera de las variables (Inventario o ficheros) se pueden anular redefiniendo la variable desde el comando.

```
[user@demo ~]$ ansible-playbook demo2.example.com main.yml -e "package=apache"
```

- Las variables que van en el inventario pueden ir para un host o para el grupo, en ese caso la variable de host tiene prioridad sobre las de grupo.

```
[grupo2]
192.168.0.32 package=ftp

[madrid:children]
grupo1
grupo2

[all:vars]
ansible_user=ansible
ansible_become=yes
package=httpd
```

- La práctica recomendada es definir variables de inventario utilizando los directorios host_vars y group_vars, y no definirlas en el inventario.

Ejemplo:

```
project
├── ansible.cfg
├── group_vars
│   ├── datacenters
│   ├── datacenters1
│   └── datacenters2
├── host_vars
│   ├── demo1.example.com
│   ├── demo2.example.com
│   ├── demo3.example.com
│   └── demo4.example.com
├── inventory
└── playbook.yml
```

- En el playbook, se hace referencia a la variable utilizando llaves dobles.
- Si la variable se usa como primer elemento para comenzar un valor, es obligatorio usar comillas dobles.

```
vars:
  user: joe

tasks:
  # This line will read: Creates the user joe
  - name: Creates the user {{ user }}
    user:
      # This line will create the user named Joe
      name: "{{ user }}"
```

```
yum:
     name: {{ service}}
            ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes.  Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}

Should be written as:

    with_items:
      - "{{ foo }}"
```

5.1 FACTS DE ANSIBLE

- Los facts son variables que Ansible descubre automáticamente de un host gestionado.
- **Por ejemplo:**
  - ✓ Se puede reiniciar un servidor dependiendo de la versión actual del kernel.
  - ✓ La configuración de MySQL se puede personalizar dependiendo de la memoria disponible.
  - ✓ Los usuarios pueden crearse dependiendo del nombre de host.
- Pueden formar parte de blocks, loops, conditionals, etc.

- Que datos podemos extraer:
  - ✓El nombre del host
  - ✓La versión del kernel.
  - ✓Las interfaces de red.
  - ✓Las direcciones IP
  - ✓La versión del sistema operativo.
  - ✓Variables de entorno.
  - ✓El número de CPUs.
  - ✓La memoria disponible o libre.
  - ✓El espacio en disco disponible

```
[user@demo ~]$ ansible demo1.example.com -m setup
ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/01/2011",
```

NOTA: La salida se muestra en **JSON** y cada valor se almacena en un diccionario de Python. Podemos navegar por los diccionarios para recuperar un valor particular.

- **Ansible Facts:**

Fact	*Variable*
Hostname	`{{ ansible_hostname }}`
Main IPv4 address (based on routing)	`{{ ansible_default_ipv4.address }}`
Main disk first partition size (based on disk name, such as **vda**, **vdb**, and so on.)	`{{ ansible_devices.vda.partitions.vda1.size }}`
DNS servers	`{{ ansible_dns.nameservers }}`
kernel version	`{{ ansible_kernel }}`

NOTA: Cuando se utiliza un fact en un playbook, Ansible sustituye dinámicamente el nombre de la variable con el valor correspondiente.

- **Filtros Facts:**

Los filtros sirven para limitar los resultados que se obtienen. Los filtros se pueden utilizar para agrupar datos:

- ✓Solo recuperar información de las tarjetas de red.
- ✓Solo recuperar información de los discos.
- ✓Solo recuperar información de los usuarios.

```
ansible madrid -m setup -a 'filter=*user*'
```

Opción *-a 'filter=EXPRESION'*.

```
[user@demo ~]$ ansible demo1.example.com -m setup -a 'filter=ansible_eth0'
demo1.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "ipv4": {
                "address": "172.25.250.10",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
```

Personalizar los FACTS:

- Es posible crear sus propios facts y enviarlos a un nodo administrado.
- Ansible encuentra los facts personalizados si está en el directorio /etc/ansible/facts.d de cada servidor cliente. Los fichero tienen extensión *.fact*. Un archivo fact es un archivo de texto plano en formato INI o JSON.

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = joe
user2 = jane
```

```
{
  "packages": {
    "web_package": "httpd",
    "db_package": "mariadb-server"
  },
  "users": {
    "user1": "joe",
    "user2": "jane"
  }
}
```

NOTA: Un archivo INI contiene una sección seguido de los pares clave-valor

Con el filtro *-a 'filter=ansible_local'* podemos ver los facts de cada servidor cliente.

```
[ansible@ansible_server30 ansible]$ ansible madrid -m setup -a 'filter=ansible_local'
192.168.0.31 | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "madrid": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                },
                "users": {
                    "user1": "joe",
                    "user2": "jane"
                }
            }
        }
    },
    "changed": false
}
192.168.0.32 | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "madrid": {
                "packages": {
                    "db_package": "mysql",
                    "web_package": "nginx"
                },
                "users": {
                    "user1": "kike",
                    "user2": "vane"
                }
            }
        }
    },
    "changed": false
}
```

```
[ansible@ansible_cliente31 facts.d]$ cat /etc/ansible/facts.d/madrid.fact
{
  "packages": {
    "web_package": "httpd",
    "db_package": "mariadb-server"
  },
  "users": {
    "user1": "joe",
    "user2": "jane"
  }
}
```

```
[ansible@ansible_cliente32 facts.d]$ cat /etc/ansible/facts.d/madrid.fact
[packages]
web_package = nginx
db_package = mysql

[users]
user1 = kike
user2 = vane
```

```
[ansible@ansible_server30 ansible]$ cat ejemplo14.yml
---
- hosts: madrid
  tasks:
    - name: Installs packages on {{ ansible_fqdn }}
      yum:
        name:
          - "{{ ansible_local.madrid.packages.db_package }}"
          - "{{ ansible_local.madrid.packages.web_package }}"
        state: latest
      register: result
    - name: Displays the result of the command
      debug:
        var: result
...
```

```
TASK [Displays the result of the command] ****************************
ok: [192.168.0.31] => {
    "result": {
        "changed": false,
        "failed": false,
        "msg": "",
        "rc": 0,
        "results": [
            "All packages providing mariadb-server are up to date",
            "All packages providing httpd are up to date",
            ""
        ]
    }
}
ok: [192.168.0.32] => {
    "result": {
        "changed": false,
        "failed": false,
        "msg": "",
        "rc": 0,
        "results": [
            "All packages providing mysql are up to date",
            "All packages providing nginx are up to date",
            ""
        ]
    }
}
```

5.2 INCLUSIONES

- Cuando se trabaja con playbooks muy complejos o largos se puede usar archivos para dividir tareas y listas de variables en partes más pequeñas.

```
tasks:
  - name: Include tasks to install the database server
    include: tasks/db_server.yml
```

- El módulo include_vars puede incluir variables definidas en archivos **JSON** o **YAML**, anulando variables de host y variables de playbook ya definidas.

```
tasks:
 - name: Include the variables from a YAML or JSON file
   include_vars: vars/variables.yml
```

• INCLUSIONES DE TASKS

- ✓ Para servidores nuevos se puede crear varios conjuntos de tareas, para crear usuarios, instalar paquetes, configurar servicios, configurar privilegios, configurar el acceso a un sistema de archivos compartido, fortalecer los servidores, instalar actualizaciones de seguridad e instalar un agente de monitoreo . Cada uno de estos conjuntos de tareas se podría administrar a través de un archivo de tareas independiente y separado.
- ✓ Si los desarrolladores, los administradores del sistema o de base de datos, administran los servidores de forma colectiva, cada organización puede escribir su propio conjunto de tareas que serán revisadas e integradas por el administrador del sistema.
- ✓ Si un servidor requiere una configuración particular, puede integrarse como un conjunto de tareas ejecutadas en función de un condicional (es decir, incluir las tareas solo si se cumplen criterios específicos).

- La directiva *include_tasks* se usa en el playbook para especificar qué fichero de tarea incluir en qué punto del playbook.
- Se puede usar una directiva *vars* para establecer las variables utilizadas por el fichero de tareas y anulará las variables del playbook, las variables de inventario, las variables registradas y los facts definidos en el fichero de tareas.

NOTA: A partir de la versión 2.4 de Ansible, se utiliza el modulo *include_tasks* y no solo *include*.

- Ejemplo: El fichero de tareas *environment.yml* ***(izq)*** se incluye en un playbook que tiene un *vars* con variables que se utilizan en el fichero de tareas:

```
- name: Installs the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest

- name: Starts the {{ service }} service
  service:
    name: "{{ service }}"
    state:  "{{ state }}"
```

```
---
- name: Install, start, and enable services
  hosts: all
  tasks:
  - name: Includes the tasks file and defines the variables
    include: environment.yml
    vars:
      package: mariadb-server
      service: mariadb
      state: started
    register: output

  - name: Debugs the included tasks
    debug:
      var: output
```

NOTA: Se puede crear un directorio tasks dedicado para archivos de tareas y guardar todos los archivos de tareas en ese directorio. Luego, en el playbook se indica el directorio *tasks/<<my_task.yml>>*

- **INCLUYENDO VARIABLES:**

Al igual que las tareas, las variables también pueden definirse externamente e incluirse en los los playbooks.

Algunas de las formas de establecer variables incluyen:

- Variables de inventario definidas en el archivo de inventario o en ficheros externos en los directorios host_vars y group_vars
- Facts y variables registradas.
- Variables de Playbook definidas en el archivo de playbook con *vars* o en un archivo externo a través de vars_files

- **Modulo include_vars**

El módulo *include_vars* es una forma más de establecer variables en un playbook desde un archivo externo.

Reemplaza cualquier valor establecido a través de los métodos anteriores.

Los archivos de variables incluidos se pueden definir utilizando JSON o YAML, siendo YAML la sintaxis preferida. El módulo include_vars toma la ruta del archivo de variables para importar como un argumento.

- Ejemplo: El variables.yml (izq) contiene 2 variables que indican que paquetes instalar:

```
---
packages:
  web_package: httpd
  db_package: mariadb-server
```

```
---
- name: Install web application packages
  hosts: all
  tasks:
  - name: Includes the tasks file and defines the variables
    include_vars: variables.yml

  - name: Debugs the variables imported
    debug:
      msg: >
        "{{ packages['web_package'] }} and {{ packages.db_package }}
        have been imported"
```

6.- TASK CONTROL

El control de tareas o task control permite realizar lo siguiente:

- Implementar loops o bucles en playbooks.
- Construir condicionales en playbooks.
- Combinar bucles y condicionales.

6.1 ANSIBLE LOOPS O BUCLES DE ANSIBLE

- Ansible admite muchos formatos de bucle para iterar sobre un conjunto de valores definidos en un array.
- El uso de bucles evita que se escriban tareas repetitivas que usan el mismo módulo; por ejemplo, si hay que crear varios usuarios o instalar varios paquetes, se puede definir un array y crear una tarea que se repita en el array en lugar de usar el módulo yum dos veces.
- Ansible soporta los siguientes bucles:
  - ✓ Bucles simples
  - ✓ Lista de hashes
  - ✓ Bucles anidados

BUCLES SIMPLES:

Un bucle simple es una lista de elementos que Ansible lee y repite. Se definen proporcionando una lista de elementos a la palabra clave *with_items*.

```
- yum:
    name: postfix
    state: latest

- yum:
    name: dovecot
    state: latest
```

=

```
- yum:
    name: "{{ item }}"
    state: latest
  with_items:
    - postfix
    - dovecot
```

```
vars:
  mail_services:
    - postfix
    - dovecot

tasks:
  - yum:
      name: "{{ item }}"
      state: latest
    with_items: "{{ mail_services }}"
```

LISTA DE HASHES:

Cuando se pasan arrays como argumentos, el array puede ser una lista de hashes. El siguiente fragmento de código muestra cómo se pasa un array multidimensional (un array con valores de par de claves) al módulo de usuario para personalizar tanto el nombre como el grupo:

```
- user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  with_items:
    - { name: 'jane', groups: 'wheel' }
    - { name: 'joe', groups: 'root' }
```

BUCLES ANIDADOS:

Los bucles anidados son bucles dentro de los bucles llamados con la palabra clave *with_nested*. Cuando se utilizan bucles anidados, Ansible se repite sobre el primer array siempre que haya valores en el. Por ejemplo, cuando se requieren múltiples privilegios de MySQL para varios usuarios, se puede crear un array multidimensional e invocarlo con la palabra clave *with_nested*. El siguiente fragmento de código muestra cómo usar el módulo mysql_user en un bucle anidado para otorgar a dos usuarios un conjunto de tres privilegios:

```
- mysql_user:
    name: "{{ item[0] }}"
    priv: "{{ item[1] }}.*:ALL"
    append_privs: yes
    password: redhat

  with_nested:
    - [ 'joe', 'jane' ]
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

COMBINANDO AMBOS BUCLES:

```
vars:
  users:
    - joe
    - jane

tasks:
  - mysql_user:
      name: "{{ item[0] }}"
      priv: "{{ item[1] }}.*:ALL"
      append_privs: yes
      password: redhat

    with_nested:
      - "{{ users }}"
      - [ 'clientdb', 'employeedb', 'providerdb' ]
```

Podemos utilizar *register* para capturar la salida de un módulo que usa un array.

```
- shell: echo "{{ item }}"
  with_items:
    - one
    - two
  register: echo
```

OTROS LOOPS:

Ansible Loops

Loop Keyword	Description
`with_file`	Takes a list of control node file names. **`item`** is set to the content of each file in sequence.
`with_fileglob`	Takes a file name globbing pattern. **`item`** is set to each file in a directory on the control node that matches that pattern, in sequence, non-recursively.
`with_sequence`	Generates a sequence of items in increasing numerical order. Can take **`start`** and **`end`** arguments which have a decimal, octal, or hexadecimal integer value.
`with_random_choices`	Takes a list. **`item`** is set to one of the list items at random.

6.2 CONDICIONALES

- ✓ Ansible puede ejecutar tareas o plays cuando se cumplan ciertas condiciones.
- ✓ Se puede definir un límite en una variable (por ejemplo, min_memory) y compararla con la memoria disponible en un host administrado.
- ✓ La salida de un comando puede ser capturada y evaluada por Ansible para determinar si una tarea se completa o no antes de tomar una acción adicional. Por ejemplo, si un programa falla, será necesario omitir un block.
- ✓ Los facts se pueden usar para determinar la configuración de la red y decidir qué archivo de plantilla enviar (por ejemplo, enlace de red o enlace troncal).
- ✓ El número de CPU's puede determinar cómo configurar un servidor web.
- ✓ Las variables registradas se pueden comparar con las variables definidas para verificar un cambio de servicio. Por ejemplo, esto se puede usar para verificar la suma de comprobación MD5 de un archivo.

Ansible conditionals operators

Operator	Example
Equal	`"{{ max_memory }} == 512"`
Less than	`"{{ min_memory }} < 128"`
Greater than	`"{{ min_memory }} > 256"`
Less than or equal to	`"{{ min_memory }} <= 256"`
Greater than or equal to	`"{{ min_memory }} >= 512"`
Not equal to	`"{{ min_memory }} != 512"`
Variable exists	`"{{ min_memory }} is defined"`
Variable does not exist	`"{{ min_memory }} is not defined"`
Variable is set to **1**, **True**, or **yes**	`"{{ available_memory }}"`
Variable is set to **0**, **False**, or **no**	`"not {{ available_memory }}"`
Value is present in a variable or an array	`"{{ users }} in users["db_admins"]"`

IMPORTANTE: El operador == utilizado para probar la igualdad en condiciones no debe confundirse con el operador = utilizado para asignar un valor a una variable.

DECLARACIÓN WHEN:

Para implementar un condicional en un elemento, se debe utilizar la instrucción *when*, seguida de la condición a probar. Cuando la declaración esté presente, Ansible la evaluará antes de ejecutar la tarea.

Ejemplo: Antes de crear el usuario db_admin, Ansible debe asegurarse de que el host administrado sea parte del grupo de bases de datos:

```
- name: Create the database admin
  user:
    name: db_admin
  when: inventory_hostname in groups["databases"]
```

IMPORTANTE: La instrucción *when* no es una variable del módulo, debe colocarse "fuera" del módulo mediante una sangría en el nivel superior de la tarea, tampoco puede estar por encima de la tarea.

CONDICIONES MULTIPLES:

Una declaración *when* se puede utilizar para evaluar múltiples valores, para ello se pueden combinar con *"and"* o *"or"* o agrupándolos entre paréntesis.

Con *"and"* ambas condiciones tienen que cumplirse para que se cumpla toda la declaración condicional.

Por ejemplo, se cumplirá la siguiente condición si el kernel instalado es la versión especificada y si el inventory_hostname está en el grupo staging:

```
ansible_kernel == 3.10.0-327.el7.x86_64 and inventory_hostname in groups['staging']
```

Con "or" cualquiera de las condiciones puede ser verdadera.

Por ejemplo, se cumple la siguiente condición si el host remoto es Red Hat Enterprise Linux o Fedora:

```
ansible_distribution == "RedHat" or ansible_distribution == "Fedora"
```

Las declaraciones condicionales más complejas se pueden expresar claramente agrupando las condiciones con paréntesis. Por ejemplo, se cumple la siguiente declaración condicional si el host es un Red Hat Enterprise Linux 7 o Fedora 23:

```
(ansible_distribution == "RedHat" and ansible_distribution_major_version == 7) or
  (ansible_distribution == "Fedora" and ansible_distribution_major_version == 23)
```

Se pueden combinar bucles y condicionales. En el siguiente ejemplo, yum instalará el mariadb-server si hay un sistema de archivos montado en / con más de 300 MB libres. El fact ansible_mounts es una lista de diccionarios, cada uno de los cuales representa facts sobre un sistema de archivos montado. El bucle recorre cada la lista, y la declaración condicional no se cumple a menos que se encuentre un diccionario que represente un sistema de archivos montado donde ambas condiciones son verdaderas.

```
- name: install mariadb-server if enough space on root
  yum:
    name: mariadb-server
    state: latest
  with_items: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 300000000
```

Aquí hay otro ejemplo que combina condicionales y variables registradas. El siguiente play reiniciará el servicio httpd solo si el servicio postfix se está ejecutando.

