Ansible

Introducción

Ansible es una herramienta de administración remota que permite permite **definir estados** (que esté instalada esta herramienta, o que exista aquella interfaz de red, ...) y llevar a los equipos hasta dicho estado realizando las operaciones que sean necesarias. Para ello únicamente debe instalarse en la máquina que controla a las demás, ya que accede a los equipos remotos utilizando simplemente **SSH** (esto implica, eso sí, que Ansible deberá autenticarse en cada uno de los equipos remotos a administrar mediante un determinado usuario existente en ellos y, preferiblemente, ese usuario deberá tener privilegios de "root" vía *sudo* para poder realizar la mayoría de operaciones importantes del sistema).

NOTA: Ansible realiza el acceso por SSH usando una librería Python llamada Paramiko (http://www.paramiko.org)

Simplificando mucho, podríamos decir que Ansible actúa como un cliente SSH masivo capaz de conectarse en paralelo a múltiples servidores SSH (los hosts a administrar) y realizar en cada uno de ellos la operación deseada. Para ello, Ansible ofrece diferentes módulos (la lista completa se puede encontrar aquí: http://docs.ansible.com/ansible/modules_by_category.html), los cuales permiten realizar diferentes operaciones predefinidas de forma sencilla sin la necesidad de tener que escribir complejos comandos o shells scripts. Ejemplos de módulos son, por ejemplo: "apt" (para instalar paquetes en los hosts remotos), "service" (para administrar servicios en los hosts remotos), "file" (para cambiar los atributos de ficheros en los hosts remotos), etc, etc. La mayoría de comandos Unix tienen asociado un módulo Ansible equivalente.

Hay que saber, por otro lado, que tras conectar mediante SSH en un host remoto, Ansible abre en él una subshell (como root si la operación lo requiere) para ejecutar allí dentro un intérprete de **Python** que se encargará de ejecutar un determinado script que representa la implementación real del módulo indicado. Esto quiere decir que, además de tener en marcha un servidor SSH en todas los hosts remotos a administrar, también será necesario tener instalado un intérprete de Python.

Por otro lado, un aspecto muy importante de Ansible es que todas las operaciones realizadas en los hosts remotos son **idempotentes**. Esto significa que no hay que pensar en condiciones: se definen las operaciones a realizar y únicamente producirán efecto aquellas que sean necesarias (por no estar el host remoto de acuerdo al estado especificado) dejándose sin tocar los hosts que ya estén en dicho estado.

Un concepto importante en Ansible son los "playbooks". Un "**playbook**" no es más que un fichero de texto donde se especifican (mediante una sintaxis llamada YAML, http://yaml.org) todos los detalles necesarios para que Ansible pueda, en el momento de repasar dicho "playbook", saber qué determinado conjunto de operaciones deberá realizar (mediante el uso de qué módulos determinados) sobre qué determinado conjunto de hosts remotos. Ansible will execute the playbook in sequence and ensure the state of each command is as desired before moving onto the next (this is what makes Ansible idempotent: if you cancel the playbook execution partway through and restart it later, only the commands that haven’t completed previously will execute.

Playbooks allow you to create truly complex instructions, but if you’re not careful they can quickly become unwieldy, which brings us onto another concept: "**roles**". Roles add organisation to playbooks: they allow you to split your complex build instructions into smaller reusable chunks, very much like a function in programming terms. This makes it possible to share your roles across different playbooks, without duplicating code. For example, you may have a role for installing Nginx and configuring sensible defaults, which can be used across multiple hosting environments.

Para instalar la versión más moderna de Ansible en nuestra máquina controladora (suponiendo que es un Ubuntu), podemos ejecutar los siguientes comandos:

sudo apt-add-repository ppa:ansible/ansible -y
sudo apt update
sudo apt install ansible

NOTA: En Fedora la versión que hay en los repositorios oficiales suele estar bastante actualizada, así que bastará con hacer *sudo dnf install ansible*

Ansible no és l'única eina d'administració remota de sistemes. Altres alternatives són:
CFEngine (https://cfengine.com)
Chef (https://www.chef.io/chef)
Puppet (https://puppet.com)
Salt (https://saltstack.com)
Foreman (https://www.theforeman.org). Aquest inclou a més un servidor PXE integrat

Configuración general

El archivo de configuración general de Ansible es /etc/ansible/ansible.cfg. No será necesario editarlo en la mayoría de ocasiones porque los valores predefinidos ya son suficientes la mayoría de veces. No obstante, a continuación se nombran los datos que tal vez haya que editar en algún momento:

*Directiva *forks* (dentro de la sección [defaults]) : Indica el número máximo de "víctimas" a las que Ansible conectará a la vez para ejecutar una determinada tarea. Si el número total de "víctimas" es mayor que ese número, cuando una de las "víctimas" elegida inicialmente haya finalizado la ejecución de esa tarea, Ansible pasará a conectarse a otra máquina nueva para ejecutar dicha tarea allí, y así sucesivamente, siempre manteniendo el número de conexiones paralelas como mucho en el número dado por esta directiva. A mayor número, menor tiempo empleado en finalizar todo, obviamente, pero también más estrés para la máquina controladora.

*Directiva *host_key_checking* (dentro de la sección [defaults]) : Every host that runs an SSH server has an associated "host key". This key acts like a signature that uniquely identifies the host; "host keys" exist to prevent man-in-the-middle attacks because allow you to check that the server that claims to be your desired server really it is. This means that you need to have the host key (a copy of what the signature should look like) before you try to connect to the host. This step must be done only once but it must be done before Ansible connect first time to the server. The simpler manner of having the host key in Ansible's machine is connecting with regular ssh client and answering "yes" to shown question, that's all. Another "solution" would be disabling the requirement of having the host keys (but be careful, it's dangerous!); this can be done assigning the value *false* to the *host_key_checking* directive (default value is *true*).

*Directiva *retry_files_enabled* (dentro de la sección [defaults]) : When a playbook fails, by default a .retry file is created (with the same name as the playbook in question) inside the same folder where that playbook resides. This .retry file contains the IPs of offending "victim" machines, one by line. You can disable this feature by setting *retry_files_enabled* to false. If it is true, though, you could change instead the location of the .retry file by setting the new .retry file's absolute path as the value of *retry_files_save_path* option (if this path contained an inexistent folder, it would be created).

NOTA: És possible tenir una configuració d'Ansible personalitzada que sobreescrigui la configuració general definida a l'arxiu /etc/ansible/ansible.cfg. Concretament, es pot utilitzar l'arxiu ~/.ansible.cfg per tal de tenir una configuració particular per l'usuari concret que executi Ansible i/o també es pot utilitzar un arxiu /ruta/qualsevol/ansible.cfg per tal d'establir una configuració particular per només el cas en què Ansible s'executi dins de la ruta indicada (/ruta/qualsevol), sobreescrivint en aquest cas a tots els arxius anteriors. En qualsevol cas, sempre es podrà indicar un arxiu de configuració específic (que tindrà prioritat sobre tota la resta) indicant la seva ruta completa com a valor de la variable d'entorn ANSIBLE_CONFIG.

NOTA: A la pàgina https://docs.ansible.com/ansible/latest/reference_appendices/config.html#common-options podeu trobar la llista de totes les directives de configuració, juntament amb la seva explicació, el seu valor per defecte i la seva ubicació dins de l'arxiu /etc/ansible/ansible.cfg . Allà també apareixen llistades les possibles variables d'entorn que es podrien fer servir per sobreescriure en un moment puntual la configuració indicada al fitxer.

Inventario

La lista de hosts reconocidos por Ansible se ha de indicar en el archivo /etc/ansible/hosts. A continuación se muestra un ejemplo de contenido autoexplicativo de este archivo:

[grupo1]
192.168.0.111
192.168.0.123
#Para indicar un rango de IPs
192.168.1.[01:50]
...
#Si la asociación nombre<-> IP ya está definida en el archivo /etc/hosts de la máquina donde se #ejecutará Ansible (o es resoluble mediante DNS), en vez de IPs se puede indicar directamente los #nombres de la máquinas víctimas, así:
nombreMaquina

[grupo2]
#Para indicar que el servidor SSH de ese host escucha en otro puerto diferente del puerto por defecto #(que es 22 y está indicado en ansible.cfg) se puede usar la variable *ansible_port* así:
192.168.0.222 ansible_port=1234
#Para indicar el usuari a usar por defecto contra el servidor SSH (que por defecto es root, y está #indicado en ansible.cfg) se puede usar la variable *ansible_user* así:
192.168.0.231 ansible_user=pepa
#Para realizar una conexión a la máquina local y, por tanto, no usar SSH (por defecto el valor de la #variable *ansible_connection* es "ssh". Otros valores posibles son "docker", "podman", "lxd", ...)
127.0.0.1 ansible_connection=local

#Si queremos indicar el mismo valor para determinadas variables sin tenerlas que escribir host a #host, se pueden asignar a grupos enteros así (también se podría hacer [all:vars])
[grupo2:vars]
ansible_port=2222
ansible_user=pepe
#Si queremos generar grupos que contengan otros grupos, se puede hacer así
[grupoPadre:children]
grupo1
grupo2

NOTA: Existe siempre un grupo predefinido llamado "all" que contiene todos los grupos indicados en el inventario. El orden de sobreescritura de una variable, si se escribe la misma en diferentes lugares del inventario es el siguiente: all → grupo_padre → grupo_hijo → victima_individual

NOTA: A la hora de ejecutar Ansible será posible indicar como "víctima" solamente una IP concreta pero en cualquier caso esa IP debe aparecer en algún lugar del archivo /etc/ansible/hosts : si no está Ansible no la reconocerá. También se pueden especificar varias IPs concretas mediante la notación IP1:IP2:... o también mediante el uso del comodín "*". Para más información consultar https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html

NOTA: Se podría utilizar otro fichero diferente de /etc/ansible/hosts si se indica su ruta en la directiva *inventory* del fichero *ansible.cfg* (bajo la sentencia *[defaults]*). O mediante la variable de entorno *ANSIBLE_INVENTORY=/ruta/fichero* (otro nombre alternativo a dicha variable es *ANSIBLE_HOSTS*). También se puede indicar mediante el parámetro *-i* del comando *ansible* en el momento de lanzar la/s operación/es

En el apartado "List of behavioral inventory parameters de la página http://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html aparecen listadas todas las posibles variables "ansible_*" que se pueden indicar en el inventario. Entre las más interesantes, podemos encontrar:

ansible_ssh_pass : Contrasenya SSH a usar. Si no es vol escriure en texte pla, caldrà usar un "Ansible Vault", que ja veurem. Una altra opció és no indicar aquesta opció i deixar llavors que Ansible la demani interactivament en el moment d'executar-lo

ansible_become : Si val *yes* es permet fer servir *sudo/su* a les màquines víctimes corresponents si calgués

ansible_become_user : Indica l'usuari en el què es convertirà Ansible en executar sudo/su. Per defecte és "root"

ansible_become_pass: Contrasenya sudo/su a usar. Si no es vol escriure en texte pla, caldrà usar un "Ansible Vault". Una altra opció és no indicar aquesta opció i deixar llavors que Ansible la demani interactivament en el moment d'executar-lo

ansible_python_interpreter : Indica la ruta absoluta de l'intèrpret Python que Ansible farà servir a les víctimes. Aquesta opció és útil quan la màquina víctima no poseeix l'intèrpret Python a la ruta per defecte, que és "/usr/bin/python" (per exemple, a Fedora 29 és /usr/bin/python3)

Todas estas opciones ansible_* se pasarán luego como variables a cada playbook que se ejecute en las víctimas en cuestión. De hecho, dentro de alguna sección "[*xxx*:vars]" del inventario se podría indicar cualquier otro tipo de opción personalizada que quisiéramos pasar como variables a dichos playbooks (más adelante veremos la utilidad de esto). No obstante, para no tener el inventario demasiado atiborrado, en vez de indicar las opciones ansible_* y/o las variables personalizadas directamente dentro de él, lo que se suele hacer es indicar el nombre y valor de las distintas variables (con esta sintaxis: *nombre: valor*, y una variable por línia) en un fichero YAML con extensión ".yml" o ".yaml" llamado o bien "/etc/ansible/group_vars/nombreGrupoInvent.yml" o bien "/etc/ansible/host_vars/nombreMaqInvent.yml", según si las variables en cuestión queremos que afecten a un grupo o a una "víctima" en particular (las carpetas "group_vars" y "host_vars" habrá que crearlas si no existen).

NOTA: The "group_vars" and "host_vars"directories can exist in the playbook directory OR the inventory directory. If both paths exist, variables in the playbook directory will override variables set in the inventory directory.

Uso directo de Ansible (sin "playbooks"). Ejemplos de uso de módulos básicos

Las operaciones se pueden lanzar directament mediante el comando ansible de la siguiente manera:

ansible nombreGrupo -m modulo [-a "param1=valor1 param2=valor2 ..."] -u usuario -k [-b -K]

donde:

**nombreGrupo* representa el nombre de un grupo indicado en /etc/ansible/hosts (podría ser también una IP individual existente en ese archivo o la palabra "all")

**-m modulo* representa el módulo concreto de Ansible a utilizar

**-a "..."*: Si el módulo a utilizar necesitara parámetros para funcionar (algunos los requieren, otros no), se han de indicar como una lista de nombre<->valor separados por espacios

**-u usuario* representa el usuario de la "víctima" con el que se entrará en ella a realizar la operación. No es pot indicar usuaris diferents per "víctimes" diferents: totes les "víctimes" hauran de tenir el mateix nom d'usuari per poder ser controlades des d'Ansible.