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix (1)
      ignore_errors: yes (2)
      register: result (3)

    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0 (4)
```

(1) ¿Se está ejecutando Postfix o no?
(2) Si no se está ejecutando y el comando "falla", no detenga el procesamiento
(3) Guarda información en el módulo **register** en una variable llamada resultado.
(4) Evalúa la salida de la tarea Postfix. Si el código de salida systemctl es 0, entonces Postfix está activo y esta tarea reiniciará el servicio httpd.

USANDO BOOLEANOS:

Los booleanos son variables que toman uno de dos valores posibles:

- **True** or **Yes** or **1**
- **False** or **No** or **0**

Los booleanos se pueden usar como un simple SWICHT para habilitar o deshabilitar tareas.

```
---
- hosts: all
  vars:
    my_service: True

  tasks:
    - name: Installs a package
      yum:
        name: httpd
      when: my_service
```

```
---
- hosts: all
  vars:
    my_service: False

  tasks:
    - name: Installs a package
      yum:
        name: httpd
      when: my_service
```

6.2 HANDLERS

- ✓ Los handlers son tareas que responden a una notificación enviada por otras tareas.
- ✓ Cada controlador tiene un nombre único a nivel mundial, y se activa al final de un bloque de tareas en un playbook.
- ✓ Los handlers pueden verse como tareas inactivas que solo se activan cuando se invocan explícitamente mediante una declaración de notificación.
- ✓ Si una o más tareas notifican al handler, se ejecutará solo una vez después de que se hayan completado las demás tareas en el play.
- ✓ Normalmente los handlers se utilizan para reiniciar hosts y reiniciar servicios.

- El siguiente ejemplo muestra cómo el servidor Apache solo se reinicia mediante la tarea restart_apache cuando se actualiza un archivo de configuración y lo notifica:

```
tasks:

  - name: copy demo.example.conf configuration template (1)
    copy:
      src: /var/lib/templates/demo.example.conf.template
      dest: /etc/httpd/conf.d/demo.example.conf

    notify: (2)

      - restart_apache (3)


handlers: (4)

  - name: restart_apache (5)

    service: (6)
      name: httpd
      state: restarted
```

1. La tarea que notifica al handler.
2. Notifica que la tarea debe activar un handler.
3. El nombre del handler a ejecutar.
4. La sentencia inicia la sección de handlers.
5. El nombre del handler invocado por las tareas.
6. El módulo a utilizar para el handler.

6.3 TAGS O ETIQUETAS

Para playbooks largos, es útil poder ejecutar subconjuntos de las tareas. Etiquetar un solo recurso requiere que se use la palabra clave *tags*, seguido de una lista de etiquetas para aplicar. Cuando los plays están etiquetados, la opción *--tags* se puede usar con ansible-playbook para filtrar para ejecutar solo plays etiquetados. Las etiquetas están disponibles para los siguientes recursos:

En los playbooks:

```
tasks:
  - name: Install {{ item }} package
    yum: name={{ item }} state=installed
    with_items:
      - postfix
      - mariadb-server
    tags:
      - packages
```

Con include o include_tasks

```
- include: common.yml
  tags: [webproxy, webserver]
```

Se puede utilizar el argumento *--tags* para ejecutar los recursos etiquetados o también el *--skip-tags* para evitar que se ejecuten en el playbook.

```
---
- hosts: all
  tasks:
    - name: My tagged task
      yum:
        name: httpd
        state: latest
      tags: production

    - name: Installs postfix
      yum:
        name: postfix
        state: latest
```

```
[user@demo ~]$ ansible-playbook main.yml --tags 'production'
TASK [My tagged task] ***********************************************************
ok: [demo.example.com]
... Output omitted ...
```

```
[user@demo ~]$ ansible-playbook main.yml --skip-tags 'production'
... Output omitted ...

TASK [Installs postfix] *********************************************************
ok: [demo.example.com]
```

6.4 ERRORES

- Ansible evalúa los códigos de retorno de los módulos y comandos para determinar si una tarea tuvo éxito o no.
- **Ignorar la tarea fallida:** de forma predeterminada, si una tarea falla, la reproducción se cancela; sin embargo, este comportamiento se puede anular omitiendo las tareas fallidas. Para hacerlo, la palabra clave *ignore_errors*.

  Por ejemplo, si el paquete notapkg no existe, el módulo *yum* fallará, pero al tener *ignore_errors* configurado en *yes* permitirá que la ejecución continúe.

```
- yum:
    name: notapkg
    state: latest
  ignore_errors: yes
```

- **Forzar la ejecución de los handlers:** de manera predeterminada, si una tarea que notifica un handler falla, el handler también será omitido. Se puede anular este comportamiento utilizando la palabra clave *force_handlers* en la tarea. Esto obliga a que se llame al handler incluso si la tarea falla.

```
---
- hosts: all
  force_handlers: yes
  tasks:
    - yum:
        name: notapkg
        state: latest
      notify: restart_database

  handlers:
    - name: restart_database
      service:
        name: mariadb
        state: restarted
```

- **Anular el estado fallido:** Una tarea en sí misma puede tener éxito, sin embargo, se puede querer marcar la tarea como fallida en función de criterios específicos. Para hacerlo, la palabra clave *failed_when* puede usarse con una tarea. Esto se usa a menudo con módulos que ejecutan comandos remotos y capturan la salida en una variable. Por ejemplo, se puede ejecutar un script que emita un mensaje de error y usar ese mensaje para definir el estado fallido de la tarea.

```
tasks:
  - shell:
      cmd: /usr/local/bin/create_users.sh
    register: command_result
    failed_when: "'Password missing' in command_result.stdout"
```

- **Anular el cambio de estado:** Cuando una tarea actualiza un host, adquiere el estado modificado; sin embargo, si una tarea no realiza ningún cambio en el nodo administrado, los handlers se omiten. La palabra clave *changed_when* se puede usar para anular el estado modificado. Por ejemplo, si se desea reiniciar un servicio cada vez que se ejecuta un playbook, se puede agregar la palabra clave *changes_when* a la tarea.

```
tasks:
  - shell:
      cmd: /usr/local/bin/upgrade-database
    register: command_result
    changed_when: "'Success' in command_result.stdout"
    notify:
      - restart_database

handlers:
  - name: restart_database
     service:
       name: mariadb
       state: restarted
```

Bloques ansibles y manejo de errores.

En los playbooks, los bloques son cláusulas que encierran tareas. Los bloques permiten la agrupación lógica de tareas y controlar cómo se ejecutan. Por ejemplo, definir un conjunto principal de tareas y un conjunto de tareas adicionales que solo se ejecutarán si falla el primer conjunto.

- ✓ **block :** Define las principales tareas a ejecutar.
- ✓ **rescue :** Define las tareas que se ejecutarán si las tareas de block fallan.
- ✓ **always :** Define las tareas que siempre se ejecutarán independientemente del éxito o el fracaso de las tareas block o rescue.

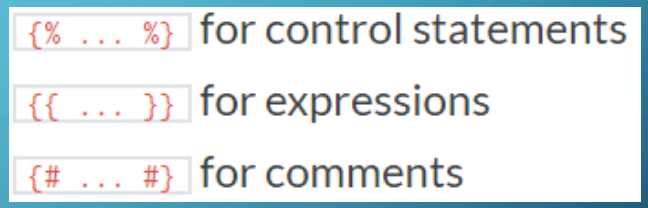
```
tasks:
  - block:
      - shell:
          cmd: /usr/local/lib/upgrade-database

    rescue:
      - shell:
          cmd: /usr/local/lib/create-users

    always:
      - service:
          name: mariadb
          state: restarted
```

7.- PLANTILLAS JINJA2.

- Ansible utiliza el sistema de plantillas Jinja2 para modificar los archivos antes de que se distribuyan a los hosts administrados.
- Están basadas en Python.
- Ansible usa plantillas de Jinja2 para habilitar expresiones dinámicas y acceso a variables.
- Ansible amplía enormemente la cantidad de filtros y pruebas disponibles, además incorpora un nuevo tipo de complemento: lookups
- Las plantillas sólo se alojan en el servidor controlador.

DELIMITADORES:

Las variables o expresiones lógicas se colocan entre etiquetas, o delimitadores. Por ejemplo, las plantillas Jinja2 usan {% EXPR%} para expresiones (por ejemplo, bucles), mientras que {{EXPR}} se usan para enviar los resultados de una expresión o una variable al usuario final. La última etiqueta, cuando se procesa, se reemplaza con un valor o valores, y los ve el usuario final. Utilice la sintaxis {# COMMENT #} para incluir comentarios.



En el siguiente ejemplo, la primera línea incluye un comentario que no se incluirá en el archivo final. Las referencias de variables en la segunda línea se reemplazan con los valores de los facts del sistema a los que se hace referencia.

```
{# /etc/hosts line #}
{{ ansible_default_ipv4.address }}    {{ ansible_hostname }}
```

ESTRUCTURAS DE CONTROL:

A menudo, el resultado de un play depende del valor de una variable, un fact o el resultado de una tarea anterior. En algunos casos, los valores de las variables dependen de otras variables. Además, se pueden crear grupos adicionales para los hosts administrados en función de si los hosts coinciden con otros criterios.

BUCLES:

Jinja2 usa la instrucción for para proporcionar funcionalidad de bucle.

La siguiente instrucción for se ejecuta a través de todos los valores en la variable de users, reemplazando myuser con cada valor, excepto cuando el valor es Snoopy

```
{# for statement #}
{% for myuser in users if not myuser == "Snoopy"%}
{{loop.index}} - {{ myuser }}
{% endfor %}
```

La variable loop.index incrementa al número de índice en el que se encuentra actualmente el bucle. Tiene un valor de 1 la primera vez que se ejecuta el bucle, y se incrementa en 1 en cada iteración.

Condicionales

Jinja2 usa la sentencia *if* para control condicional. En el siguiente ejemplo, el resultado se muestra si la variable finished es true:

```
{% if finished %}
    {{ result }}
{% endif %}
```

FILTROS EN VARIABLES:

Jinja2 proporciona filtros que cambian el formato de salida de las expresiones de plantilla (por ejemplo, a JSON). Hay filtros disponibles para lenguajes como YAML o JSON. El filtro *to_json* formatea la salida de expresión usando JSON, y el filtro *to_yaml* usa YAML como la sintaxis de formato.

```
{{ output | to_json }}
{{ output | to_yaml }}
```

NOTA: Ocasionalmente son útiles para la depuración de código.

Hay filtros adicionales disponibles, como los filtros *to_nice_json* y *to_nice_yaml*, que dan formato a la salida de la expresión en formato legible para humanos JSON o YAML.

```
{{ output | to_nice_json }}
{{ output | to_nice_yaml }}
```

Los filtros *from_json* y *from_yaml* esperan una cadena en formato JSON o YAML y la analizan.

```
{{ output | from_json }}
{{ output | from_yaml }}
```

Ejemplo:

```
tasks:
  - shell: cat /some/path/to/file.json
    register: result

  - set_fact:
      myvar: "{{ result.stdout | from_json }}"
```

Jinja2 proporciona un filtro util "default“ que establece valor por defecto, en caso de ser vacías o inexistentes.

```
{{ lookup('env', 'MY_USER') | default('admin', true) }}
```

Parámetros de omisión:

Es posible usar el filtro *default* para omitir los parámetros del módulo usando la variable especial *omit*:

```
- name: touch files with an optional mode
  file: dest={{ item.path }} state=touch mode={{ item.mode | default(omit) }}
  loop:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
      mode: "0444"
```

Para los dos primeros archivos de la lista, el modo predeterminado estará determinado por la umask del sistema, ya que el parámetro mode = no se enviará al módulo del archivo, mientras que el archivo final recibirá la opción mode = 0444.

Las expresiones utilizadas con la cláusula *when* en los playbooks de Ansible son expresiones Jinja2. Los filtros de Ansible que se utilizan para probar los valores de retorno incluyen failed, changed, succeeded y skipped.

```
tasks:
... Output omitted ...
  - debug: msg="the execution was aborted"
    when: returnvalue | failed
```

YAML vs. Jinja2

El uso de algunas expresiones Jinja2 dentro de un playbook YAML puede cambiar el significado de esas expresiones, por lo que requieren algunos ajustes en la sintaxis utilizada.

1. La sintaxis de YAML requiere comillas cuando un valor comienza con una referencia de variable ({{}}). Las comillas impiden que el analizador trate la expresión como el inicio de un diccionario YAML. Por ejemplo, el siguiente fragmento de un playbook fallará:

```
- hosts: app_servers
  vars:
    app_path: {{ base_path }}/bin
```

En su lugar, utilice la siguiente sintaxis:

```
- hosts: app_servers
  vars:
    app_path: "{{ base_path }}/bin"
```

2. Cuando exista la necesidad de incluir elementos {{...}} anidados, deben eliminarse las llaves alrededor de los internos.

```
    - name: display the host value
      debug:
        msg: hostname = {{ params[ {{ host_ip }} ] }}, IPaddr = {{ host_ip }}
```

```
... Output omitted ...
TASK [display the host value] ***************************************************
fatal: [localhost]: FAILED! => {"failed": true, "msg": "template error
 while templating string: expected token ':', got '}'. String: hostname =
 {{ params[ {{ host_ip }} ] }}, IPaddr = {{ host_ip }}"}
... Output omitted ...
```

Utilice la siguiente sintaxis en su lugar. Se ejecutará sin error.

```
    - name: display the host value
      debug:
        msg: hostname = {{ params[ host_ip ] }}, IPaddr = {{ host_ip }}
```

7.1 IMPLEMENTANDO PLANTILLAS JINJA2

Una plantilla Jinja2 se compone de: datos, variables y expresiones. Esas variables y expresiones se reemplazan cuando se representa la plantilla Jinja2.

Las variables utilizadas en la plantilla se pueden especificar en la sección vars del playbook. El siguiente ejemplo muestra cómo crear una plantilla con variables usando los facts: ansible_hostname y ansible_date_time.date. Cuando se ejecuta el playbook, esos dos facts se reemplazarán por sus valores en el host administrado:

```
Welcome to {{ ansible_hostname }}.
Today's date is: {{ ansible_date_time.date }}.
```

El siguiente ejemplo utiliza la instrucción *for*, y asume que se ha definido una variable myhosts en el archivo de inventario que se está utilizando. Esta variable contendría una lista de hosts para ser gestionados.

Con el siguiente *for*, se enumerarán todos los hosts en el grupo myhosts del inventario.

```
{% for myhost in groups['myhosts'] %}
{{ myhost }}
{% endfor %}
```

Es recomendable incluir al comienzo de una plantilla Jinja2 la variable `{{ ansible_managed }}`, para reflejar que el archivo es administrado por Ansible. Esto incluye en el host administrado un string de la fecha, ID de usuario y el hostname.

```
ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on
 {host}
```

USANDO PLANTILLAS JINJA2 EN PLAYBOOKS:

Las plantillas Jinja2 son una herramienta poderosa para personalizar los archivos de configuración que se implementarán en los hosts administrados. Cuando se ha creado la plantilla Jinja2 para un archivo de configuración, se puede implementar en los hosts administrados mediante el módulo de plantillas, que admite la transferencia de un archivo local en el nodo de control a los hosts administrados.

El valor asociado con la clave src especifica la plantilla de origen Jinja2, y el valor asociado a la clave dest especifica el archivo que se creará en los hosts de destino.

```
tasks:

    - name: template render
      template:
        src: /tmp/j2-template.j2
        dest: /tmp/dest-config-file.txt
```

8. ROLES:

Los datacenters tienen una variedad de diferentes tipos de hosts. Algunos funcionan como servidores web, otros como servidores de bases de datos y otros pueden tener instaladas y configuradas herramientas de desarrollo de software. Un playbook con tareas y handlers para manejar todos estos casos, se volvería grande y complejo con el tiempo. Los roles permiten organizar los playbooks y separarlos en ficheros más pequeños.

Los roles proporcionan a Ansible una forma de utilizar tareas, handlers y variables desde archivos externos. Los archivos estáticos y las plantillas también se pueden asociar y hacer referencia mediante un rol.

BENEFICIOS DE UTILIZARLO:

- ✓ Roles con contenido del grupo, lo que permite compartir fácilmente el código con otros.
- ✓ Se pueden escribir roles que definen los elementos esenciales de un tipo de sistema: servidor web, servidor de base de datos, repositorio de git u otro propósito
- ✓ Los roles hacen que los proyectos más grandes sean más manejables.
- ✓ Los roles pueden ser desarrollados en paralelo por diferentes administradores.

Estructura del ROL:

La funcionalidad de un rol ansible se define por su estructura de directorios. El directorio de nivel superior define el nombre de la función en sí. Algunos de los subdirectorios contienen archivos YAML, llamados **main.yml**. Los subdirectorios de ficheros y plantillas pueden contener objetos a los que hacen referencia los ficheros YAML.

```
[user@host roles]$ tree user.example
user.example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Ansible role subdirectories

Subdirectory	Function
`defaults`	The `main.yml` file in this directory contains the default values of role variables that can be overwritten when the role is used.
`files`	This directory contains static files that are referenced by role tasks.
`handlers`	The `main.yml` file in this directory contains the role's handler definitions.
`meta`	The `main.yml` file in this directory contains information about the role, including author, license, platforms, and optional role dependencies.
`tasks`	The `main.yml` file in this directory contains the role's task definitions.
`templates`	This directory contains Jinja2 templates that are referenced by role tasks.
`tests`	This directory can contain an inventory and `test.yml` playbook that can be used to test the role.
`vars`	The `main.yml` file in this directory defines the role's variable values.

VARIABLES Y VALORES POR DEFECTO:

Las variables de rol se definen creando un fichero vars/main.yml con pares clave: valor. Se mencionan en el archivo YAML de rol como cualquier otra variable: {{ VAR_NAME }}. Estas variables son de prioridad alta y no pueden ser anuladas por las variables de inventario.

Las variables *default* permiten establecer valores predeterminados. Se definen creando un archivo defaults/main.yml con pares clave: valor. Las variables predeterminadas tienen la prioridad más baja de todas las variables disponibles. Se pueden anular fácilmente con cualquier otra variable, incluidas las variables de inventario.

Defina una variable específica en vars/main.yml o defaults/main.yml, pero no en ambos lugares. Las variables predeterminadas se deben usar cuando se pretende que sus valores se anulen.

ROLES EN UN PLAYBOOK:

Para cada rol, las tareas de rol, los handlers de rol, las variables de rol y las dependencias de rol se incluirán en el playbook, en ese orden. Cualquier tarea de copia, script, plantilla o include en el rol pueden hacer referencia a los ficheros, plantillas o tareas relevantes sin nombres de ruta absolutos o relativos.

DEPENDENCIAS DE ROLES:

Las dependencias de roles permiten que un rol incluya otros roles como dependencias en un playbook. Por ejemplo, un rol que define un servidor de documentación puede depender de otro rol que instale y configure un servidor web. Las dependencias se definen en el archivo meta/main.yml.

```
---
dependencies:
  - { role: apache, port: 8080 }
  - { role: postgres, dbname: serverlist, admin_user: felix }
```

ORDEN DE EJECUCIÓN

Normalmente, las tareas de los roles se ejecutan antes que las tareas de los playbooks que los utilizan. Ansible proporciona una manera de anular este comportamiento predeterminado: las tareas pre_tasks y post_tasks.

Las tareas pre_tasks se realizan antes de que se apliquen los roles. Las tareas post_tasks se realizan después de que todos los roles se hayan completado.

```
---
- hosts: remote.example.com
  pre_tasks:
    - debug:
        msg: 'hello'
  roles:
    - role1
    - role2
  tasks:
    - debug:
        msg: 'still busy'
  post_tasks:
    - debug:
        msg: 'goodbye'
```

8.1 CREANDO ROLES

Crear y usar un rol es un proceso de tres pasos:

1. Crea la estructura del directorio de roles.
2. Definir el contenido del rol.
3. Usa el rol en un playbook.

Creando la estructura del directorio de roles

Ansible busca roles en un subdirectorio llamado roles en el directorio del proyecto. Los roles también se pueden mantener en los directorios a los que hace referencia la variable *roles_path* en los archivos de configuración de Ansible. Esta variable contiene una lista de directorios separados por dos puntos para buscar.

Cada rol tiene su propio directorio con subdirectorios. La siguiente estructura de directorio contiene los archivos que definen el rol de motd.

```
[user@host ~]$ tree roles/
roles/
└── motd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    ├── tasks
    │   └── main.yml
    └── templates
        └── motd.j2
```

El subdirectorio contiene archivos de contenido fijo y el subdirectorio de templates contiene plantillas que el rol puede implementar cuando se usa. Los otros subdirectorios pueden contener archivos main.yml que definen valores predeterminados, handlers, tareas, metadatos de rol o variables, según el subdirectorio en el que se encuentren. Si existe un subdirectorio pero está vacío, como los handlers en el ejemplo anterior, se ignora. Si un rol no utiliza una característica, el subdirectorio se puede omitir por completo, por ejemplo, los subdirectorios meta y vars en el ejemplo anterior.

Definición del contenido del rol.

Después de crear la estructura de directorios, se debe definir el contenido del rol. ROLENAME/tasks/main.yml. Este archivo define a qué módulos llamar en los hosts administrados a los que se aplica este rol.

El fichero tasks/main.yml gestiona el fichero /etc/motd en los hosts administrados. Utiliza el módulo de *template* para copiar la plantilla denominada motd.j2 en el host administrado. La plantilla se encuentra en el subdirectorio *templates* del rol.

```
[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd

- name: deliver motd file
  template:
    src: templates/motd.j2
    dest: /etc/motd
    owner: root
    group: root
    mode: 0444
```

El siguiente comando muestra el contenido de la plantilla templates/motd.j2 del rol motd. Hace referencia a los facts y una variable system-owner.

```
[user@host ~]$ cat roles/motd/templates/motd.j2
This is the system {{ ansible_hostname }}.

Today's date is: {{ ansible_date_time.date }}.

Only use this system with permission.
You can ask {{ system_owner }} for access.
```

El rol puede definir un valor predeterminado para la variable system_owner. El fichero defaults/main.yml es donde se establecen estos valores.

El siguiente fichero defaults/main.yml establece la variable system_owner en **usuario@host.ejemplo.com**. Esta será la dirección de correo electrónico que está escrita en el archivo /etc/motd de los hosts a los que se aplica este rol.

```
[user@host ~]$ cat roles/motd/defaults/main.yml
---
system_owner: user@host.example.com
```

ROLES EN UN PLAYBOOK:

El siguiente playbook se refiere al rol de motd. Debido a que no se especifican variables, el rol se aplicará con sus valores de variable predeterminados.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true

  roles:
    - motd
```

Cuando se ejecuta el playbook, las tareas realizadas de un rol pueden ser identificadas por el prefijo del nombre del rol.

```
[user@host ~]$ ansible-playbook -i inventory use-motd-role.yml

PLAY [use motd role playbook] **************************************************

TASK [setup] *******************************************************************
ok: [remote.example.com]

TASK [motd : deliver motd file] ************************************************
changed: [remote.example.com]

PLAY RECAP *********************************************************************
remote.example.com         : ok=2    changed=1    unreachable=0    failed=0
```

Cambiar el comportamiento de un rol con variables.

Las variables se pueden usar con roles, como parámetros, para anular los valores predeterminados definidos previamente.

Cuando se referencian, los pares variable: valor también deben especificarse.

El siguiente ejemplo muestra cómo usar la función motd con un valor diferente para la variable de función system_owner. El valor especificado, someone@host.example.com, reemplazará la referencia de variable cuando el rol se aplique a un host administrado.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true

  roles:
    - role: motd
      system_owner: someone@host.example.com
```

8.2 Implementación de roles con Galaxy ansible

Galaxia ansible:

Ansible Galaxy *https://galaxy.ansible.com* es una biblioteca pública de roles de Ansible escritos por una variedad de administradores y usuarios de Ansible. Es un archivo que contiene miles de roles de Ansible y tiene una base de datos de búsqueda que ayuda a los usuarios a identificar los roles que podrían ayudarles a realizar una tarea administrativa. Ansible Galaxy incluye enlaces a documentación y videos para nuevos usuarios de Ansible y desarrolladores de roles.

Consejos para usar Ansible Galaxy

Hay contenido que describe cómo descargar y usar roles de Ansible Galaxy. Las instrucciones sobre cómo desarrollar roles y subirlos a Ansible Galaxy también se encuentran en esa página.

Hay contadores asociados con cada rol que se encuentra en el sitio web de Ansible Galaxy. Los usuarios pueden votar por la utilidad de un rol haciendo clic en el botón de estrella. El número de estrellas que tiene un rol es una pista de su popularidad entre la comunidad de Ansible. Los usuarios también pueden ver un papel. El número de observadores para un rol da una indicación del interés de la comunidad en un rol en desarrollo

Línea de comandos ansible-galaxy.

La herramienta de línea de comandos ansible-galaxy se puede utilizar para buscar, mostrar información sobre, instalar, listar, eliminar o inicializar roles.

El comando *ansible-galaxy search* busca en Ansible Galaxy la cadena especificada como argumento. Las opciones *--author, --platforms* y *--galaxy-tags* se pueden usar para limitar los resultados de la búsqueda. El siguiente ejemplo muestra los nombres de los roles que incluyen **"install"** y **"git"** en su descripción, y están disponibles para la plataforma Enterprise Linux **(el)**.

```
[user@host ~]$ ansible-galaxy search 'install git' --platforms el

Found 77 roles matching your search:

 Name                                  Description
 ----                                  -----------
 zzet.gitlab                           Undev Gitlab installation
 jasonrsavino.git                      Install GIT
 samdoran.gitlab                       Install GitLab CE Omnibus
 kbrebanov.git                         Installs git
 AsianChris.git                        git installation
... Output omitted ...
```

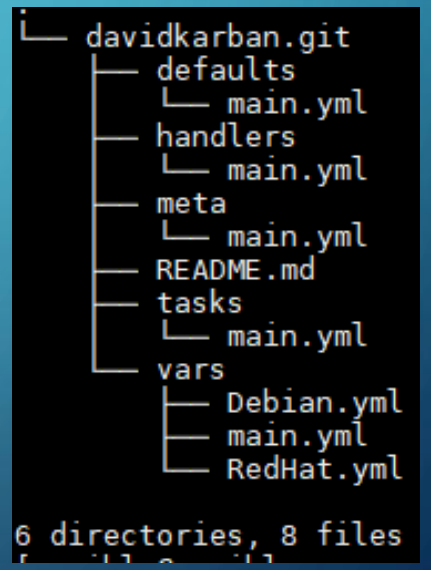
El subcomando *ansible-galaxy info* muestra información más detallada sobre un rol.

Ejem: *davidkarban.git*.

```
[user@host ~]$ ansible-galaxy info davidkarban.git
[DEPRECATION WARNING]: The comma separated role spec format, use the
yaml/explicit format instead.. This feature will be removed in a future release.
 Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less

Role: davidkarban.git
        description: Install git
        active: True
        commit:
        commit_message:
        commit_url:
        company: David Karban
        created: 2015-12-08T09:15:48.542Z
        download_count: 1
        forks_count: 0
        github_branch:
        github_repo: ansible-git
        github_user: davidkarban
        id: 6422
        is_valid: True
        issue_tracker_url: https://github.com/davidkarban/ansible-git/issues
        license: license (GPLv2, CC-BY, etc)
        min_ansible_version: 1.2
        modified: 2016-04-20T20:13:35.549Z
        namespace: davidkarban
        open_issues_count: 0
        path: /etc/ansible/roles
:
```

El subcomando *ansible-galaxy install* descarga un rol de Ansible Galaxy y luego lo instala localmente en el nodo servidor. La ubicación de instalación predeterminada para roles es */etc/ansible/roles*. Esta ubicación puede ser anulada ya sea por el valor de la variable de configuración *role_path*, o por una opción *-p DIRECTORY* en la línea de comandos.

```
[user@host ~]$ ansible-galaxy install davidkarban.git -p roles/
- downloading role 'git', owned by davidkarban
- downloading role from https://github.com/davidkarban/ansible-git/archive/master.tar.gz
- extracting davidkarban.git to roles/davidkarban.git
- davidkarban.git was installed successfully
[user@host ~]$ ls roles/
davidkarban.git
```



El *ansible-galaxy list* enumera los roles locales.

```
[user@host ~]$ ansible-galaxy list
- davidkarban.git, master
- student.bash_env, (unknown version)
```

Un rol puede eliminarse localmente con *ansible-galaxy remove*.

```
[user@host ~]$ ansible-galaxy remove student.bash_env
- successfully removed student.bash_env
[user@host ~]$ ansible-galaxy list
- davidkarban.git, master
```

El comando *ansible-galaxy init* crea una estructura de directorios para un nuevo rol que se desarrollará. El autor y el nombre del rol se especifican como un argumento para el comando y crean la estructura de directorios en el directorio actual. ansible-galaxy interactúa con la API del sitio web de Ansible Galaxy cuando realiza la mayoría de las operaciones. La opción *--offline* permite que se use el comando init cuando el acceso a Internet no está disponible.

```
[user@host roles]$ ansible-galaxy init --offline student.example
- student.example was created successfully
[user@host roles]$ ls student.example/
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

9. OPTIMIZANDO ANSIBLE

9.1 TIPOS DE CONEXIÓN:

Los tipos de conexión determinan cómo Ansible se conecta a los hosts administrados para comunicarse con ellos. De forma predeterminada, Ansible utiliza el protocolo SSH para la conexión remota a los nodos de destino.

Por lo tanto, tener una conexión SSH rápida, estable y segura es de suma importancia para Ansible.

Además de ssh, Ansible ofrece otros tipos de conexión que pueden usarse para conectarse a hosts administrados, incluyendo paramiko y local. El tipo de conexión se puede especificar en el archivo de configuración de Ansible (*ansible.cfg*), en el inventario en un host o grupo, dentro de un playbook o en la línea de comandos.

Tipos de conexión:

Varios tipos de conexión soportados por Ansible incluyen:

SMART: Este tipo de conexión comprueba si el cliente SSH instalado localmente es compatible con la función *ControlPersist*. Si se admite ControlPersist, Ansible utilizará el cliente SSH local. Si el cliente SSH no es compatible, el tipo de conexión smart volverá a usar paramiko. El tipo de conexión inteligente es el predeterminado.

La función ControlPersist está disponible en OpenSSH 5.6 o posterior. Permite que las conexiones SSH persistan, de modo que cuando los comandos frecuentes se ejecutan sobre SSH, no tienen que pasar repetidas veces por el control inicial. Cuando se alcanza el tiempo de espera de ControlPersist, la conexión SSH pasa nuevamente por el protocolo TCP. Esta configuración se puede establecer en la configuración de OpenSSH, pero se mejora en la configuración de Ansible. En */etc/ansible/ansible.cfg*, la configuración predeterminada aparece en un comentario:

```
#ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

El valor predeterminado de 60 segundos puede cambiarse a mas tiempo, incluso aunque la conexión inicial haya salido y la conexión esté inactiva durante más tiempo. Este valor se debe establecer en función del tiempo que demore la ejecución de los playbooks y la frecuencia con la que se ejecuten. Un valor de 30 minutos puede ser más apropiado:

```
ssh_args = -o ControlMaster=auto -o ControlPersist=30m
```

PARAMINKO: Es una implementación de Python del protocolo SSHv2. Ansible lo admite como el tipo de conexión para la compatibilidad con versiones anteriores de RHEL6, que no tenía soporte para la configuración ControlPersist en su versión OpenSSH.

LOCAL: La conexión local ejecuta comandos localmente, en lugar de SSH.

SSH: La conexión ssh utiliza una conexión basada en OpenSSH, que admite ControlPersist.

DOCKER: Ansible ofrece un nuevo tipo de conexión docker. Se conecta a los contenedores mediante el comando docker exec.

9.2 CONFIGURACION DE LA CONEXIÓN.

El tipo de conexión se puede especificar en el fichero ansible.cfg en la sección *[defaults]* usando *transport*.

```
[defaults]

# some basic default values...
... Output omitted ...
#transport = smart
```

El valor de *transport* se puede anular utilizando la opción *-c TRANSPORTNAME* al ejecutar un comando ad hoc o al ejecutar un ansible-playbook. La configuración para el tipo de conexión ssh se especifica en la sección *[ssh_connection]*. Del mismo modo, la configuración de paramiko se puede encontrar en la sección *[paramiko_connection]*.

```
[ssh_connection]
... Output omitted ...
# control_path = %(directory)s/%%h-%%r
#control_path = %(directory)s/ansible-ssh-%%h-%%p-%%r
#pipelining = False
#scp_if_ssh = True
#sftp_batch_mode = False
```

Fichero de inventario:

El tipo de conexión también se puede especificar en archivos de inventario para cada host utilizando la variable *ansible_connection*.

```
[targets]
localhost  ansible_connection=local
demo.lab.example.com  ansible_connection=ssh
```

Playbooks:

Los tipos de conexión también se pueden especificar cuando se ejecuta un playbook. Especifique la opción *--connection=CONNECTIONTYPE*.

```
[user@host ~]$ ansible-playbook playbook.yml --connection=local
```

También se puede indicar:

```
---
- name: Connection type in playbook
  hosts: 127.0.0.1
  connection: local
```

Configuración de variables de entorno en playbooks

A veces, un host administrado necesita configurar algunas variables de entorno de shell antes de ejecutar comandos. Por ejemplo, La variable de entorno de un servidor proxy podría necesitarse antes de instalar paquetes o descargar archivos. Ansible puede realizar estos tipos de configuración de entorno utilizando el argumento *environment* en un playbook.

Algunos módulos de Ansible, como *get_url*, *yum* y *apt*, utilizan variables de entorno para configurar su servidor proxy. El siguiente ejemplo muestra cómo configurar el servidor proxy con *$http_proxy*.

```
---
- hosts: devservers
  tasks:
     - name: download a file using demo.lab.example.com as proxy
       get_url:
         url: http://materials.example.copm/file.tar.gz
         dest: ~/Downloads
       environment:
          http_proxy: http://demo.lab.example.com:8080
```

El siguiente ejemplo muestra cómo definir una variable de entorno y hacer referencia a ella utilizando la palabra clave de *environment* para un bloque completo de tareas.

```
---
- name: Demonstrate environment variable in block
  hosts: localhost
  connection: local
  gather_facts: false
  vars:
    myname: ansible

  tasks:
  - block:
    - name: Print variable
      shell: "echo $name"
      register: result
    - debug: var=result.stdout
    environment:
      name: "{{ myname }}"
```

Cuando se usan roles Ansible, las variables de entorno se pueden pasar en el nivel de playbooks.

```
---
- hosts: demohost
  roles:
    - php
    - nginx
  environment:
    http_proxy: http://demo.lab.example.com:8080
```

9.3 CONFIGURACIÓN DE LA DELEGACION:

Para completar algunas tareas de configuración, puede ser necesario que se realicen acciones en un servidor diferente al que se está configurando. Algunos ejemplos de esto pueden incluir una acción que requiere esperar a que se reinicie el servidor, agregar un servidor a un balanceador de carga o un servidor de monitoreo, o realizar cambios en DHCP o DNS necesaria para el servidor que se está configurando.

Algunos escenarios que la DELEGATION puede manejar incluyen:

- Delegar una tarea a la máquina local.
- Delegar una tarea a un host que esté fuera del play.
- Delegar una tarea a un host que existe o no en el inventario

Delegación de tareas a la máquina local.

Realiza una acción en el nodo que ejecuta Ansible, se delega a localhost mediante *delegate_to: localhost*.

Aquí hay un playbook de muestra que ejecuta el comando *ps* en el localhost usando delegate_to y muestra el resultado usando el módulo de debug:

```
---
- name: delegate_to:localhost example
  hosts: dev
  tasks:
    - name: remote running process
      command: ps
      register: remote_process

    - debug: msg="{{ remote_process.stdout }}"

    - name: Running Local Process
      command: ps
      delegate_to: localhost
      register: local_process

    - debug:
        msg: "{{ local_process.stdout }}"
```

```
---
- name: local_action example
  hosts: dev
  tasks:
    - name: remote running process
      command: ps
      register: remote_process

    - debug: msg="{{ remote_process.stdout }} "

    - name: Running Local Process
      local_action: command 'ps'
      register: local_process

    - debug:
        msg: "{{ local_process.stdout }}"
```

Delegación de tareas a un host fuera del play.