**-k* : Hace que Ansible pida interactivamente la contraseña del usuario anterior (este parámetro es imprescindible si no hemos activado todavía la autentiación por claves o si no hemos indicado la contraseña en el inventario; si sí hemos hecho alguna de estas dos cosas, entonces no se ha de escribir).

**-b :* Si la operación a realizar necesita ejecutarse con permisos de administrador, este parámetro indica que se ejecute el comando *sudo* automático. Por tanto, será imprescindible escribirlo para la mayoría de operaciones del sistema. Obviamente, si el usuario indicado no pudiera utilizar el comando *sudo* en la máquina "víctima", este parámetro no funcionará.

*-*K* : Si en realizarse el *sudo*, este está configurado para solicitar contraseña interactivamente (su configuración por defecto es así), este parámetro permite introducirla (a no ser que la hayamos indicado explícitamente en el inventario).

Otros parámetros interesantes son:

*-*f nº* : Se puede indicar como parámetro también el nivel de paralelismo que se desea al lanzar las operaciones, sobreescribiendo así el valor de la directiva general *forks*

*-*t /ruta/carpeta* : Indica la carpeta donde se guardará por cada máquina "víctima" probada un fichero (cuyo nombre será la IP de la "víctima" correspondiente) conteniendo lo mismo que se muestra en pantalla relativo a esa "víctima" en concreto tras la ejecución del comando ansible indicado.

A continuación veremos algunos ejemplos de módulos básicos:

**ping* : Ejecuta el comando ping a las "víctimas" indicadas: *ansible all -m ping -u usuari -k*

**setup* : Before running any Tasks, Ansible will automatically gather information about the system it's provisioning. This informacion is returned to the controller machine with the form of a a JSON object. The items inside this object are called "facts", and include a wide array of system information such as the number of CPU cores, available ipv4 and ipv6 networks, mounted disks, Linux distribution and more. Ansible facts have the form of variables starting all with "ansible_..." and are globally available for use any place variables can be used. Examples are: *{{ansible_hostname}}, {{ansible_distribution_release}}, {{ansible_devices.sda.model}}, {{ansible_processor_cores}}, {{ansible_processor_count}},* etc. Ansible uses these facts for various housekeeping purposes but we can keep and see this information thansk to the "setup" module. This module shows a list of all of the facts that are available about a machine: *ansible all -m setup -u usuari -k* Facts can be filtered by using the *-a "filter=ansible_*_mb"* parameter, for instance.

**shell* : Ejecuta en las "víctimas" indicadas un comando arbitrario especificado en el parámetro *-a*: *ansible all -m shell -a "free -m"*... Otro parámetro que ofrece este módulo interesante a conocer es *chdir=/ruta/carpeta.* Por otro lado son interesantes también los módulos *script* (https://docs.ansible.com/ansible/latest/modules/script_module.html) y *command* (https://docs.ansible.com/ansible/latest/modules/command_module.html). También el módulo *expect* (https://docs.ansible.com/ansible/latest/modules/expect_module.html)

**apt o dnf* : Instala/Desintala en las "víctimas" Ubuntu/Fedora indicadas el programa especificado en el parámetro *-a : ansible all -m apt -a "name=prog1 state=present* " ... , donde *name* indica el paquete (o paquetes, separados por comas) a instalar (también puede ser la ruta absoluta de un fichero ".deb"/".rpm"!) y *state* indica el estado final deseado (también puede ser "latest" o "absent" para desinstalarlo). También podemos actualizar todo el sistema simplemente indicando el parámetro "upgrade=dist" (si usamos el módulo *apt*) o los parámetros "name='*' state=latest" (si usamos el módulo *dnf*)

**service* : Controla en las "víctimas" indicadas un servicio especificado en el parámetro *-a*, así: *ansible all -m service -a "name=serv1 state=started"*... Además de parar/iniciar también puede habilitar/deshabilitar, entre otras opciones. Este módulo es genérico para diferentes gestores de servicios existentes (Systemd, SysV, OpenRC, etc) pero si sabemos que nuestras máquinas "víctimas" usan todas Systemd, podemos usar como alternativa el módulo específico *systemd* (https://docs.ansible.com/ansible/latest/modules/systemd_module.html), que dispone de opciones muy similares

*copy : Copia ficheros desde la máquina controladora hacia las "víctimas" indicadas. En el parámetro *-a* se ha de indicar como mínimo qué fichero/s se quieren copiar (parámetro *src*) y a qué carpeta de destino (parámetro *dest*), indicando opcionalmente el propietario y permisos que tendrán los ficheros copiados: *ansible all -m copy -a "src=/ruta/fich dest=/etc/nuevonombrefich owner=root group=root mode=0640" ...* En vez de *src* se puede usar el parámetro *content="algo"* para indicar un determinado contenido textual a grabar en el fichero destino (similar a ejecutar *echo "algo" > fichero.dest* en la máquina "víctima").

> NOTA: Local path to a file to copy to the remote server can be absolute or relative. This path can be a directory too; if so, it is copied recursively. In this case, if path ends with "/", only inside contents of that directory are copied to destination. Otherwise, if it does not end with "/", the directory itself with all contents is copied.

*file : Este módulo permite realizar diversas acciones sobre un determinado fichero remoto de las "víctimas" indicadas (cuya ruta se especifica con el parámetro *path)* según el valor del parámetro *state*. Por ejemplo:

-a "path=/etc/fstab state=absent" (borra fichero)
-a "path=/etc/fstab state=touch" (crea fichero vacío)
-a "path=/etc/fstab.lnk src=/etc/fstab state=link" (crea link)
-a "path=/var/www/html/public mode=775 state=directory owner=ana group=ana" (crea carpeta)
-a "path=/var/www/html mode=775 state=directory owner=ana group=ana recurse=yes" (cambia permisos carpeta)