Ansible se puede configurar con *delegate_to* para ejecutar una tarea en un host que no sea parte del play. El módulo se ejecutará en el host especificado. El siguiente ejemplo muestra como delegarar una tarea a una máquina externa *(loadbalancer-host)*. Este ejemplo ejecuta un comando en el host del balanceador local para eliminar hosts del balanceador de carga antes de implementar la última versión de web stack. Una vez finalizada esa tarea, se ejecuta una secuencia de comandos para volver a agregar los hosts al grupo del balanceo de carga.

```
- hosts: webservers
  tasks:
    - name: Remove server from load balancer
      command: remove-from-lb {{ inventory_hostname }}
      delegate_to: loadbalancer-host

    - name: deploy the latest version of web stack
      git:repo=git://foosball.example.org/path/to/repo.git dest=/srv/checkout

    - name: Add server to load balancer pool
      command: add-to-lb {{ inventory_hostname }}
      delegate_to: loadbalancer-host
```

Delegación de una tarea a un host que no existe en el inventario

Al delegar una tarea a un host que no existe en el inventario, Ansible usará el mismo tipo de conexión y los detalles utilizados para que el host administrado se conecte al host delegado. Para ajustar los detalles de la conexión, use el módulo add_host para crear un host efímero en su inventario con los datos de conexión definidos.

```
- name: test play
  hosts: localhost
  tasks:

    - name: add delegation host
      add_host: name=demo ansible_host=172.25.250.10 ansible_user=devops

    - name: echo Hello
      command: echo "Hello from {{ inventory_hostname }}"
      delegate_to: demo
      register: output

    - debug:
        msg: "{{ output.stdout }}"
```

```
[student@demo ~]$ ansible-playbook test.yml -vvv
... Output omitted ...
TASK [add delegation host] ******************************************************
task path: /home/student/ansible/dev-optimizing-ansible/inventory/test.yml:6
creating host via 'add_host': hostname=demo
changed: [localhost] => {"add_host": {"groups": [], "host_name": "demo",
"host_vars": {"ansible_host": "172.25.250.10",
"ansible_user": "devops"}}, "changed": true, "invocation":
{"module_args": {"ansible_host": "172.25.250.10",
"ansible_user": "devops", "name": "demo"}, "module_name": "add_host"}}
... Output omitted ...
TASK [silly echo] ***************************************************************
... Output omitted ...
changed: [localhost -> 172.25.250.10] => {"changed": true, "cmd": ["echo", "Hello from
 localhost"],
```

9.4 Configurando el paralelismo:

Configurar el paralelismo en Ansible usando forks.

Ansible permite mayor control sobre la ejecución del playbook ejecutando las tareas en paralelo en todos los hosts. De forma predeterminada, Ansible tiene configurado *forks* igual a 5, por lo que ejecutará una tarea en cinco máquinas diferentes a la vez.

```
[student@demo ~]$ grep forks /etc/ansible/ansible.cfg
#forks          = 5
```

Se puede cambiar usando la opción *--forks* para ansible-playbook o ansible.

En un playbook para reducir temporalmente el número de máquinas que se ejecutan en paralelo se utiliza la palabra clave *serial*:

```
---
- name: Limit the number of hosts this play runs on at the same time
  hosts: appservers
  serial: 2
```

TAREAS ASINCRONAS

Para operaciones que tardan mucho tiempo en completarse. Por ejemplo, al descargar un archivo grande o al reiniciar un servidor. Al usar el paralelismo y forks, Ansible inicia el comando en los hosts, luego sondea los hosts para determinar su estado hasta que todos hayan terminado.

Para ello use las palabras clave **async** y **poll**. La palabra clave **async** activa Ansible para ejecutar el trabajo en segundo plano y pueda verificarse más tarde y su valor será el tiempo máximo que esperará a que se complete el comando. El valor de **poll** indica a Ansible con qué frecuencia realizar una prueba para verificar si se ha completado. El valor predeterminado es de 10s.

```
---
- name: Long running task
  hosts: demoservers
  remote_user: devops
  tasks:
    - name: Download big file
      get_url: url=http://demo.example.com/bigfile.tar.gz
      async: 3600
      poll: 10
```

NOTA: Se puede configurar Ansible para esperar el trabajo todo el tiempo necesario. Para esto, establezca el valor de async a 0
Para configurar Ansible para que no espere a que se complete el trabajo, utilice poll a 0 para que Ansible inicie el comando y en lugar de sondear su finalización, pase a las siguientes tareas.

10. ANSIBLE VAULT

Es posible que Ansible necesite acceso a datos confidenciales, como contraseñas o claves API, para configurar servidores remotos. Normalmente, esta información se puede almacenar como texto sin formato en las variables de inventario u otros ficheros de Ansible. Pero cualquier usuario con acceso a los ficheros de Ansible o un sistema de control de versiones que almacena los ficheros tendrá acceso a estos datos confidenciales. Esto plantea un riesgo de seguridad evidente.

Hay dos formas principales de almacenar estos datos de forma más segura:

- Usar Ansible Vault, que puede cifrar y descifrar cualquier fichero de datos estructurados utilizado por Ansible.
- Utilizar un servicio de administración de claves de terceros para almacenar los datos en la nube, como Vault by HashiCorp, el Servicio de administración de claves AWS de Amazon o Microsoft Azure Key Vault.

Para usar Ansible Vault, se utiliza una herramienta de línea de comandos llamada *ansible-vault* para crear, editar, cifrar, descifrar y ver ficheros. Ansible Vault puede cifrar cualquier fichero de datos estructurados utilizado por Ansible.

IMPORTANTE: Ansible Vault no implementa sus propias funciones criptográficas pero utiliza un kit de herramientas externo de Python. Los ficheros están protegidos con cifrado simétrico utilizando AES256 con una contraseña como clave secreta.

Para crear un nuevo archivo cifrado, use el comando *ansible-vault create filename*. El comando solicitará la nueva contraseña del almacén y abrirá un archivo utilizando el editor predeterminado (vim) pero puede cambiarse a vi en Ansible 2.1. Se puede usar un editor diferente configurando y exportando la variable $EDITOR.

```
[student@demo ~]$ ansible-vault create secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

En lugar de ingresar la contraseña de vault a través de la entrada estándar, se puede usar un archivo de contraseña de vault para almacenar la contraseña. Este archivo deberá protegerse cuidadosamente mediante permisos de archivo y otros medios.

```
[student@demo ~]$ ansible-vault create --vault-password-file=vault-pass secret.yml
```

Para editar un archivo cifrado existente, Ansible Vault proporciona el comando *ansible-vault edit filename*. Este comando desencripta el fichero a un archivo temporal y le permite editar el archivo. Cuando se guarda, copia el contenido y elimina el archivo temporal.

```
[student@demo ~]$ ansible-vault edit secret.yml
Vault password: redhat
```

Cambia la contraseña de un archivo encriptado:

La contraseña de vault se puede cambiar con el comando *ansible-vault rekey filename*. Este comando puede reconstruir múltiples archivos de datos a la vez. Le pedirá la contraseña original y la nueva contraseña.

```
[student@demo ~]$ ansible-vault rekey secret.yml
Vault password: redhat
New Vault password: RedHat
Confirm New Vault password: RedHat
Rekey successful
```

Para encriptar un fichero que ya existe, use el comando *ansible-vault encrypt filename* o desencriptar *ansible-vault decrypt filename*. Este comando puede tomar los nombres de varios archivos para ser cifrados como argumentos.

```
[student@demo ~]$ ansible-vault encrypt secret1.yml secret2.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

PLAYBOOKS Y ANSIBLE VAULT

Para ejecutar un playbook que acceda a archivos cifrados con Ansible Vault, la contraseña de cifrado debe proporcionarse al comando ansible-playbook. Si el comando se ejecuta sin hacer esto, devolverá un error:

```
[student@demo ~]$ ansible-playbook site.yml
ERROR: A vault password must be specified to decrypt vars/api_key.yml
```

Para proporcionar la contraseña use la opción *--ask-vault-pass*.

```
[student@demo ~]$ ansible-playbook --ask-vault-pass site.yml
Vault password: redhat
```

Alternativamente, un archivo que almacena la contraseña de cifrado en texto sin formato puede usarse especificándolo con la opción *--vault-password-file*

```
[student@demo ~]$ ansible-playbook --vault-password-file=vault-pw-file site.yml
```

11. SOLUCIÓN DE PROBLEMAS DE ANSIBLE

Ficheros de log en Ansible:

Ansible permite una infraestructura de registro que se puede configurar a través del parámetro *log_path* en la sección *default* del ansible.cfg o a través de la variable de entorno $ANSIBLE_LOG_PATH. Si alguno o ambos están configurados, Ansible almacenará la salida de ansible y ansible-playbook en el fichero de log de /var/log, aunque lo normal es en el directorio local.

El módulo debug:

Proporciona una mejor comprensión de lo que está sucediendo en el nodo de control, porporciona el valor para una determinada variable en el tiempo de ejecución del playbook, es clave para tareas (por ejemplo, tareas que dependen del resultado de otras).

```
- debug: msg="The free memory for this system is {{ ansible_memfree_mb }}"
```

Errores de gestión:

Hay varios problemas que pueden ocurrir durante la ejecución de un playbook, principalmente relacionados con la sintaxis del playbook, cualquiera de las plantillas que usa o debido a problemas de conectividad con los hosts administrados (por ejemplo, un error en el nombre de host de host gestionado en el archivo de inventario). Esos errores son emitidos por el comando ansible-playbook en el momento de la ejecución. La opción *--syntax-check* comprueba la sintaxis de YAML para el playbook. Si un playbook tiene una gran cantidad de tareas, puede ser útil usar *--step* o *--start-at-task*

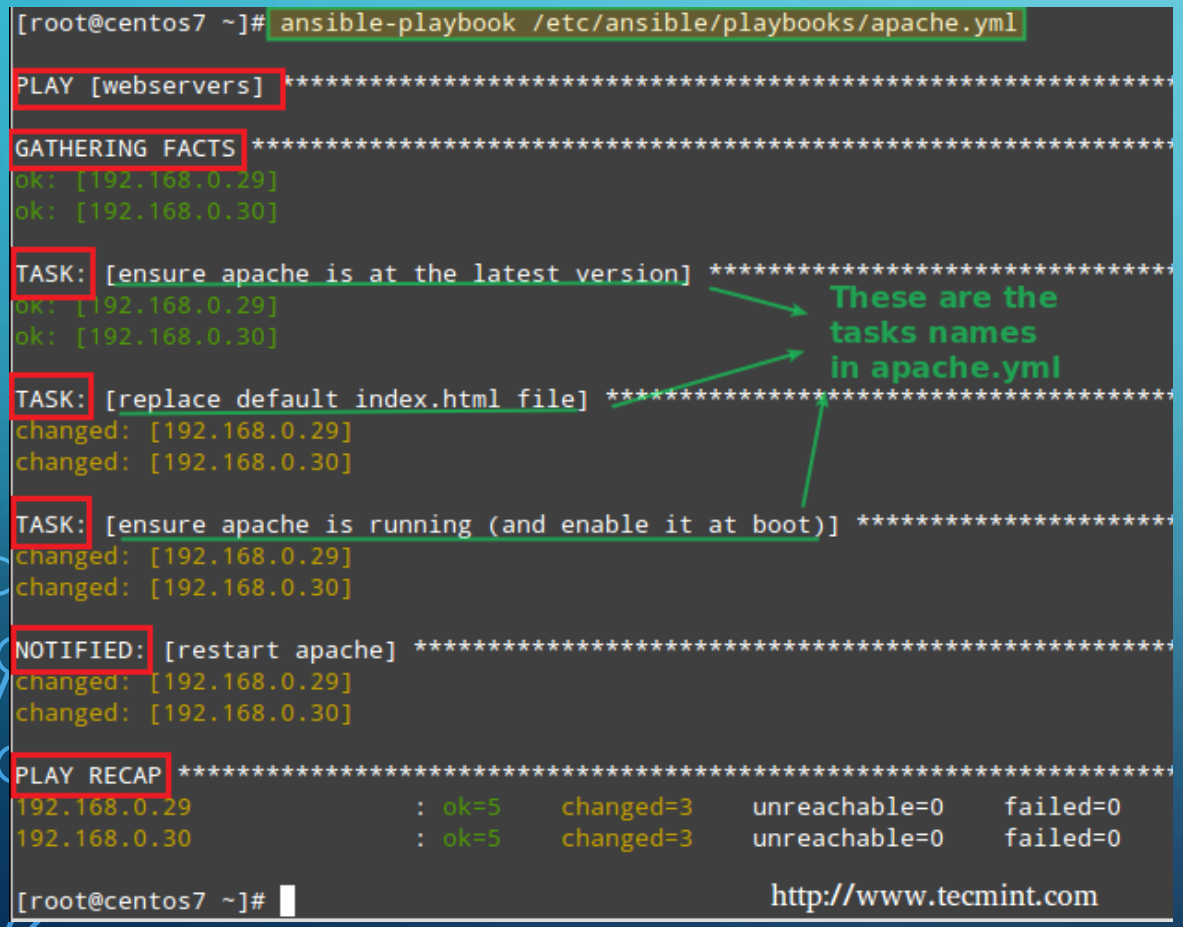
```
[student@demo ~]#ansible-playbook play.yml --step
```

```
[student@demo ~]#ansible-playbook play.yml --start-at-task="start httpd service"
```

```
[student@demo ~]#ansible-playbook play.yml --syntax-check
```

Depuración con playbook:

El resultado dado por un ansible-playbook es un buen punto de partida para comenzar a resolver problemas relacionados con hosts administrados. Por ejemplo, se ejecuta un playbook y muestra la siguiente salida:



El encabezado PLAY muestra el nombre del play que se ejecutará, luego se incluyen uno o más encabezados TASK y se ejecutará en todos los hosts incluidos en el playbook en el parámetro hosts.

El host se muestra bajo el encabezado TASK correspondiente, junto con el estado de la tarea en ese host administrado, que puede configurarse como **ok, fatal** o **changed** . En la parte inferior del playbook, en la sección **PLAY RECAP**, se muestra la cantidad de tareas ejecutadas para cada host administrado como su estado de salida.

La salida predeterminada de ansible-playbook no proporciona suficientes detalles para solucionar los problemas que pueden darse. El comando *ansible-playbook -v* proporciona información de depuración adicional, con hasta cuatro niveles.

Verbosity configuration	
Option	**Description**
-v	The output data is displayed.
-vv	Both the output and input data are displayed.
-vvv	Includes information about connections to managed hosts.
-vvvv	Adds extra verbosity options to the connection plug-ins, including the users being used in the managed hosts to execute scripts, and what scripts have been executed.

Prácticas recomendadas para la gestión de libros de jugadas:

Aunque las herramientas discutidas anteriormente pueden ayudar a identificar y solucionar problemas en los libros de jugadas, es importante tener en cuenta algunas prácticas recomendadas que pueden ayudar a facilitar el proceso para solucionar problemas. Estas son algunas de las prácticas recomendadas para el desarrollo de playbooks.

- Utilizar *"name"* en las tareas, proporcionando una descripción en el nombre del propósito de la tarea. Este nombre se muestra cuando se ejecuta el playbook.
- Incluir comentarios para agregar documentación adicional en línea sobre las tareas.
- Hacer uso del espacio en blanco. La sintaxis de YAML se basa principalmente en espacios, por lo tanto, evite el uso de tabulaciones para evitar errores.
- Trate de mantener el playbook lo más simple posible. Solo utiliza las funciones que necesites.

Solución de problemas de Ansible Managed Hosts

Puede usar el comando *ansible-playbook --check* para ejecutar las tareas sin realizar ningún cambio "smoke testing".

```
[student@demo ~]# ansible-playbook --check playbook.yml
```

El comando ansible-playbook --check se usa cuando todas las tareas deben ejecutarse en el modo de verificación, pero si solo algunas de esas tareas deben ejecutarse en el modo de verificación, la opción *always_run* es mejor. Cada tarea puede tener asociada una cláusula always_run. El valor para esta cláusula es *true* si la tarea debe ejecutarse en modo de verificación o *false* si no lo es.

```
tasks:
  - name: task in check mode
    shell: uname -a
    always_run: yes
```

Ansible también proporciona la opción *--diff*. Esta opción informa de los cambios realizados en los archivos de plantilla en los hosts administrados. Si se usa con la opción *--check*, esos cambios se muestran y no se realizan realmente.

```
[student@demo ~]# ansible-playbook --check --diff playbook.yml
```

Módulos para pruebas

Algunos módulos pueden proporcionar información adicional sobre el estado de un host administrado.

- El módulo uri proporciona una manera de verificar que una API RESTful esté devolviendo el contenido requerido.

```
tasks:
  - action: uri url=http://api.myapp.com return_content=yes
    register: apiresponse

  - fail: msg='version was not provided'
    when: "'version' not in apiresponse.content"
```

- El módulo *script* admite la ejecución de un script en un host administrado, y falla si el código de retorno no es cero. El script debe estar en el nodo de control y se transferirá (y ejecutará) en el host administrado.

```
tasks:
  - script: check_free_memory
```

- El módulo *stat* puede verificar que los ficheros y directorios no administrados directamente por Ansible estén presentes. El módulo de *assert* en el siguiente ejemplo verifica que exista un archivo en el host administrado.

```
tasks:
  - stat: path=/var/run/app.lock
    register: lock

  - assert:
      that:
        - lock.stat.exists
```

Usando comandos ad hoc para pruebas:

Las comprobaciones que se pueden realizar en un host administrado mediante el uso de comandos ad hoc. Esos ejemplos utilizan módulos como *yum* para realizar verificaciones en los nodos administrados. Este ejemplo comprueba que el httpd está instalado actualmente en el host administrado de demohost.

```
[student@demo ~]# ansible demohost -u devops -b -m yum -a 'name=httpd state=present'
```

Este ejemplo devuelve el espacio disponible actualmente en los discos configurados en el host administrado de demohost.

```
[student@demo ~]# ansible demohost -a 'lsblk'
```

Este ejemplo devuelve la memoria libre disponible actualmente en el host administrado de demohost.

```
[student@demo ~]# ansible demohost -a 'free -m'
```