*user : Administra usuarios en las "víctimas". Dispone de varios parámetros importantes como *name* (el nombre del usuario), *password* (la contraseña encriptada...para encriptarla deberemos ejecutar primero el comando *python3 -c "import crypt,getpass; print(crypt.crypt('contrasenya', crypt.mksalt(crypt.METHOD_SHA512)))"* o bien, en sistemas Debian, *echo 'contrasenya' | mkpasswd –method=sha-512*), *groups* (para indicar los grupos donde se añadirá el usuario), *append* (para indicar si los grupos indicados con el parámetro anterior son los únicos -si vale *no-* o se añadirán a los ya existentes -si vale *yes-*), *shell* (para indicar el shell por defecto del usuario), etc. Si quisiéramos eliminar un usuario, deberíamos utilizar simplemente los parámetros *name* y *state=absent* (y opcionamente, el parámetro *remove=yes* para borrar también la carpeta personal del usuario a eliminar). Ver más opciones en la documentación oficial

> NOTA: Atenció! Si a la contrasenya encriptada obtinguda de la comanda python3 indicada al paràgraf anterior apareixen símbols "$", aquests s'hauran d'escapar (és a dir, afegir una "\" al davant) en el moment d'escriure-la com a valor del paràmetre "password" del mòdul "user". Si no la contrasenya no s'assignarà correctament. És a dir, si representa que la contrasenya encriptada és 6qwb23$2ghw , el paràmetre "password" s'haurà d'escriure de la següent manera: password=\$6\$qwb23\$2ghw

Uso de "playbooks"

Para realizar más de una tarea de forma coordinada, se han de crear "playbooks", los cuales son simplemente archivos de texto escritos en YAML (y, por ello, con extensión *.yml) que contienen una lista de ítems que definen las operaciones a realizar, dónde y por quién. Dentro de cada "playbook" hay uno o más grupos de hosts donde las operaciones deseadas se ejecutarán. Cada uno de estos grupos se llaman "play" y tienen la forma de elemento de un array YAML. Cada "play" debería tener los siguientes ítems:

*name : Nombre del play (ha de ser descriptivo sobre las tareas que realizará)

*hosts : Nombre del grupo de hosts (definido en el inventario) al que se le aplicarán las operaciones

*remote_user : Usuario de la "víctima" con el que se entrará en ella a realizar las operaciones. Si no se indica, se deberá seguir utilizando el parámetro -u en el terminal a la hora de ejecutar el "playbook" (mediante el comando *ansible-playbook*, como ahora veremos). Igualmente, si usamos autenticación por contraseña (o no la hemos indicado en el inventario), será necesario indicar también el parámetro *-k* en el terminal porque no existe ninguna directiva para indicar la contraseña SSH dentro de un playbook.

become: Si vale *yes*, es equivalente a indicar el parámetro -b del comando *ansible* . Si no hemos indicado la contraseña sudo en el inventario (o no hemos configurado previamente sudo para no pedir contraseña) deberemos indicar obligatoriamente el parámetro *-K* a mano a la hora de ejecutar el "playbook" (mediante el comando *ansible-playbook*, como ahora veremos).

**vars* : Opcional. Array que representa el conjunto de variables cuyos valores podrán utilizar las operaciones definidas a lo largo del "playbook". Para definir el array la sintaxis es:

```
vars:
    nombreVar1:  valor1
    nombreVar2:  valor2
```

Para utilizar el valor de una variable determinada en cualquier sitio posterior del "playbook", la sintaxis es: *{{ nombreVar1 }}*

> NOTA: Como ya sabemos, las variables que se hayan definido en un inventario o fichero "host_vars" equivalente se podrán utilizar en cualquier lugar de un playbook directamente

**gather_facts* : Opcional (por defecto vale *yes*). Si vale *no*, Ansible no realizará una recogida previa de metadatos de las máquinas víctimas, los cuales contienen información sobre su CPU, su RAM, sus discos duros, sus interfícies y configuración de red, etc y pueden ser útiles -o no- para las tareas a realizar. Estos metadatos son los llamados "facts" y es la misma información que obtiene el módulo *setup* explícitamente.

> NOTA: El valor por defecto de la directiva "gather_facts" es "yes" porque en el fichero "/etc/ansible/ansible.cfg" existe la directiva "gathering" con el valor "implicit". Si esta directiva valiera "explicit", el valor por defecto sería "no" y deberíamos explicitar la línia "gather_facts: yes" en los playbooks donde quisiéramos obtener los "facts".

**tasks* : Array YAML que representa el conjunto de operaciones a realizar EN ORDEN (a no ser que se use el ítem *order* con un valor diferente del por defecto, que es *"sorted"*...ya lo veremos). Cada una de ellas contendrá a su vez los siguientes ítems básicamente:

```
*name: Descripción de la tarea
*nombreModulo : param1=valor1 param2=valor2 ...
*ignore_errors: yes
*become: yes
```

> NOTA: Las líneas *ignore_errors* y *become* dentro de una task son opcionales. La primera por defecto vale *no*; en ese caso, la ejecución del "playbook" se para para un determinado host remoto al detectar algún error en la realización de una tarea en ese host. La segunda es útil si no se desea utilizar la opción homónima para todo el "playbook" porque solo se necesita para una tarea en concreto.

> NOTA: Per tenir un control molt més detallat sobre com s'ha de comportar Ansible en trobar errors durant l'execució d'un playbook que el que ofereix l'opció *ignore_errors*, hi ha diferents solucions (*failed_when, changed_when, meta:clear_host_errors, any_errors_fatal:true, max_fail_percentage* ...); totes elles es poden consultar amb exemples a https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

Por ejemplo, un "playbook" sencillo (con un solo "play" que realiza tres tareas) podría ser este:

```
- name: Hacer ping, instalar gnuchess e iniciar Apache2 en Fedora para unGrupoDelInventario
  hosts: unGrupoDelInventario
  remote_user: pepito
  tasks:
      - name: Hacer ping
        ping:
      - name: Instalar gnuchess en Fedora
        dnf: name=gnuchess state=latest
        become: yes
      - name: Iniciar Apache2 en Fedora
        systemd: name=httpd2 state=started
        become: yes
```

Una vez definido el "playbook", para ejecutarlo se deberá escribir el siguiente comando:

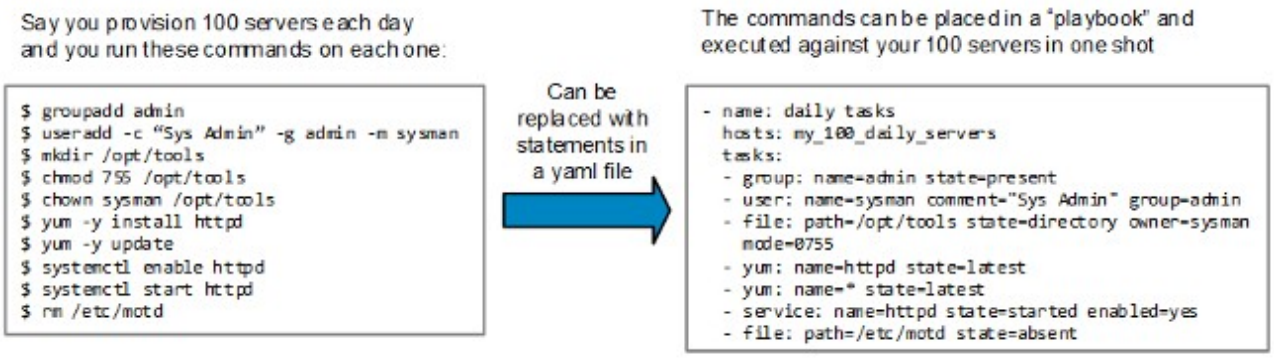
ansible-playbook nombrePlaybook.yml [-k][-K]