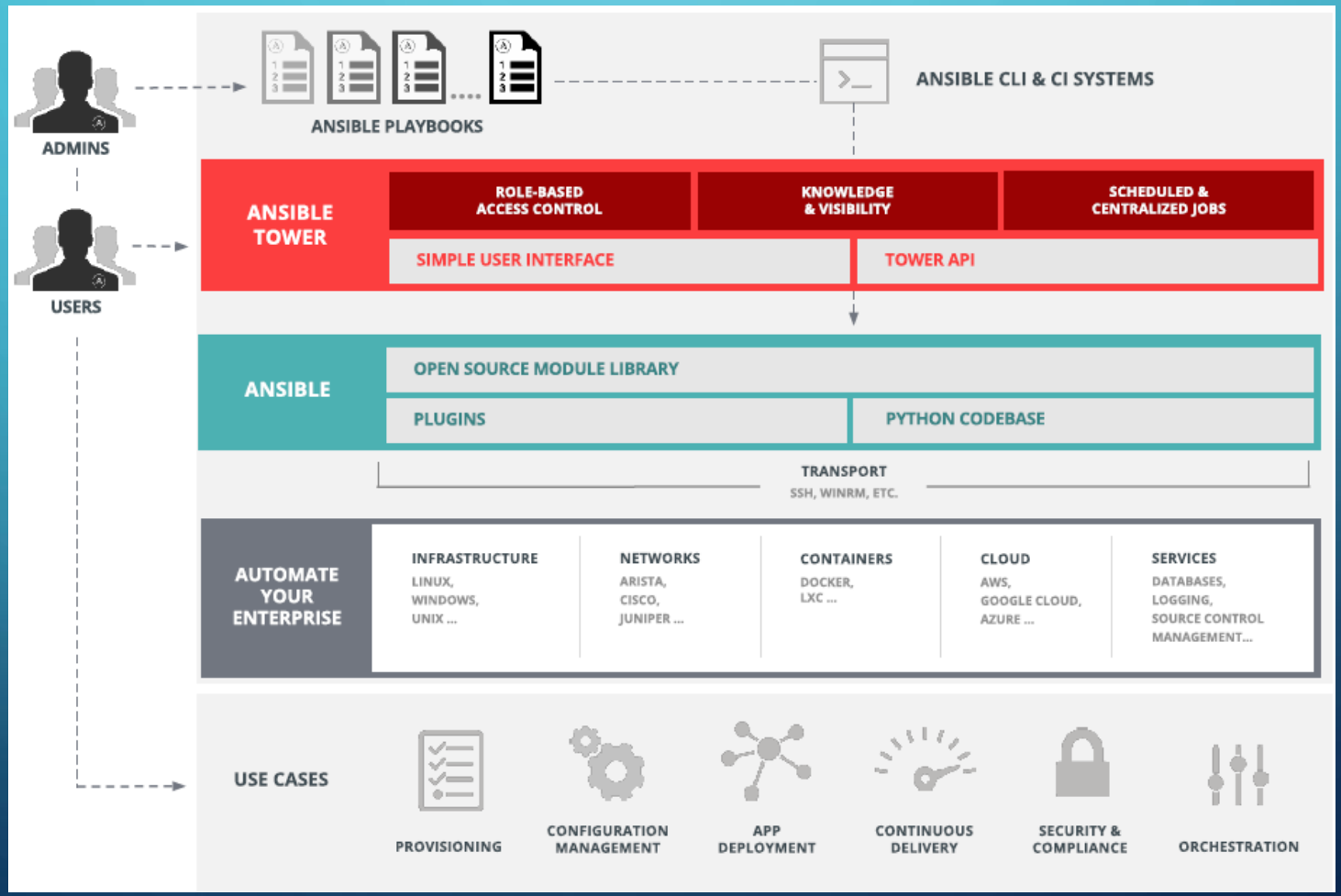
12. ANSIBLE TOWER

Ansible Tower es una interfaz de usuario que proporciona una solución empresarial para la automatización de TI. Cuenta con un panel de control fácil de usar para administrar implementaciones y monitorear recursos. Ansible Tower complementa a Ansible, y agrega capacidades de automatización, gestión visual y monitoreo.

Tower proporciona control de acceso a los administradores. Facilita el uso de las credenciales SSH para la administración, al mismo tiempo que impide el acceso directo a esas credenciales o la transferencia de dichas credenciales. El inventario de Ansible se puede administrar gráficamente o sincronizar con una amplia variedad de fuentes en la nube, incluida la plataforma OpenStack de Red Hat. Incluye actividad reciente de hosts y problemas que pueden haber surgido.

Arquitectura de torre ansible:

Ansible Tower se basa en una aplicación web Django y se basa en un back-end de base de datos que utiliza PostgreSQL. La instalación predeterminada instala todos los componentes requeridos por Ansible Tower en una sola máquina.



Características de ansible tower:

• Soporte para la automatización de las ejecuciones de playbook, la sincronización de inventario y el control de la fuente del proyecto, por lo que se pueden programar para que se ejecuten en un momento dado, pudiendo monitorear el estado de un host administrado.

• Una API REST que admite todas las funciones proporcionadas a través del panel. Tower también proporciona un CLI que soporta esta API RESTful.

• Se mejoró la ejecución del playbook al proporcionar una forma fácil de suministrar variables del playbook y seleccionar las credenciales, monitorear el playbook mientras se ejecuta y visualizar los resultados con muchas vistas diferentes, incluido un inventario gráfico con el historial de cada host administrado.

• Control de acceso basado en roles, que permite a diferentes equipos o usuarios administrar los recursos de tower según sus requisitos.

12.1 DESPLEGANDO ANSIBLE TOWER

Al instalar Ansible Tower, el nodo de control debe tener instalada la última versión de Ansible. Tower es una aplicación basada en navegador y el proceso de instalación instala varias dependencias, como PostgreSQL, Django, Apache HTTPD y otras. Se requiere que Ansible Tower esté instalado en un servidor independiente y no se debe ubicar junto con ninguna otra aplicación.

Requisitos de ansible tower:

Ansible Tower es compatible con sistemas basados en Linux en la arquitectura x86-64, incluidos Red Hat Enterprise Linux 6, Red Hat Enterprise Linux 7, CentOS 6, CentOS 7, Ubuntu 12.04 LTS y Ubuntu 14.04 LTS. Este curso se centrará en el procedimiento de instalación para Red Hat Enterprise Linux 7 o CentOS 7. Se requiere un mínimo de 2 GB de RAM y 20 GB de espacio en el disco duro para la instalación.

Instalación en paquete:

Tener en cuenta que el instalador incluido en ansible, solo admite la instalación en Red Hat Enterprise Linux o CentOS.

Instalación de ansible tower:

El primer paso es descomprimir el paquete de instalación y ejecutar un script que configurará un playbook y un archivo de inventario que se usará para instalar Tower. Puede ejecutar tanto el paso de configuración como el playbook desde un nodo de control remoto o localmente.

Primero se debe extraer el paquete de instalación que viene en el archivo como un archivo gzip comprimido.

```
[student@demo ~]$ tar xvf ansible-tower-setup-bundle-1.el7.tar.gz
```

```
[student@demo ansible-tower-setup-bundle-2.4.5-1.el7]$ ./configure
```

Configuración de la máquina:

El fichero extraído contiene un script de configuración que se utiliza para configurar el playbook y el archivo de inventario según los parámetros que se pasaron cuando se ejecuta el script.

Option	Description
-l, **--local**	Install Ansible Tower on the local machine with an internal PostgreSQL database.
--no-secondary-prompt	Skip prompts regarding secondary Tower nodes to be added.
-A, **--no-autogenerate**	Disable auto-generation of passwords for PostgreSQL, and instead prompts the user for passwords.
-o ***FILE***, **--option-file=*FILE***	Use *FILE* as the source of answers for configuration.

El playbook de configuración se crea en el directorio de instalación con el nombre *tower_setup_conf.yml*. Este fichero contiene las contraseñas de Tower necesarias, la información de conexión a la base de datos y la información de conexión SSH.

```
[student@demo ansible-tower-setup-bundle-2.4.5-1.el7]$ ./setup.sh
```

El script setup.sh se ejecuta para ejecutar el playbook e instala Ansible Tower.

Option	Description
`-c FILE`	Specify the file that stores the Tower configuration. The default file is **`tower_setup_conf.yml`** in the setup bundle directory.
`-i FILE`	Specify the file that is to be used for the host inventory. The default file is **`inventory`** in the setup bundle directory.
`-p`	Require Ansible to prompt for SSH passwords when connecting to remote machines.
`-s`	Require Ansible to prompt for **sudo** passwords on remote machines when installing Tower.
`-u`	Require Ansible to prompt for **su** passwords on remote machines when installing Tower.
`-e`	Set additional Ansible variables for Tower to use during installation.
`-b`	Perform a database backup instead of installing Tower.
`-r BACKUP_FILE`	Perform a database restore from the file specified, instead of installing Tower.

Ansible Tower Dashboard

Cuando inicie sesión en Ansible Tower mediante la interfaz de usuario web, el administrador puede ver un gráfico que muestra toda la actividad reciente del trabajo, el número de hosts administrados y los punteros rápidos a las listas de hosts con problemas. El panel también muestra datos en tiempo real sobre la ejecución de tareas completadas en los playbooks.

12.2 Configurando Usuarios en Ansible Tower:

Hay tres tipos de usuarios en Ansible:

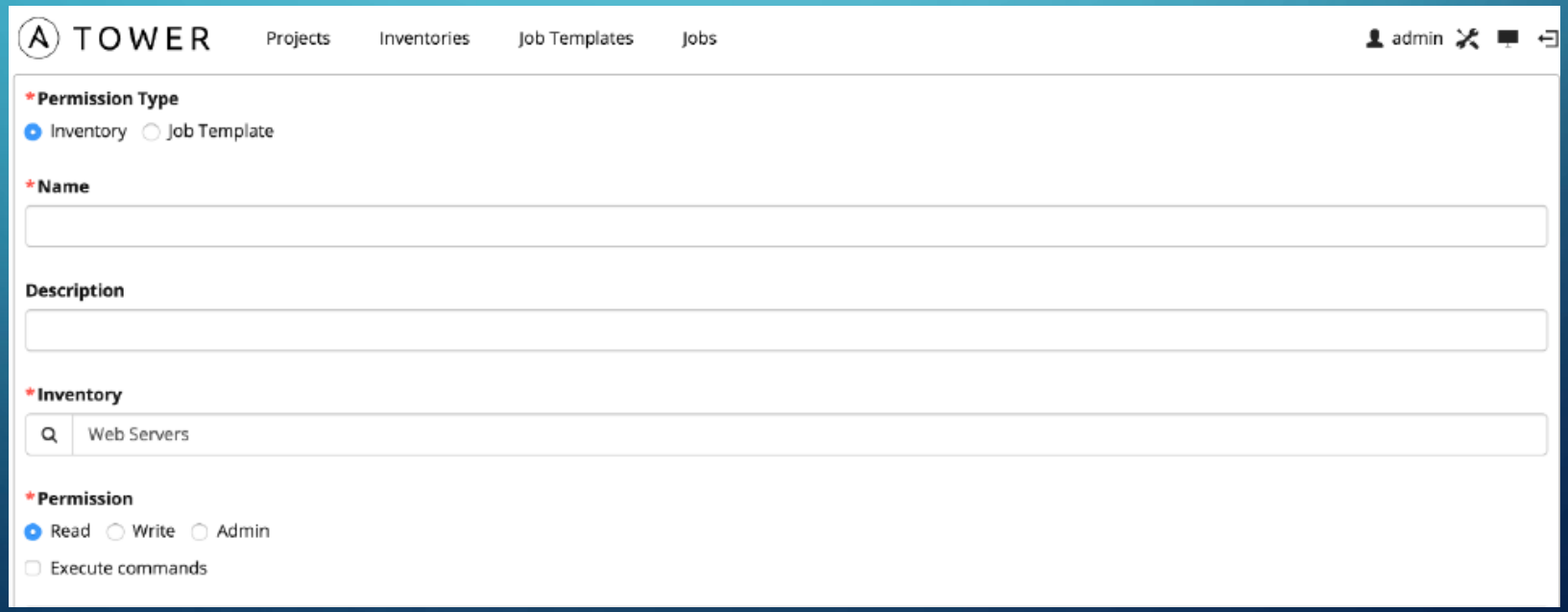
- **Usuario normal:** Los usuarios normales tienen acceso de lectura y escritura al inventario y a los proyectos que han sido asignados.
- **Administrador de la organización:** Tienen los derechos de los usuarios normales, así como el acceso de lectura y escritura sobre toda la organización y todos sus inventarios y proyectos. No tienen acceso al contenido que pertenece a otras organizaciones. Pueden crear y administrar usuarios dentro de su propia organización.
- **Superusuario:** Los superusuarios tienen permisos de lectura y escritura en toda la instalación de la Tower. Son administradores de sistemas que son responsables de administrar Ansible Tower y de delegar responsabilidades a los administradores de la organización.

Permisos de usuario:

Privilegios asignados a los usuarios o equipos de usuarios. Determinan la capacidad de los usuarios para leer, modificar y administrar proyectos, inventarios y plantillas de trabajo.

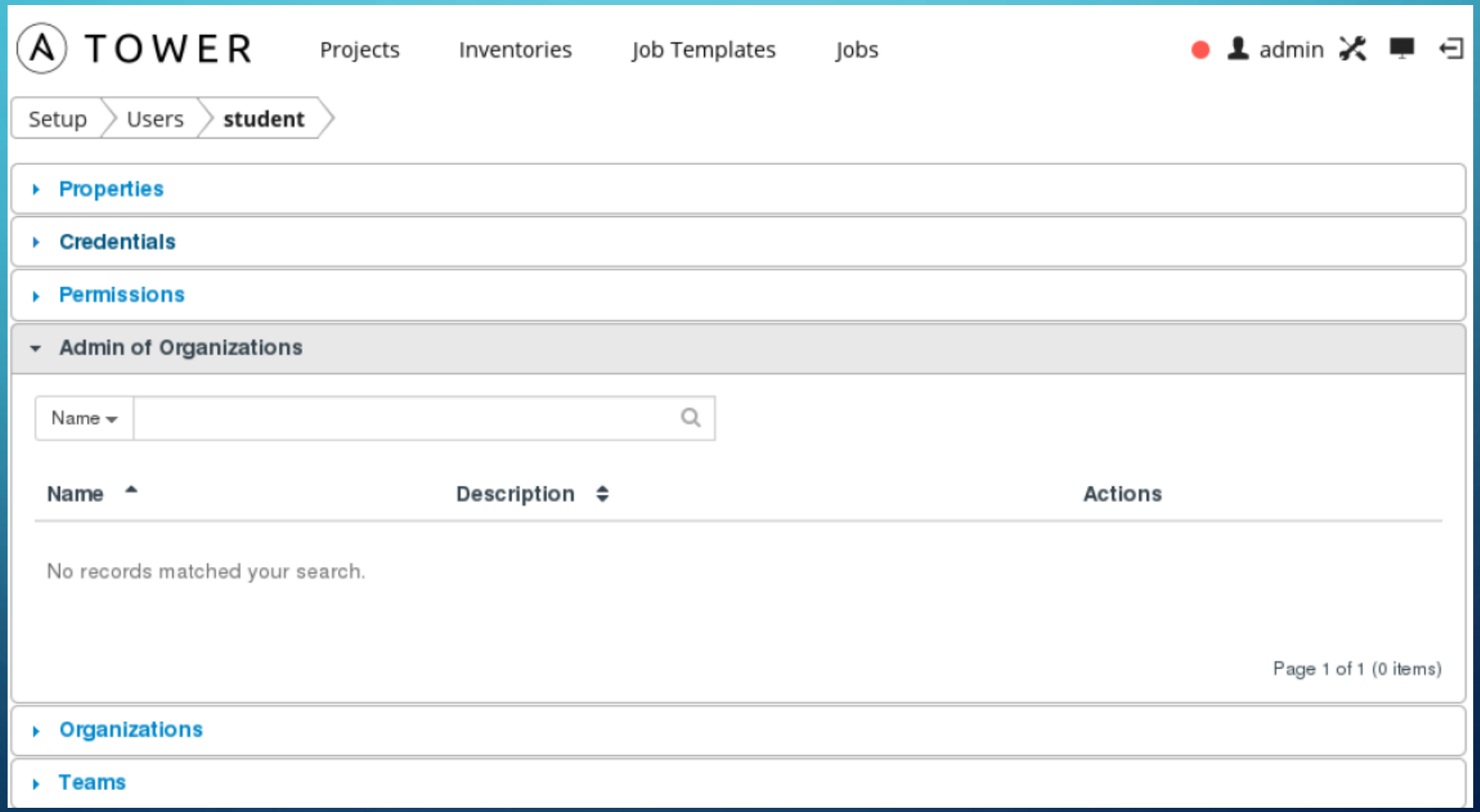
- **INVENTARIO:** Otorga permiso a los usuarios para administrar inventarios, grupos y hosts.
  - ✓**READ**: Permite ver grupos y hosts dentro de un inventario específico.
  - ✓**WRITE**: Permite la creación, modificación y eliminación de grupos y hosts .
  - ✓**ADMIN**: Permite modificar las configuraciones para el inventario especificado. Este nivel de permiso otorga permisos de lectura y escritura.
  - ✓**EXECUTE COMMANDS:** Permite al usuario ejecutar comandos en el inventario.

- **JOB TEMPLATE:** Otorga permiso para iniciar trabajos del proyecto especificado contra el inventario especificado:
  - ✓**CREATE:** Permite a los usuarios o equipos crear plantillas de trabajo. Este rol requiere permisos de run y de check.
  - ✓**RUN:** Permite a los usuarios o equipos ejecutar trabajos. Darle a un usuario o equipo este nivel también otorga los permisos de check.
  - ✓**CHECK:** Permite a los usuarios o equipos iniciar trabajos en modo dry-run. Los elementos que se cambiarían se informan, pero los cambios no se realizan realmente.