NOTA: Es posible pasar valores desde el terminal a variables definidas en el array "vars" del playbook indicando el parámetro *-e "variable1=valor1 variable2=valor2 ..."* en el comando *ansible-playbook* y definiendo el array "vars" así:

```
vars:
    nombreVarEnPlaybook: "{{ nombreVarPasadoPorTerminal }}"
    nombreOtraVarEnPlaybook: " {{ nombreOtraVarPasadoPorTerminal }}"
    nombreOtraVarMásEnPlaybook : unValorExplícito
```

También se pueden especificar valores de variables en ficheros exclusivos aparte, con la misma sintaxis que el contenido del array "vars", y utilizar esos valores en un "playbook" indicando la ruta del/los fichero/s en cuestión como valor/es de un ítem general llamado *vars_files* o, de forma similar, mediante el módulo *include_vars*

Un ejemplo gráfico de la utilidad de los "playbooks" es el siguiente diagrama:



Truc per fer que *sudo* no pregunti contrasenya (i així no haver de fer servir el paràmetre -K)

Es tractaria de modificar la seva configuració a l'arxiu /etc/sudoers. Concretament, si l'usuari que fem servir per entrar amb l'Ansible a les víctimes ja pot fer *sudo* (només que introduïnt la contrasenya), l'únic que haurem de fer serà modificar la següent línia...:

%sudo ALL=(ALL:ALL) ALL

...per

%sudo ALL=(ALL:ALL) ***NOPASSWD:*** *ALL*

NOTA: A Fedora, aquesta línia no existeix però n'hi ha una altra d'equivalent que comença per *%wheel*. Caldrà fer el mateix canvi

Si l'usuari que fem servir per entrar amb l'Ansible a les víctimes no pot fer *sudo*, haurem de fer el mateix pas anterior però, a més, haurem d'executar la següent comanda: *sudo usermod -a -G sudo nomusuari* (a Fedora seria *sudo usermod -a -G wheel nomusuari*)

Per conèixer altres vies més sofisticades per obtenir el mateix objectiu, es pot consultar una explicació detallada aquí: http://www.ducea.com/2006/06/18/linux-tips-password-usage-in-sudo-passwd-nopasswd/

Truc per fer que SSH no pregunti contrasenya fent servir autenticació per claus (i així no haver de fer servir el paràmetre -k)

1.-Generate the SSH key on the master node being the same user that will execute ansible command:

ssh-keygen -t rsa

2.-Copy your recently generated public key (stored in ~/.ssh folder with the name of id_rsa.pub) to all the "victims", connecting to them with the same user Ansible will use. This can be achieved in several ways (via e-mail, pendrive, executing *scp* command...or using a specific command for this: ssh-copy-id):

ssh-copy-id usuari@ipVictima1
ssh-copy-id usuari@ipVictima2
...

NOTA: Concretamente, lo que hace el comando anterior es añadir el contenido de id_rsa.pub al fichero /home/usuari/.ssh/authorized_keys de cada "víctima" (el cual, por cierto, ha de tener permisos 600 para que funcione)

3.-Opcional. Para deshabilitar el acceso por contraseña (y, por tanto, conseguir así que solo se pueda acceder a las máquinas "víctimas" solo si se intenta acceder desde la máquina controladora -y solamente siendo el usuario "usuari"-), en la configuración del servidor SSH de la máquina imagen (fichero /etc/ssh/sshd_config) hay que poner "no" en vez de "yes" en la línea PasswordAuthentication (es decir, dejarla así: ***PasswordAuthentication no***) y ya está; el resto de líneas relevantes ya están bien como están. (*PermitRootLogin prohibit-password*, *PubKeyAuthentication yes* y *RSAAuthentication yes*) . Una vez hecho esto, hay que reiniciar el servicio SSH

EXERCICIS:

0.-a) Crea (o reutilitza) tres màquines virtuals de tipus Server amb la tarja de xarxa en mode "adaptador pont", amb 1GB de RAM i que tinguin un usuari amb el mateix nom a totes tres que pugui fer *sudo*. En un d'aquestes màquines (l'anomenarem "A") hauràs d'instal.lar Ansible i en les altres dues (que anomenarem "B" i "C" i seran administrades per la primera) hauràs d'instal.lar els paquets "openssh-server" i "python".

b) Edita la llista de hosts remots reconeguts per l'Ansible (el que es diu el seu "inventari") per tal d'afegir un nou grup anomenat *[victimes]* que inclogui les IPs de "B" i "C".

c) Modifica la configuració general de l'Ansible (arxiu /etc/ansible/ansible.cfg) per tal de no realitzar la comprovació de hosts en el moment de connectar-se a màquines remotes.

> NOTA: Si aquest darrer pas no es realitza, hi hauria una altra manera d'evitar l'error que Ansible mostra però és molt més pesat: consistiria en entrar prèviament una a una a totes les màquines remotes amb el client SSH estàndar per tal de respondre "yes" a la pregunta que sempre apareix la primera vegada que hom es conecta a un host remot

1.-a) Utilitza el mòdul "apt" (o "dnf") d'Ansible amb la comanda *ansible* per tal d'instal.lar a aquestes dues màquines a la vegada, mitjançant l'usuari comú, els paquets "curl" i "nmap". Comprova seguidament que hi sigui.

b) Utilitza el mòdul "apt" (o "dnf") d'Ansible amb la comanda *ansible* per desinstal.lar a les màquines "B" i "C" només el paquet "curl". Comprova seguidament que ja no hi sigui.

c) Utilitza el mòdul "apt" (o "dnf") d'Ansible amb la comanda *ansible* per tal d'actualitzar tots els paquets però només de la màquina "B"