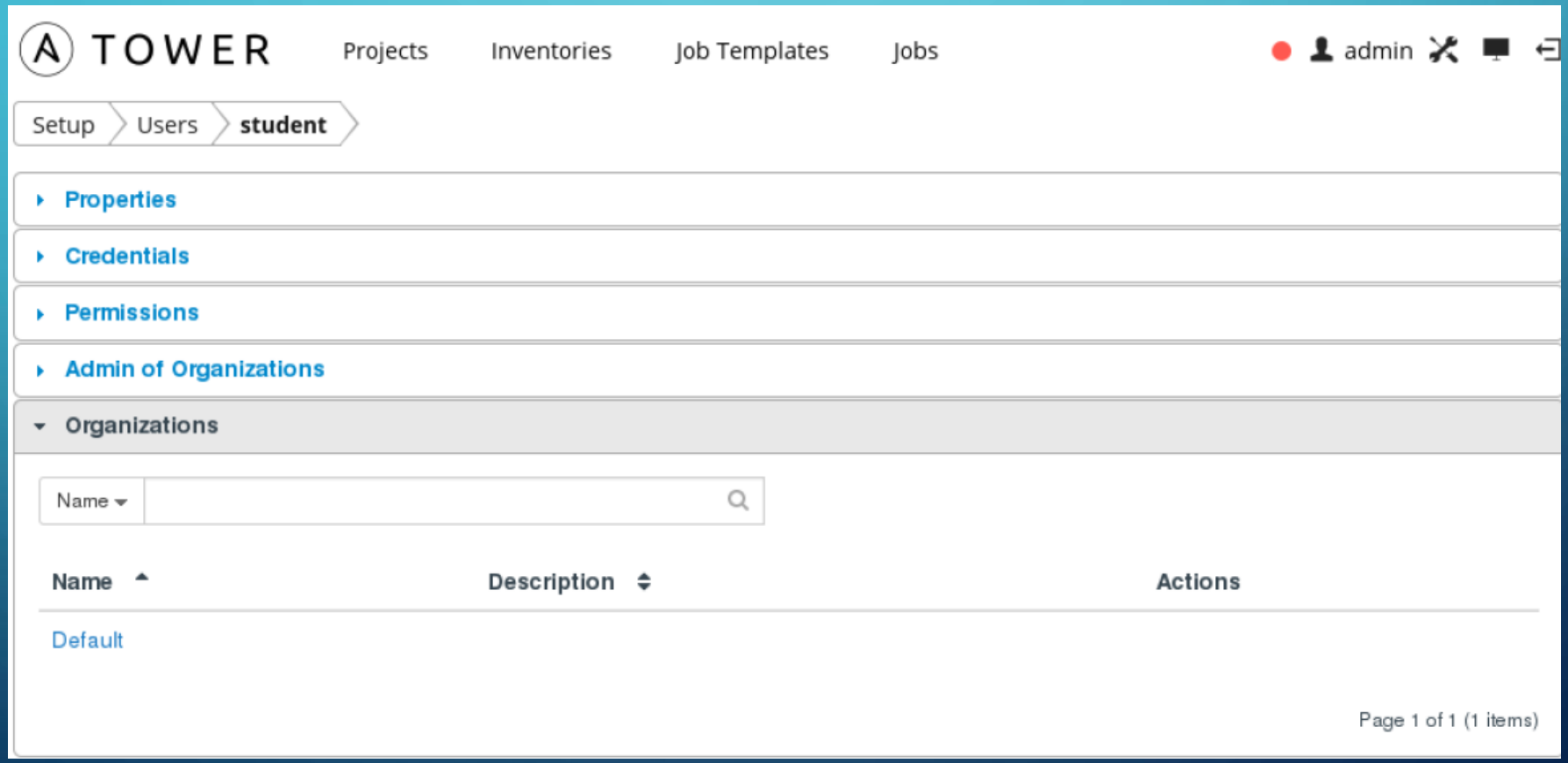
- **ADMIN OF ORGANIZATIONS:**

Los usuarios pueden configurarse como administradores de organización, lo que les brinda la capacidad de administrar inventarios y proyectos de la organización.



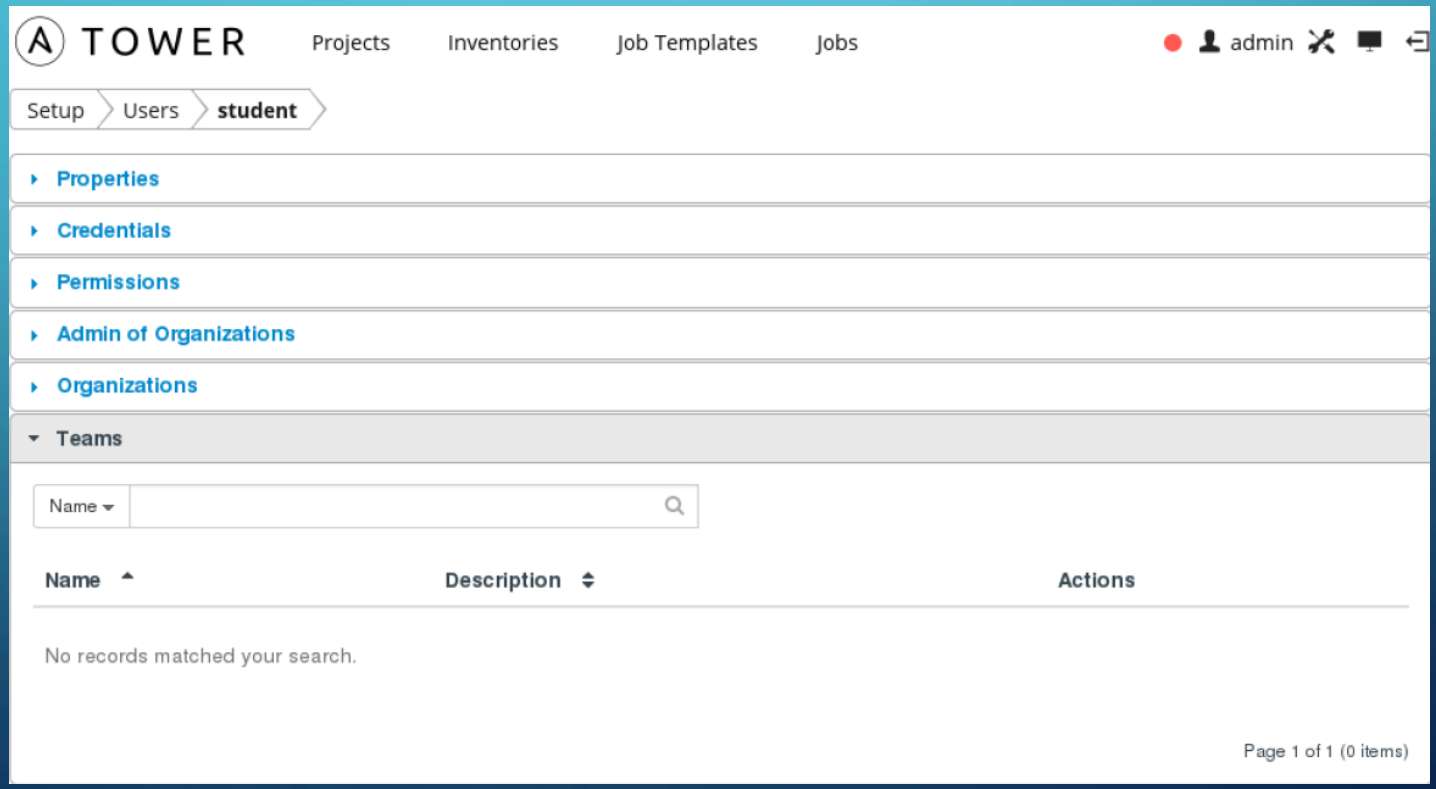
- **ORGANIZATIONS:**

Ansible Tower se envía con la organización predeterminada, pero los administradores pueden definir sus propias organizaciones para la agrupación lógica de usuarios.



- **USER TEAMS:**

Los teams son una subdivisión de una organización y tienen usuarios, proyectos, credenciales y permisos asociados. Proporcionan estructuras de control de acceso basados en roles y delegar responsabilidades en las organizaciones. Permiten asignar fácilmente los mismos permisos a un conjunto de usuarios, en lugar de configurar permisos manualmente para cada usuario individualmente.



12.3 GESTIONANDO HOSTS CON TOWER

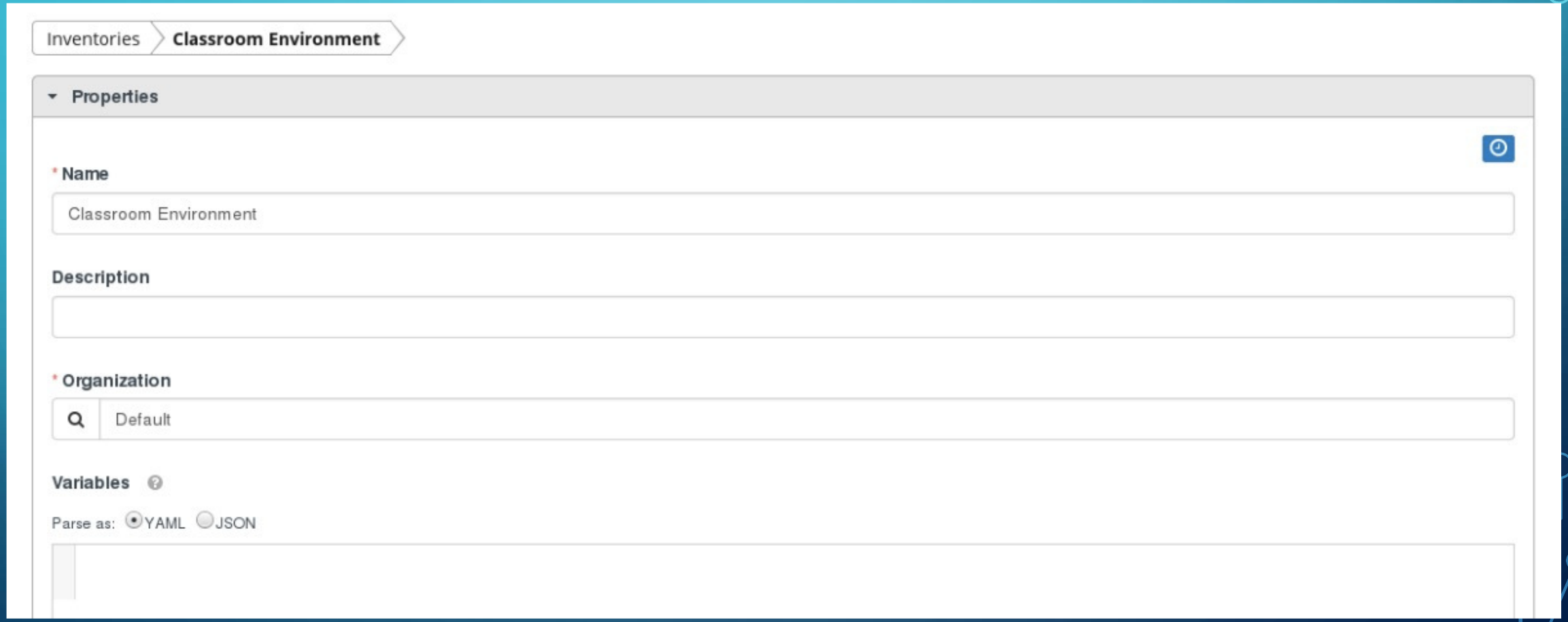
INVENTARIOS DE TOWER:

Los administradores pueden administrar los inventarios de Ansible directamente en la interfaz web de Ansible Tower.

Recuerde que un inventario es una colección de hosts en los que Ansible Tower puede ejecutar trabajos. La siguiente tabla muestra los campos disponibles:

CAMPO	DESCRIPCION
Name	El nombre del inventario a crear. Este campo es obligatorio
Description	Más información sobre el inventario. Este campo es opcional.
Organization	La organización a la que pertenece el inventario. Esto controla qué usuarios tienen acceso a este inventario.
Variables	Variables específicas del host, en formato JSON o YAML, que se pueden usar en proyectos Tower.

- Se puede editar un inventario existente haciendo clic en el ícono del lápiz al lado de su nombre. La siguiente captura de pantalla muestra la ventana **Crear inventario:**



Los administradores pueden ejecutar comandos ad hoc contra un inventario haciendo clic en el icono del cohete . El asistente enumera los módulos compatibles y las opciones de módulo que se pueden pasar como argumentos.

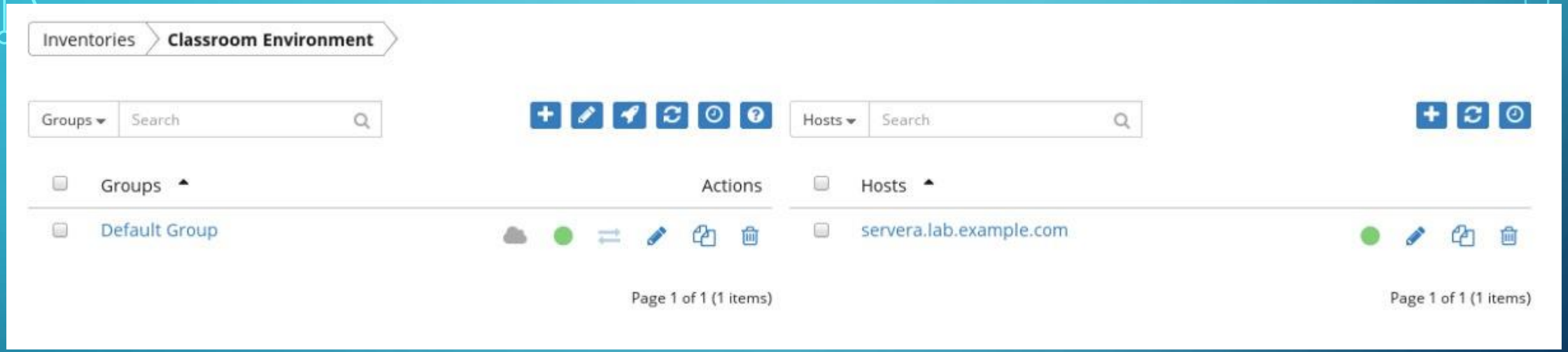
La siguiente captura de pantalla muestra el asistente para comandos ad hoc:

Gestionando grupos con Ansible Tower

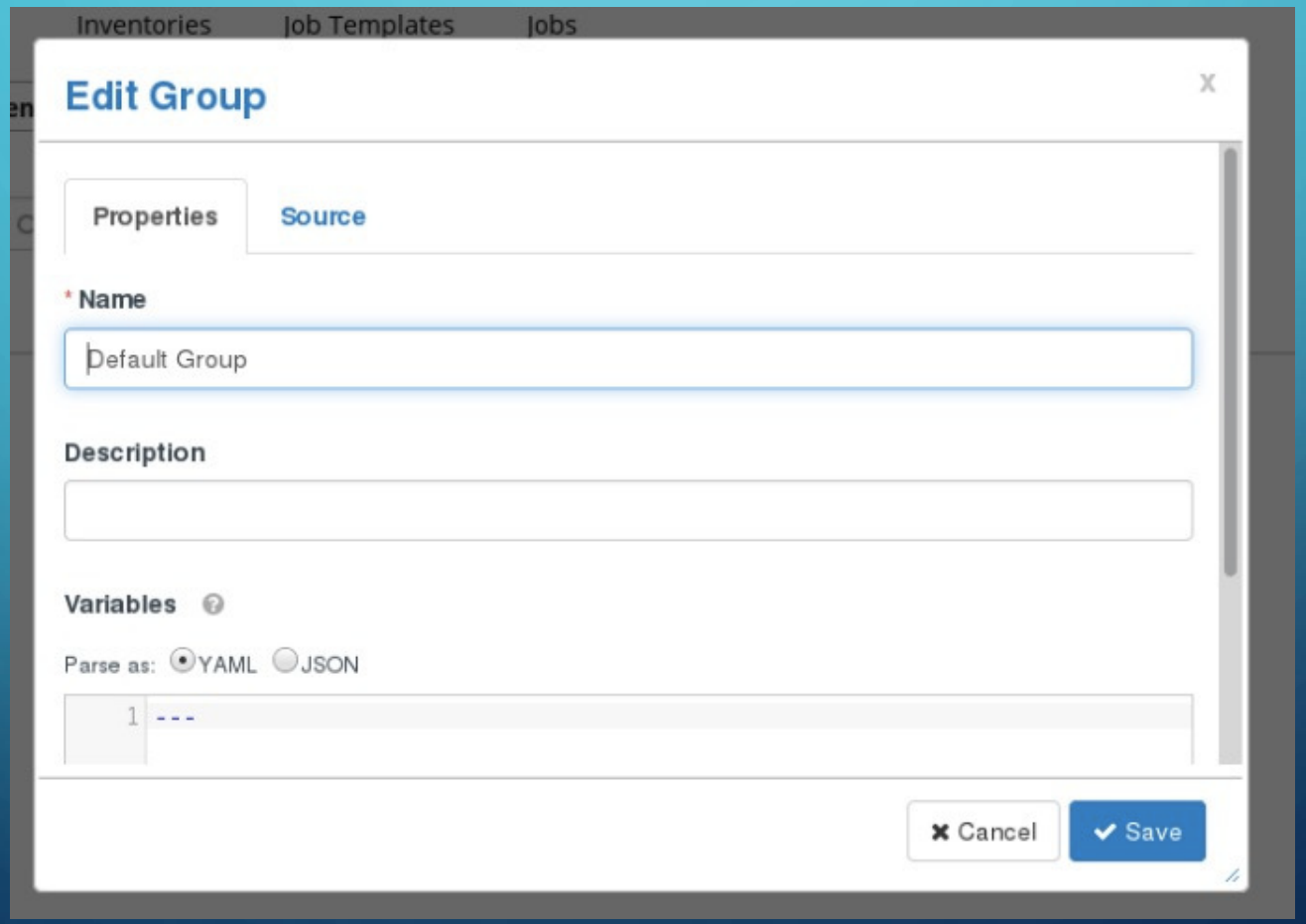
Una vez que se crea un inventario, los hosts y los grupos de hosts se pueden crear en él. Los grupos permiten la organización lógica de los hosts y también se pueden usar para el inventario dinámico de hosts desde un proveedor de la nube u otra fuente de información. La siguiente tabla muestra las opciones disponibles para los grupos de inventario:

CAMPO	DESCRIPCION
Name	El nombre del inventario a crear. Este campo es obligatorio
Description	La descripción detallada y opcional para el inventario.
Variables	Variables específicas del host, en formato JSON o YAML, que se pueden usar en proyectos Tower.
Source	La fuente para gestionar hosts. La fuente puede ser un proveedor de nube o infraestructura.

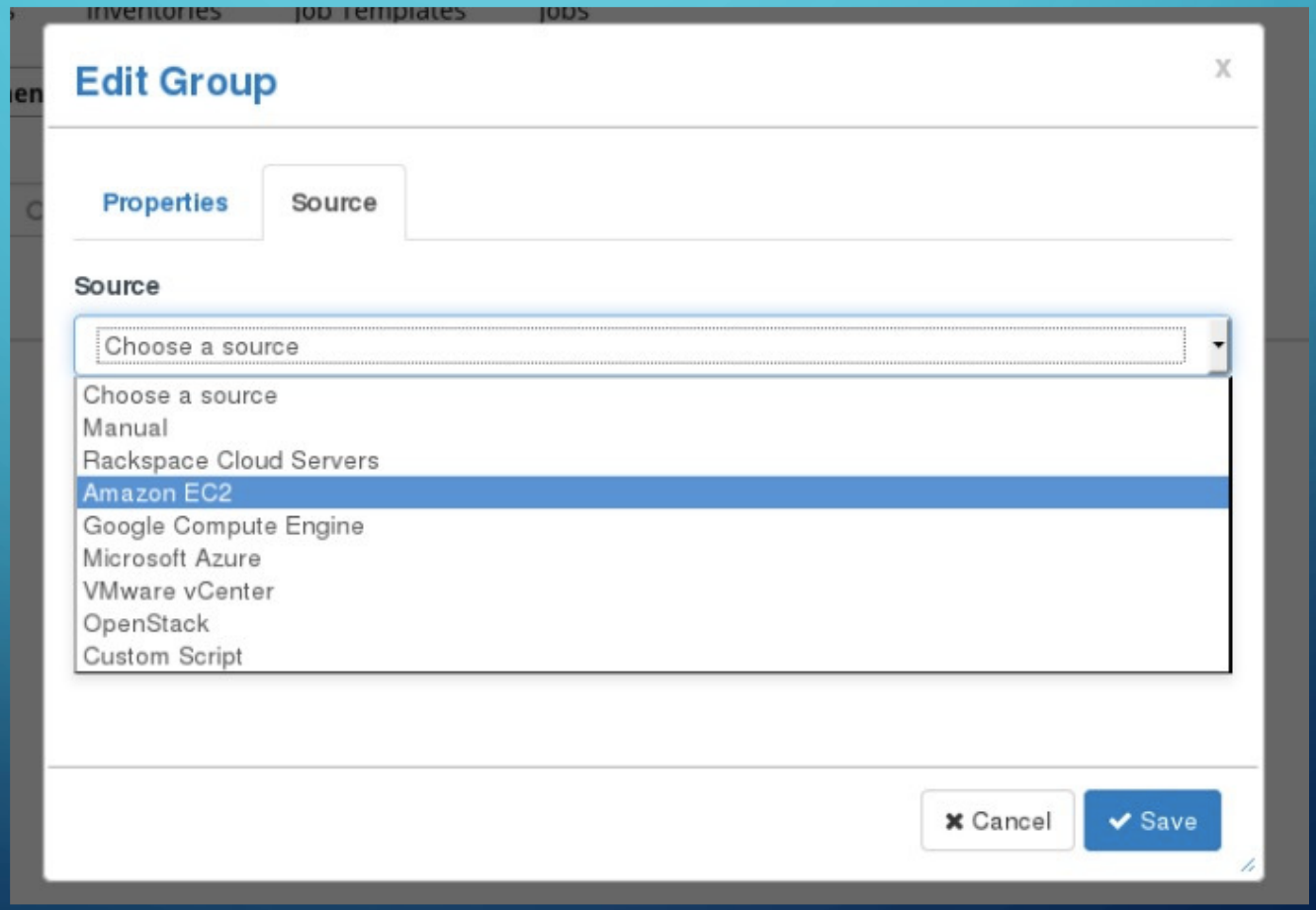
La siguiente captura de pantalla muestra un resumen de inventario para la administración de host y grupo:



Un grupo puede ser editado o copiado. Un grupo también se puede mover para que sea un subgrupo de un grupo existente dentro de una organización. Los grupos de hosts en Ansible Tower siguen los mismos principios que los grupos de hosts normales creados dentro de los archivos de inventario.



Los administradores pueden elegir varias fuentes para el aprovisionamiento de host dentro de un inventario. Antes de poder descubrir automáticamente los hosts de los proveedores, se debe definir un conjunto de credenciales para un equipo o un usuario. Las credenciales se utilizarán para conectarse al punto final del proveedor.



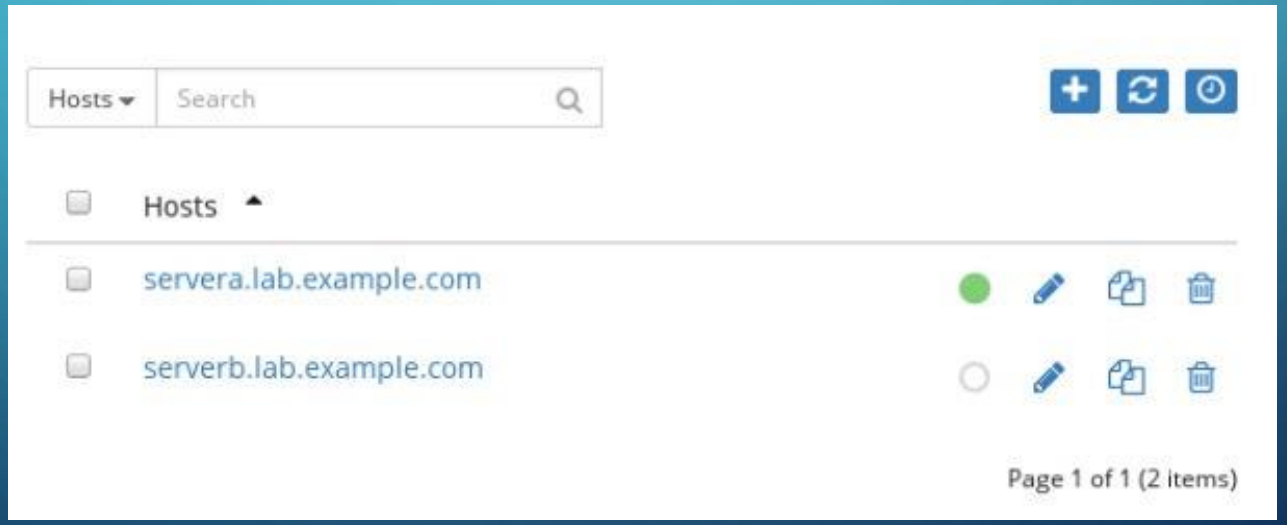
GESTIONANDO HOST CON ANSIBLE TOWER

Tras la creación de un inventario, los hosts pueden agregarse manualmente o pueden descubrirse automáticamente desde una plataforma o proveedor en la nube. Los ejemplos de dichos proveedores incluyen Amazon EC2, Microsoft Azure, OpenStack, Rackspace Public Cloud y otros. La siguiente tabla muestra los campos disponibles para los hosts de inventario en Ansible Tower:

Componentes del nodo de ansible:

CAMPO	DESCRIPCION
Host Name	El nombre de dominio completo o la dirección IP del host que se va a administrar. Este campo es obligatorio.
Description	La descripción del anfitrión. Este campo es opcional.
Enabled	Esta casilla de verificación determina si incluir o excluir un host de los plays.
Variables	Variables específicas del host, en formato JSON o YAML, que se pueden usar en proyectos Tower.

Una vez que se ha creado un host, se puede quitar, copiar o mover a otro inventario. El círculo al lado de cada host indica el estado de los trabajos ejecutados contra el host. Un círculo vacío significa que no se han ejecutado trabajos contra el host. Un círculo verde indica que el último trabajo ejecutado contra el host tuvo éxito. Un círculo rojo significa que el último trabajo ejecutado contra el host falló. La siguiente captura de pantalla muestra varios estados para hosts administrados:



12.4 GESTION DE JOBS EN ANSIBLE TOWER:

Ansible Tower permite que los jobs se utilicen para ejecutar comandos ad hoc o para ejecutar playbooks. Una colección lógica de playbooks se organiza en un *project*. Tower permite crear *job templates* que pueden predefinir cómo ejecutar un playbook desde un project y que pueden reutilizarse repetidamente y compartirse con otros usuarios.

PROJECTS:

Los projects son colecciones de playbooks en Ansible Tower. Estos playbooks residen en la instancia de Ansible Tower o en un sistema de control de versiones compatible con Tower, como Git, Subversion o Mercurial. El directorio predeterminado de project donde Ansible Tower busca playbooks es */var/lib/awx/projects*, pero se puede modificar utilizando la variable de entorno $PROJECT_BASE. Los directorios y archivos de project deben ser propiedad del usuario awx, que ejecuta el servicio Tower.

Añadir un nuevo proyecto:

Para crear un nuevo proyecto, haga clic en + en la página Projects . Si está utilizando un directorio local en la instancia de Tower, use el Tipo de SCM como Manual, después de crear manualmente el directorio del proyecto en /var/lib/awx/projects que es propiedad del usuario awx.

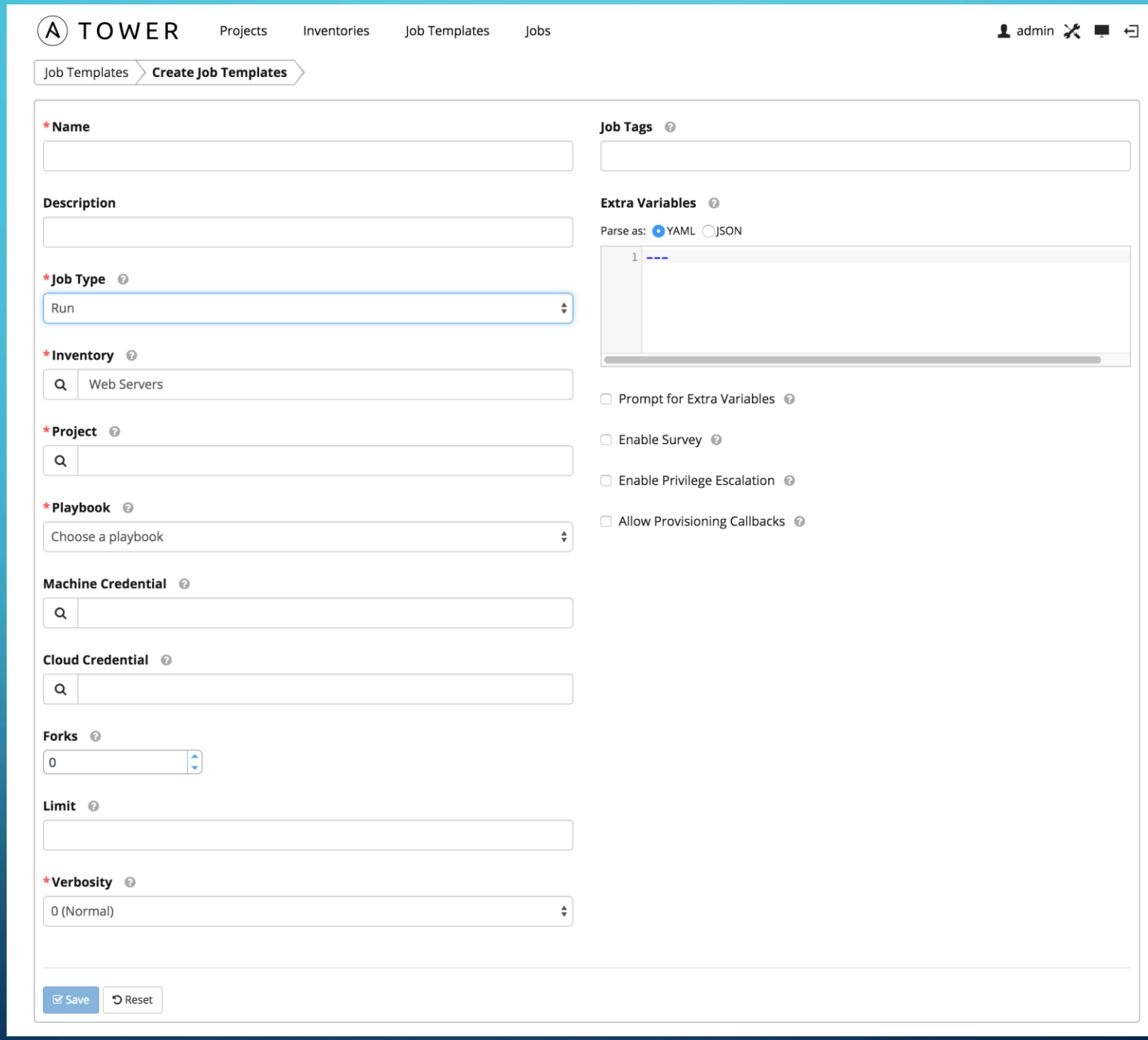
```
[student@demo ~]$ sudo mkdir /var/lib/awx/projects/demoproject
[student@demo ~]$ sudo cp demo.yml /var/lib/awx/projects/demoproject
[student@demo ~]$ sudo chown -R awx /var/lib/awx/projects/demoproject
```

CREAR UN NUEVO JOB TEMPLATE:

Para crear un nuevo job template, haga clic en + en la página JOB TEMPLATES. La página tiene los siguientes campos:

- ✓ **NAME:** El nombre a asociar con la plantilla de trabajo.
- ✓ **JOB TYPE:** Puede ser **RUN** (para ejecutar el playbook cuando se inicia), **CHECK** (verifica solo la sintaxis y la configuración del entorno pero no ejecuta el playbook), o **SCAN** (recopila información del sistema).
- ✓ **INVENTORY:** El inventario a utilizar al ejecutar esta plantilla de trabajo.
- ✓ **PROJECT:** Proyecto para ser utilizado con esta plantilla de trabajo.
- ✓ **PLAYBOOK:** El playbook que se lanzará con esta plantilla de trabajo.
- ✓ **CREDENTIAL:** Cuál de las credenciales del usuario usar con esta plantilla de trabajo para autenticarse en los hosts administrados.
- ✓ **FORKS:** La cantidad de procesos paralelos o simultáneos que se deben utilizar al ejecutar el libro de jugadas.

- ✓ **LIMIT:** Un patrón de host para restringir aún más la lista de hosts gestionados o afectados por el libro de jugadas.
- ✓ **JOB TAGS:** Una lista separada por comas de etiquetas de playbook para restringir qué partes de los playbook se ejecutan.
- ✓ **EXTRA VARIABLES:** Pasa las variables de línea de comando adicionales al playbook. Este es el parámetro de línea de comando **-e** o ***--extra-vars*** para el comando ansible-playbook.
- ✓ **PROMPT FOR EXTRA VARIABLES:** Si se marca esta opción, se le pedirá al usuario Extra variables en la ejecución del trabajo.
- ✓ **ENABLE SURVEY:** Establezca variables adicionales a través de un asistente fácil de usar cuando el job template se utiliza para iniciar un job. La encuesta también validará la entrada del usuario antes de usarla. Las encuestas solo están disponibles para clientes con licencias de edición Enterprise.

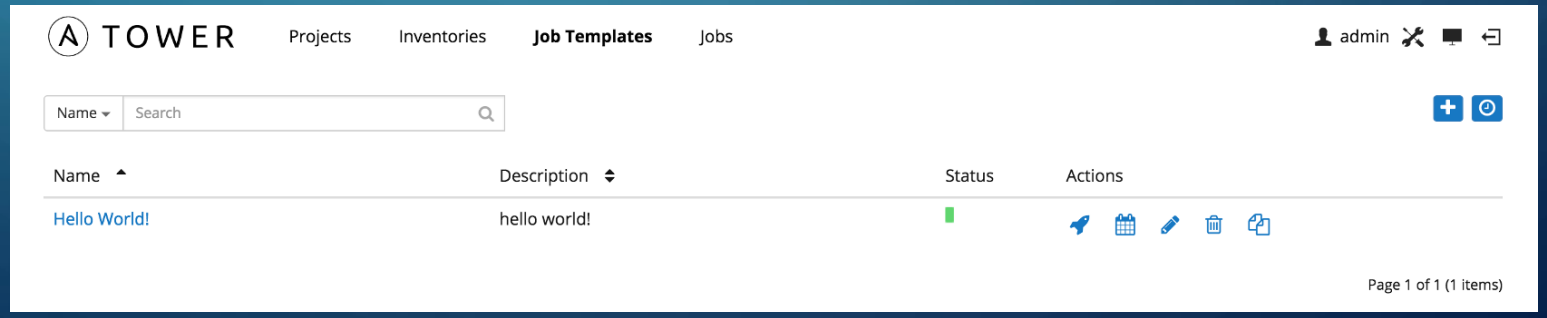

TOWER
Projects
Inventories
Job Templates
Jobs
admin
Job Templates
Create Job Templates
*Name
Description
*Job Type
Run
*Inventory
Web Servers
*Project
*Playbook
Choose a playbook
Machine Credential
Cloud Credential
Forks
0
Limit
*Verbosity
0 (Normal)
Job Tags
Extra Variables
Parse as: YAML JSON
1 ---
Prompt for Extra Variables
Enable Survey
Enable Privilege Escalation
Allow Provisioning Callbacks
Save
Reset

LANZAMIENTO DE JOBS:

Después de crear un job template que contiene todos los parámetros que normalmente se pasan al playbook de ansible, se puede iniciar un despliegue de ese job.

Para iniciar un job template, haga clic en el botón de cohete en la columna ACTIONS del job template que se lanzará. Los siguientes datos pueden ser solicitados en el lanzamiento:

- Credentials.
- Contraseña o passphrase para conectarse a los hosts de gestión remota, si se establece como ASK.
- Survey questions, si están configuradas para los job templates.
- Extra variables, si así lo solicita el job template.



PROGRAMAR UN JOB:

Ansible Tower puede programar jobs para que se ejecuten en un momento determinado o de forma repetida. Por ejemplo, puede ser que las actualizaciones disponibles deban aplicarse durante una ventana de mantenimiento a las 8 AM los martes. La página Add Schedule se puede usar para establecer una programación para ejecutar un job template.

Para programar un job, haga clic en el icono de calendario debajo de la columna Actions para el job template que se debe programar. Use el botón del icono + para agregar un nuevo horario. La página Add Schedule tiene los siguientes campos:

- Nanme: Nombre del horario
- Start Date : Fecha de inicio.
- State Time : Hora de inicio.
- Local Time Zone : Zona horaria a utilizar.
- Repeat frequency: Frecuencia de la ejecución del job programado.

Add Schedule

Options | Details

* Name

Schedule name

* Start Date mm/dd/yyyy

04/26/2016

* Start Time HH24:MM:SS

00 : 00 : 00

Local Time Zone

Asia/Kolkata

UTC Start Time

04/25/2016 18:30:00 UTC

Repeat frequency

None (run once)

Cancel | Save

Comprobando el estado del JOB:

La página JOBS para ejecución de jobs de playbooks muestra detalles de todas las tareas y eventos para esa ejecución de playbooks. La página JOBS consta de varias áreas: *Status*, *Plays*, *Tasks*, *Host Events*, *Events Summary* y *Hosts Summary*.

STATUS: El área status muestra el estado del job, que puede ser *Running*, *Pending*, *Successful* o *Failed*.

PLAYS: El área plays muestra los plays que se ejecutaron como parte del playbook. Para cada play, tower muestra la hora de inicio, el tiempo transcurrido, el Name del play y si el play tuvo éxito o no. Al hacer clic en un play específico, se filtran las Tasks y el host.

Área de Events para mostrar solo las tareas y los hosts en relación con ese play.

• **TASKS:** El área Tasks muestra las tareas ejecutadas como parte de los playbooks. Para cada tarea, además de los otros detalles, muestra un resumen del estado del host para esa tarea. El estado del host puede ser *Success*, *Changed*, *Failure*, *Unreachable* or *Skipped*.