2.-a) Utilitza el mòdul "user" d'Ansible amb la comanda *ansible* per tal de crear a les màquines "B" i "C" un nou usuari anomenat "pepe" amb contrasenya "1234" que pertanyi al grup "adm" i que tingui el shell /bin/bash. Comprova seguidament que hi sigui (ho pots provar fent *su -l pepe* a les màquines administrades i després *id*)

b) Utilitza el mòdul "user" d'Ansible amb la comanda *ansible* per afegir aquest usuari "pepe", un cop creat, al grup "sudo". Comprova seguidament que hi pertanyi (a més de la resta de grups als que ja pertanyia)

c) Utilitza el mòdul "user" d'Ansible amb la comanda *ansible* per eliminar aquest usuari "pepe" de totes les màquines, incloent l'esborrament de la seva carpeta personal

3.-a) Modifica els valors de "ansible_ssh_pass" i "ansible_become_pass" de l'arxiu de configuració de l'Ansible per tal de què ni SSH ni *sudo* et demanin cap contrasenya (en tenir-les indicades allà). Seguidament utilitza la comanda *ansible* per, per exemple, realitzar un ping a les màquines "B" i "C" (mitjançant el mòdul "ping") i comprovar que no necessites introduïr cap contrasenya.

b) Desfés el que has realitzat a l'apartat anterior. Segueix ara les passes indicades a l'últim apartat de la teoria per aconseguir que el servidor SSH de les màquines "B" i "C" no demani contrasenya. Seguidament utilitza la comanda *ansible* per, per exemple, realitzar un ping a les màquines "B" i "C" (mitjançant el mòdul "ping") i comprovar que es fa de forma totalment desatesa.

A pesar d'haver realitzar el pas anterior, si haguéssim d'executar algun mòdul que necessités "escalar privilegis" amb *sudo* hauríem de seguir escrivint una contrasenya (en aquest cas la del *sudo*). Per evitar això podries seguir les passes manuals indicades a la teoria però, per no haver de repetir els mateixos pasos una vegada rera l'altra en cada màquina "víctima", ho farem d'una forma alternativa, que és aconseguir el mateix

efecte (és a dir, modificar una línia determinada de l'arxiu /etc/sudoers) però fent servir precisament la comanda *ansible*, que per això està: per automatitzar tasques a múltiples màquines remotes (de fet, això també ho podríem haver aplicat a l'apartat anterior). Concretament:

c) Consulta la documentació oficial del mòdul "lineinfile" i digues per a què serveix aquest mòdul

cII) Executa aquesta comanda: *ansible victimes -m lineinfile -a "path=/etc/sudoers state=present regexp='^%sudo' line='%sudo ALL=(ALL) NOPASSWD: ALL' validate='visudo -cf %s'" -u usuari -b -K* ¿Què fa aquesta comanda a les màquines "víctimes"?

d) Un cop realitzat l'apartat anterior, utilitza ara el mòdul "shell" d'Ansible per executar la comanda *sudo fdisk -l* a les màquines "B" i "C": hauràs de veure que ara no necessitaràs introduïr cap contrasenya.

4.-a) Utilitza el mòdul "copy" d'Ansible amb la comanda *ansible* per copiar l'arxiu /etc/bash.bashrc de la màquina "A" a les màquines "B" i "C"

b) Utilitza el mòdul "file" d'Ansible amb la comanda *ansible* per crear a les màquines "B" i "C" la carpeta "/home/usuari/prova" (amb permisos 755 i propietari "usuari"), crear l'arxiu buit "/home/usuari/prova/hola.txt" (amb permisos 644 i propietari "usuari") i crear un enllaç que apunti a aquest arxiu anomenat "/home/usuari/enllaç_a_hola.txt". Després de confirmar que tots aquests elements existeixin, utilitza de nou el mòdul "file" però ara per esborrar-los tots.

c) Utilitza el mòdul "setup" d'Ansible amb la comanda *ansible* i, amb l'ajuda de *grep, cut,...* escriu un petit shell script que mostri <u>només</u> la quantitat total de memòria RAM que tenen les màquines "B" i "C".

d) Utilitza el mòdul "systemd" (o "service") d'Ansible amb la comanda *ansible* per tal d'aturar i també deshabilitar el servei "ufw" a les màquines "B" i "C".

e) ¿Què fa la comanda *ansible victimes -m shell -a "ls -l /etc | grep '^d'| wc -l" -k -b -K -u usuari* ?

5.-a) Consulta la documentació oficial del mòdul "cron" i utilitza aquest mòdul per definir una tasca programada que apagui les màquines a administrar (en aquest cas, "B" i "C") cada dia a aquesta hora.

NOTA: És obligatori que indiquis els paràmetres "cron_file=/etc/crontab" i "user=root"

b) Consulta la documentació oficial del mòdul "fetch" i utilitza aquest mòdul per descarregar-te a l'escriptori de la màquina "A" els respectius arxius /etc/passwd de les màquines "B" i "C".

c) Consulta la documentació oficial del mòdul "find" i utilitza aquest mòdul per trobar a les màquines "B" i "C" tots els arxius *.log ubicats dins de la carpeta /var/log que tinguin un tamany major de 1MB

d) Consulta la documentació oficial del mòdul "hostname" i utilitza aquest mòdul per canviar el nom de la màquina "B" (només) per a què sigui "pepita". Comprova-ho.

e) Consulta la documentació oficial del mòdul "Synchronize" i digues quin hauria de ser el contingut d'un "playbook" que sincronitzés el contingut de la carpeta /home de la màquina "A" amb el de la carpeta /home de la màquina B. Molta atenció a l'ús de la directiva *delegate_to*!!

6.-a) Consulta la documentació oficial del mòdul "unarchive" i digues per a què serveix aquest mòdul en general i els seus paràmetres "src", "dest" i "remote_src" concretament

NOTA: També existeix un mòdul que fa el contrari que aquest anomenat "archive" (https://docs.ansible.com/ansible/latest/modules/archive_module.html)

b) Consulta la documentació oficial del mòdul "ipify_facts" i digues per a què serveix aquest mòdul

c) Consulta la documentació oficial del mòdul "nmcli" i digues per a què serveix aquest mòdul

https://docs.ansible.com/ansible/latest/modules/blockinfile_module.html
http://www.mydailytutorials.com/ansible-blockinfile-module-adding-multiple-lines
https://docs.ansible.com/ansible/latest/modules/ini_file_module.html
https://docs.ansible.com/ansible/latest/modules/stat_module.html
https://docs.ansible.com/ansible/latest/modules/fail_module.html
https://docs.ansible.com/ansible/latest/modules/pause_module.html
https://docs.ansible.com/ansible/latest/modules/set_fact_module.html
https://docs.ansible.com/ansible/latest/modules/wait_for_module.html
https://docs.ansible.com/ansible/latest/modules/wait_for_connection_module.html
https://docs.ansible.com/ansible/latest/modules/assemble_module.html
https://docs.ansible.com/ansible/latest/modules/get_url_module.html
https://docs.ansible.com/ansible/latest/modules/uri_module.html
https://docs.ansible.com/ansible/latest/modules/authorized_key_module.html
https://docs.ansible.com/ansible/latest/modules/wakeonlan_module.html

https://docs.ansible.com/ansible/latest/modules/modprobe_module.html
https://docs.ansible.com/ansible/latest/modules/acl_module.html

https://docs.ansible.com/ansible/latest/modules/parted_module.html
https://docs.ansible.com/ansible/latest/modules/filesystem_module.html
https://docs.ansible.com/ansible/latest/modules/mount_module.html

https://docs.ansible.com/ansible/latest/modules/mail_module.html
https://docs.ansible.com/ansible/latest/modules/telegram_module.html
https://docs.ansible.com/ansible/latest/modules/jabber_module.html
https://docs.ansible.com/ansible/latest/modules/mqtt_module.html

d) Després de llegir http://docs.ansible.com/ansible/latest/modules/include_module.html ,¿per a què creus que serveix la directiva "include" dins d'un "playbook"?

7.-a) Escriu (i executa) un "playbook" que faci el mateix que l'apartat a) de l'exercici 1. Comprova que el paquets "curl" i "nmap" tornin a estar instal.lats a les màquines "B" i "C".

b) Escriu (i executa) un "playbook" que faci el mateix que l'apartat a) de l'exercici 2. Comprova que l'usuari "pepe" torni a aparèixer a les màquines "B" i "C". ¿Com modificaries el "playbook" per a què en comptes de crear només l'usuari "pepe" creés a més l'usuari "manolo" i "ana"?

c) Escriu (i executa) un "playbook" que faci el mateix que l'apartat b) de l'exercici 4 però sense arribar a eliminar els elements. Comprova que, efectivament, tornin a aparèixer a les màquines "B" i "C".

8.-a) Consulta la documentació oficial del mòdul "replace" i digues per a què serveix aquest mòdul en general i els seus paràmetres "path", "regexp" i "replace" concretament.

b) Observa el següent "playbook" (que fa ús dels mòduls "hostname" i "replace") i dedueix per a què serveix. PISTA: el valor de la variable *{{ aula }}* s'indica en el paràmetre -e de la comanda *ansible-playbook* i representa el nom d'una aula de l'institut ("stallman", "torvalds", etc). En canvi la variable {{ ansible_default_ipv4.address }} és un "fact" predefinit (fixeu-vos també en l'ús del mètode *split()* de Python i de la conseqüent obtenció d'un determinat element de l'array generat).

```
---
  - name: xxx
    hostname: name={{ aula }}-{{ ansible_default_ipv4.address.split('.')[3] }}
  - name: yyy
    replace:
      dest: /etc/hosts
      regexp: '^127\.0\.1\.1.+$'
      replace: "127.0.1.1\t{{ aula}}-{{ ansible_default_ipv4.address.split('.')[3] }}"
```

c) Observa el següent "playbook" (que fa ús també del mòdul "replace") i dedueix per a què serveix. Consulta https://wiki.debian.org/UnattendedUpgrades#Automatic_call_via_.2Fetc.2Fapt.2Fapt.conf.d.2F20auto-upgrades

```
---
  - name: xxx
    replace:
      dest: /etc/apt/apt.conf.d/20auto-upgrades
      regexp: '^APT::Periodic::Update-Package-Lists "1";'
      replace: 'APT::Periodic::Update-Package-Lists "0";'
  - name: yyy
    replace:
      dest: /etc/apt/apt.conf.d/20auto-upgrades
      regexp: '^APT::Periodic::Unattended-Upgrade "1";'
      replace: 'APT::Periodic::Unattended-Upgrade "0";'
```

Ansible (II)

Variables

1.-a) Crea (o reutilitza) tres màquines virtuals de tipus Server amb la tarja de xarxa en mode "adaptador pont", amb 1GB de RAM i que tinguin un usuari amb el mateix nom a totes tres que pugui fer *sudo*. En un d'aquestes màquines (l'anomenarem "A") hauràs d'instal.lar Ansible (si no ho està ja) i en les altres dues (que anomenarem "B" i "C" i seran administrades per la primera) hauràs d'instal.lar els paquets "openssh-server" i "python" (si no ho estan ja).

b) Edita l'inventari de l'Ansible per tal de què tingui el següent contingut:

```
[victimes]
ip.maq.B varB=manolo
ip.maq.C
[victimes:vars]
varGlobal=pepito
```

c) Executa la següent comanda, *ansible-playbook -v -e "varD=ana varE=felisa" hola.yml* , on "hola.yml" és un playbook amb el següent contingut, i observa quin és el contingut de cada variable mostrada (i raona per què):

NOTA: El paràmetre -v (mode verbós) és important per a què apareguin en pantalla els missatges de depuració indicats al playbook. Necessitaràs afegir també le paràmetre -k si no tens configurada l'autenticació per claus SSH.

```
- name: Playbook per veure els valors de variables definides a diferents llocs
  hosts: all
  remote_user: usuari
  #Aquesta secció 'vars' es podria haver definit en un fitxer apart anomenat group_vars/all.yml
  vars:
     varPlayD : "{{ varD }}"
     varPlayE : "{{ varE }}"
     varPlayF : "perico"
  tasks:
     - name: Veure les variables definides en l'inventori
       debug:  msg="varGlobal val {{ varGlobal }} i varB val {{ varB }}"
     - name: Veure les variables passades per paràmetre
       debug:  msg="varPlayD val {{ varPlayD }} i varPlayE val {{ varPlayE }}"
     - name: Veure les variables definides en el playbook
       debug:  msg="varPlayF val {{ varPlayF }}"
     - name: Veure les variables definides en la tasca
       vars:
          varTaskG : "juan"
       debug:  msg="varTaskG val {{ varTaskG }}"
     - name: Veure el valor d'un determinat "fact"
       debug:  msg="La distribució de la víctima és {{ ansible_facts['distribution'] }}"
```