- **HOST EVENTS:** El área de eventos de host muestra los hosts afectados por la tarea y el juego seleccionados.
- **EVENT SUMMARY:** El área Events Summary muestra un resumen de eventos para todos los hosts afectados por este playbook. Para cada host, el área Events Summary muestra el nombre del host y la cantidad de tareas completadas para ese host, ordenadas por estado.
- **HOST SUMMARY:** El Host Summary muestra un gráfico que resume el estado de todos los hosts afectados por la ejecución del playbook.

TOWER Projects Inventories Job Templates Jobs admin

Jobs > 4 - Shell Test

Refresh the page

Status Successful

Timing Started 04/26/16 16:57:23 Finished 04/26/16 16:57:25 Elapsed 00:00:01

more

Plays Play Name All Failed

Started	Elapsed	Status	Name
16:57:24	00:00:00		Hello World project

Tasks Task Name All Failed

Started	Elapsed	Status	Name	Host Status
16:57:24	00:00:00		setup	1
16:57:25	00:00:00		Echo Hello World	1

Host Events Host Name All Failed

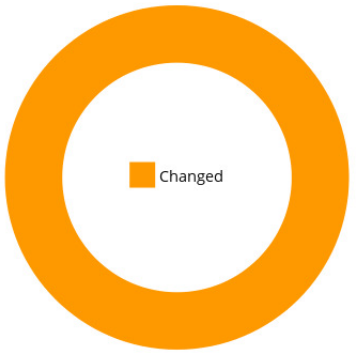
Status	Host	Item	Message
	servera.lab.example.com		

Events Summary Host Name All Failed

OK Changed Unreachable Failed

Host	Completed Tasks
servera.lab.example.com	2 1

Host Summary



13. ANSIBLE EN UN ENTORNO DEVOPS

En los últimos años, el enfoque de DevOps ha visto una creciente adopción en la empresa a medida que las organizaciones se esfuerzan por resolver el conflicto entre sus equipos de desarrollo y operativos. DevOps deriva su nombre del principio básico de que el desarrollo de software y el rendimiento de las operaciones se pueden mejorar y acelerar a través de una mejor comunicación, integración y cooperación entre los desarrolladores de software y los profesionales de operaciones de TI.

DevOps se centra principalmente en la capacidad de crear y mantener componentes esenciales con procedimientos automatizados y programáticos. Un concepto clave de DevOps es la idea de Infraestructura como código (IaC). Este concepto es un cambio de paradigma importante e innovador para los muchos administradores de sistemas que actualmente administran su infraestructura a través de la ejecución manual de comandos administrativos y la edición de archivos de configuración. Al diseñar, implementar y administrar la infraestructura como código, las configuraciones se pueden implementar y replicar de manera predecible y consistente en todo el entorno.

UTILIZANDO VAGRANT

Al probar el código para el despliegue de producción, los resultados solo son relevantes y válidos si una prueba se realiza en un entorno de desarrollo que es idéntico al entorno de producción. Esto es tan valido para el desarrollo de software como para los cambios en el código de infraestructura. La tecnología de virtualización hace que sea fácil y rentable instalar una máquina para probar el código antes del despliegue de producción. Sin embargo, el verdadero desafío es cómo construir un entorno de desarrollo en una máquina virtual para que sea una réplica del entorno de producción.

El software Vagrant supera este desafío al agilizar la creación y configuración de entornos de desarrollo virtual. Vagrant tiene su propio lenguaje de dominio específico que se utiliza para crear un conjunto de instrucciones para administrar software de virtualización como Virtualbox, KVM y VMware. También puede interactuar con software de administración de configuración como Ansible, Puppet, Salt y Chef. Vagrant simplifica el proceso de creación y administración de entornos virtualizados consistentes necesarios para el desarrollo.

Vagrant automatiza la creación de una máquina virtual, su configuración de hardware, instalación de software, configuración de sistema y recuperación de código fuente de desarrollo al permitir que todo el proceso se especifique dentro de un archivo de configuración de texto sin formato. Con Vagrant, la implementación de un entorno de desarrollo puede ser tan simple como hacer "checking out" un proyecto del control de versiones y ejecutar vagrant en la línea de comandos.

Un beneficio adicional de la gestión de Vagrant de los entornos virtualizados como código es que es muy fácil no solo crear un entorno de desarrollo virtualizado, sino también compartir el entorno idéntico con diferentes miembros del equipo. Debido a que aísla al usuario final de las complejidades de configurar y compartir entornos de desarrollo virtualizados idénticos, Vagrant es una herramienta ideal no solo para que los administradores prueben cambios en el código de la infraestructura, sino también para que los desarrolladores prueben versiones de software.

COMPONENTES DE VAGRANT:

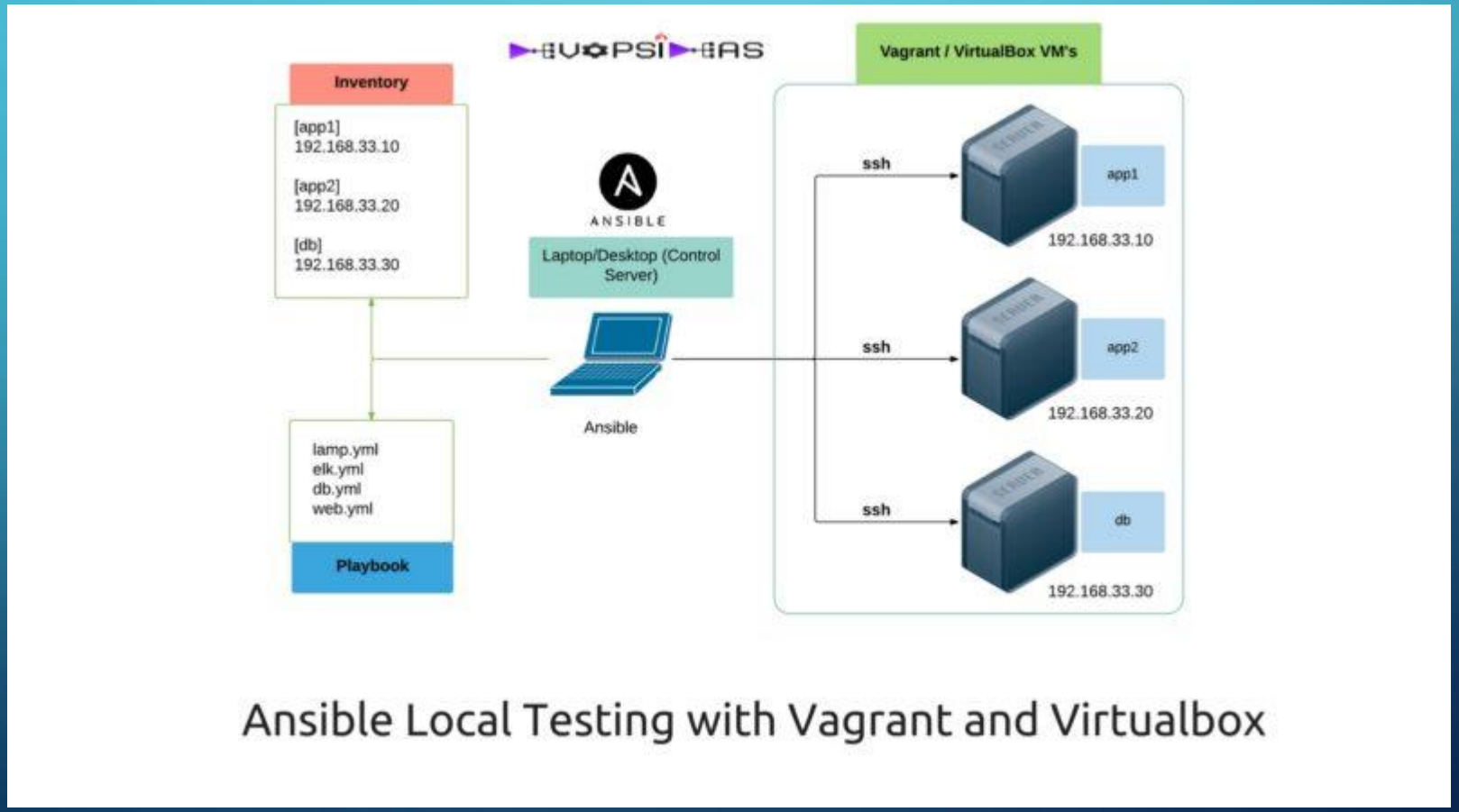
El software Vagrant se compone de los siguientes componentes principales:

VAGRANT: El software Vagrant automatiza la construcción y configuración de máquinas virtuales. Se proporciona una interfaz de línea de comandos para la gestión de proyectos Vagrant. Con el comando *vagrant* se puede crear instancias, eliminar y administrar las máquinas. Vagrant actualmente no está empaquetado con Red Hat Enterprise Linux, pero está disponible para descargarlo desde https://www.vagrantup.com .

BOX: Un box es un fichero tar que contiene una imagen de máquina virtual. Un fichero box sirve como la base de un entorno virtualizado de Vagrant y se utiliza para crear instancias de máquinas virtuales. Para una mayor flexibilidad, la imagen debe contener solo una instalación básica del sistema operativo. Esto permite que la imagen se utilice como punto de partida para crear diferentes máquinas virtuales, independientemente de los requisitos específicos de sus aplicaciones. Estos requisitos específicos de la aplicación se pueden cumplir a través de la configuración automatizada después de crear la máquina virtual.

PROVIDER: Un proveedor le permite a Vagrant interactuar con la plataforma subyacente en la que se implementa una imagen de Vagrant. Vagrant viene empaquetado con un proveedor para VirtualBox de Oracle. Los proveedores alternativos están actualmente disponibles para otras plataformas de virtualización como VMware, Hyper-V y KVM.

Vagrantfile: Vagrantfile es un fichero de texto plano que contiene las instrucciones para crear un entorno virtualizado Vagrant. Las instrucciones están escritas usando la sintaxis de Ruby. El contenido de este archivo se puede usar para establecer cómo se construirá y configurará la máquina virtual.



El Vagrantfile es un archivo de Ruby utilizado para configurar Vagrant por proyecto. La función principal de Vagrantfile es describir las máquinas virtuales necesarias para un proyecto, así como la forma de configurar y aprovisionar estas máquinas.

Los scripts Vagrantfiles están destinados a ser comprometidos directamente bajo el control de versiones. La idea es que cuando un desarrollador sincroniza su fichero vagrantfile, simplemente con lanzar el comando vagrant up obtiene un entorno virtual totalmente aprovisionado para desarrollar ese producto.

Para montar una máquina virtual con Vagrant tenemos que tener obligatoriamente un archivo llamado Vagrantfile. La sintaxis del script Vagrantfile es Ruby , pero el conocimiento del lenguaje de programación Ruby no es necesario para realizar modificaciones en el Vagrantfile, ya que es en su mayoría sólo se realizan asignaciones simples de variables. Este fichero define:

- ✓El box que servirá de plantilla de la máquina virtual.
- ✓Las características de la máquina virtual.
- ✓Conjunto de comandos que queremos que se ejecuten al crear la máquina. Suelen ser los pasos de instalación de aplicaciones y copias de datos desde carpetas compartidas en la máquina host a la máquina virtual.

Este fichero lo creamos con el comando:

```
vagrant init
```

Por defecto viene con un contenido bastante extenso a modo orientativo, pero casi todo está comentado. Veamos sólo el contenido no comentado:

```
Vagrant.configure("2") do |config|
  config.vm.box = "base"
end
```

A continuación veamos como modificar este fichero para personalizar la imagen virtual que queremos se cree automáticamente con vagrant.

Sistema operativo. En el fichero Vagrantfile anterior podemos ver que "base" es el nombre del box del que partiremos para instalar el so de la máquina virtual. Este debe existir en algunos de los portales de boxes indicados en el apartado anterior, corresponderá con el sistema operativo a instalar.

Por ejemplo si quisiéramos instalar una maquina virtual *ubuntu/xenial64* en virtualBox podríamos buscar en el portal de Boxes y llegaríamos a esta página donde nos indicarían que cambiáramos "base" por "ubuntu/xenial64" así

```
config.vm.box = "ubuntu/xenial64"
```

Redireccionamiento de puertos. Imáginemos que tenemos una maquina virtual con un servidor que escucha en el puerto 80, pero si intentamos acceder a http://localhost:80, no funcionará. Para que funcione bien tenemos que hacer un redireccionamiento de puertos, es decir, que simularemos que el puerto 8080 (es el que suelo utilizar cuando desarrollo) es el 80 de la máquina virtual. Añadimos la siguiente línea:

```
config.vm.network "forwarded_port", guest: 80, host: 8080
```

Red privada. Si en lugar de acceder por localhost, queremos hacerlo por IP, añadimos lo siguiente:

```
config.vm.network "private_network", ip: "192.168.33.10"
```

Red Publica. Configura la máquina virtual para crear un adaptador puente de red de modo que la máquina virtual pueda ser accesible desde la red pública a la que está conectado el host.

```
config.vm.network "public_network"
```

Configurar características de la máquina virtual en virtualBox. Por ejemplo para modificar la memoria asignada a la máquina indicando que sea de 8GB:

```
config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
end
```

Instalar aplicaciones en la imagen virtual. Podemos definir los comandos a ejecutarse tras la creación de la imagen para aprovisionar la imagen con las aplicaciones necesarias.

Por ejemplo si queremos instalar Docker en la imagen virtual

Podríamos añadir lo siguiente:

```
config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install apt-transport-https ca-certificates curl software-properties-common
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
    sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
    apt-get update
    apt-get install -y docker-ce

    #Añada al usuario al grupo docker.
    sudo usermod -aG docker $USER
SHELL
```

También podríamos crear un nuevo archivo de scripting con extensión .sh. Esto le indicará a Vagrant que debe hacer uso de la herramienta nativa Shell, y ejecutar todos los comandos agregados al archivo .sh para que se ejecute junto con la provisión de la máquina. Sustituiríamos todo el bloque anterior por algo similar a esto:

```
config.vm.provsion "shell", path: "bootstrap.sh"
```

El nombre del fichero no es obligatorio que sea *bootstrap.sh* puede ser el que quieras, solo debes mantener la terminación .sh para que pueda ser un fichero de scripting en linux.

```
# Define ansible provisioner
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "intranet-dev.yml"
end
```

El fichero vagrantfile quedaría así:

```
Vagrant.configure("2") do |config|

        config.vm.box = "ubuntu/xenial64"

        config.vm.network "forwarded_port", guest: 80, host: 8080
        config.vm.network "private_network", ip: "192.168.33.10"

        config.vm.network "public_network"

        config.vm.synced_folder "./backup", "/vagrant_data", create: true

        config.vm.provider "virtualbox" do |vb|
                vb.memory = "2048"
        end

        config.vm.provision "shell", inline: <<-SHELL
                apt-get update
                apt-get install apt-transport-https ca-certificates curl software-properties-common
                curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
                sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stab
le"
                apt-get update
                apt-get install -y docker-ce

                #Añada al usuario al grupo docker.
                sudo usermod -aG docker $USER
        SHELL
end
```

Configurando un entorno Vagrant:

Los entornos de vagrant están destinados a ser operados de forma aislada entre sí. Para crear un nuevo entorno Vagrant, comience creando un directorio de proyectos para el nuevo entorno. Dentro de este directorio de proyecto, cree un fichero Vagrantfile que contenga las instrucciones para implementar una nueva máquina Vagrant. El siguiente ejemplo muestra cómo los desarrolladores pueden iniciar proyectos Vagrant en sus estaciones de trabajo.

```
[root@host ~]# mkdir -p /root/vagrant/project
[root@host ~]# cd /root/vagrant/project
[root@host project]# vim Vagrantfile
```

Creando un Vagrantfile básico:

El siguiente fichero Vagrantfile crea una máquina virtual básica. La llamada al método config.vm.box especifica que la máquina virtual se clonará a partir de una imagen de cuadro llamada rhel7.1, que se puede obtener desde la ubicación especificada por la llamada al método config.vm.box_url. La llamada al método config.vm.hostname le indica a Vagrant que nombre la máquina virtual sandbox.example.com cuando se crea.

```
Vagrant.configure(2) do |config|
  config.vm.box = "rhel7.1"
  config.vm.box_url = "http://content.example.com/ansible2.0/x86_64/dvd/vagrant/rhel-
server-libvirt-7.1-1.x86_64.box"
  config.vm.hostname = "sandbox.example.com"
end
```

Manejando máquinas Vagrant:

Con un archivo Vagrantfile creado, se puede crear una instancia de la máquina Vagrant ejecutando el comando *vagrant up* desde la raíz del directorio del proyecto.

```
[root@host project]# vagrant up
Bringing machine 'default' up with 'libvirt' provider...
==> default: Box 'rhel7.1' could not be found. Attempting to find and install...
    default: Box Provider: libvirt
... Output omitted ...
==> default: Waiting for SSH to become available...
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
... Output omitted ...
==> default: Setting hostname...
==> default: Configuring and enabling network interfaces...
==> default: Rsyncing folder: /root/vagrant/project/ => /home/vagrant/sync
```

La salida de *vagrant up* muestra todas las tareas de configuración y administración de las cuales Vagrant automatiza y aísla al usuario. Estas tareas incluyen la recuperación de la imagen box, la creación de la máquina virtual, la configuración del nombre del host, la configuración de las interfaces de red y la copia de archivos desde el host a la máquina Vagrant. Si el direccionamiento IP estático no está definido en Vagrantfile, el proveedor de virtualización asigna dinámicamente una dirección IP a la máquina virtual.

Cuando se inicia la máquina Vagrant, se puede acceder a ella utilizando el comando *vagrant ssh*. Durante el despliegue de la máquina Vagrant, se genera una clave SSH en el host y luego se instala en el directorio ~/.ssh del usuario vagrant en la máquina Vagrant. El comando *vagrant ssh* usa esta clave para autenticar una sesión SSH en la máquina Vagrant como el usuario vagrant

```
[root@host project]# vagrant ssh
Last login: Wed Oct 21 14:02:44 2015 from 192.168.121.1

[vagrant@sandbox ~]$ id
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant),1001(docker)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Cuando ya no necesite una máquina Vagrant, ejecute el comando *vagrant halt* para apagar la máquina Vagrant. Otra opción es ejecutar el comando *vagrant destroy* para detener la máquina en ejecución, así como limpiar todos los recursos que se crearon cuando se implementó la máquina.

```
[vagrant@sandbox project]$ exit
logout
Connection to 192.168.121.85 closed.

[root@host project]# vagrant destroy
==> default: Removing domain...
```