NOTA: Us hauríeu de fixar en què no es realitzen les tasques per la màquina C degut a què ocorre un error al principi i ja no es continua per aquella màquina. L'error és que varB no es troba definida per ella. Aquest error no es pot obviar amb l'opció *ignore_errors: yes*

d) Al llarg dels següents exercicis veurem vàries utilitats de les variables en playbooks, però una que podem conèixer ja és la possibilitat que ofereixen de crear grups dinàmics de màquines que no estan definits a l'inventori a partir del valor comú que pugui tenir algun "fact" gràcies a l'ús de la clàusula *group_by*. En aquest sentit, digues què faria el següent playbook:

```
- name: Parlar amb totes les víctimes per recollir-ne les seves característiques
  hosts: all
  remote_user: usuari
  tasks:
     - name: Classificar víctimes segons la distribució Linux que estan executant
       group_by:
            key: os_{{ ansible_facts['distribution'] }}
- name: Després de classificar-les, ara actuo només amb les víctimes que executen CentOS
  hosts: os_CentOS
  remote_user: usuari
  gather_facts: no
  tasks:
     - name: Instalar paquets mitjançant dnf (i no cap altre gestor de paquets)
       dnf: name=gnuchess state=latest
```

e) És possible, tal com ja hem comentat en alguna ocasió anterior, definir variables en un fitxer extern. En aquest sentit, digues què faria el següent playbook, el qual utilitza el mòdul "include_vars" per això:

```
- name: Recollir variables vàlides només per certes víctimes i mostrar el seu valor
  hosts: all
  tasks:
     - name: Recollir el valor
       include_vars: "os_{{ ansible_facts['distribution'] }}.yml"
     - name: Mostrar el valor
       debug: msg="El valor de la variable 'asdf' és  {{ asdf }}"
```

f) Una altra possibilitat que ofereix l'ús de variables és, mitjançant la secció especial "vars_prompt" i el mòdul "prompt", demanar interactivament els seus valors per tal d'executar playbooks amb valors d'entrada diferents cada cop (per exemple, per introduir una contrasenya que no es vol deixar escrita). Prova el següent playbook (en mode verbós), observa què passa i dedueix per a què serveixen les opcions "default" i "private".

```
- name: Preguntar dades interactivament (els mateixos valors per totes les víctimes)
  hosts: all
  remote_user: usuari
  vars_prompt:
     - name: "name"
       prompt: "what is your name?"
       default: "Simply the best"
     - name: "password"
       prompt: "what is your password?"
       private: no
     - name: "favcolor"
       prompt: "what is your favorite color?"
  tasks:
     - name: Veure el valor introduït
       debug:  msg="El teu nom es {{ name }}, la contrasenya és {{ password }} i el teu color preferit és {{ favcolor }}"
```

NOTA: Si a través del paràmetre -e de la comanda ansible-playbook s'indica el valor d'una variable indicada a la secció "vars_prompt", llavors no es preguntarà interactivament per ella perquè s'agafarà automàticament el valor indicat a la línia de comandes.

NOTA: If Passlib library (package "python3-passlib") is installed, "vars_prompt" can also encrypt the entered value adding the following options: "encrypt: sha512_crypt", "confirm:yes" and "salt_size:7". Per saber altres valors possibles per l'opció "encrypt" es pot consultar https://docs.ansible.com/ansible/2.5/user_guide/playbooks_prompts.html#prompts

"Tags"

Tags will let you select which part of the playbook you want to run. By using tags, you can tell Ansible "Hey, just run the tasks taggued with 'pfff'". You can even negate tags, asking "Hey Ansible, run all tasks but 'pfff' ones". You can also ask things like "run tasks tagged A or B, but not C". Tagging a task is as simple as adding the keyword `tags:` and a list of tags. Example:

```
- name: Do something really interesting
  debug: msg="Yes this does something really interesting"
  tags:
    - interesting
    - awesome
```

An alternate Yaml syntax is :

```
- name: Do something really interesting
  debug: msg="Yes this does something really interesting"
  tags: ['interesting', 'awesome']
```

Now you can ask Ansible to execute tasks having certain tags by using *-t* at the command line, like this: *ansible-playbook myplaybook.yml -t interesting* The previous task will be executed when no tags are provided at the command line, *-t interesting* is provided, *-t awesome* is provided or when multiple tags are provided (like this: *-t boring,interesting* or like this: *-t boring -t interesting*). You can do the contrary: execute tasks not having certain tags by using *--skip-tags* at command line.

2.-Escriu un playbook que contingui dues tasques a executar a les víctimes: la primera ha d'instal.lar un paquet (el que tu vulguis) i la segona ha de posar en marxa el servei Cups. Aquesta segona ha d'estar etiquetada. Executa el playbook sense indicar cap etiqueta i després indicant l'etiqueta d'aquesta segona tasca amb el paràmetre *-t*. ¿Quina diferència veus? ¿Què passa si ara indiques l'etiqueta d'aquesta segona tasca amb el paràmetre *--skip-tags*?

"Handlers"

Otra funcionalidad interesante de los "playbooks" es la posibilidad de disparar automáticamente tareas en el momento de ejecutar una tarea anterior. Esto se consigue mediante el uso del ítem "notify" de la tarea disparadora y el elemento "handlers", que no deja de ser una "task" que solo se ejecuta si es "notificado". Veamos un ejemplo donde, al instalar el programa nmap, automáticamente se arranca el servicio Nginx:

```
- name: Playbook d'exemple de handlers
  hosts: all
  remote_user: usuari
  become: yes
  tasks:
   - name: Instalar nmap
     apt: name=nmap state=latest
     notify:
        - Pepe
  handlers:
   - name: Pepe
     service: name=nginx state=started
```

3.-Escriu un playbook que instal.li el servidor Apache a les màquines víctimes i que, just després de la instal.lació, mitjançant un "handler" l'habiliti i el posi en marxa.

Bucles

Often you'll want to do many things in one task, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached. To save some typing, repeated tasks can be written in short-hand like so:

```
- name: Loop playbook example
  hosts : all
  remote_user: usuari
  become: yes
  tasks:
     - name: Add several users (with the same password)
       user:
           name: "{{ item }}"
           password: \$6\$qwb23\$2ghw
           state: present
       loop:
           - testuser1
           - testuser2
```

You can see we're using the reserved word *loop* combined with the special variable *{{ item }}* . If you have defined a YAML list in the 'vars' sections (on in a variables file), you can also do:

```
- name: Loop playbook example 2
  hosts: all
  remote_user: usuari
  become: yes
  vars:
     usuaris: [ 'testuser1', 'testuser2' ]
 tasks:
     - name: Add several users (with the same password)
       user:
           name: "{{ item }}"
           password: \$6\$qwb23\$2ghw
           state: present
       loop: "{{ usuaris }}"
```

NOTA: It's possible to create several users with different passwords in a loop, but we need to use "dictionary's lookups", an Ansible construct which we haven't studied yet.

NOTA: Some plugins like, the "dnf" and "apt" modules can take lists directly to their options, and this is more optimal than looping over the task. See each action's documentation for details but here is an example:

```
- name: optimal dnf
  dnf:
    name: "{{list_of_packages}}"
    state: present
```

instead of :

```
- name: non optimal dnf, not only slower but might cause issues with interdependencies
  dnf:
     name: "{{item}}"
     state: present
  loop: "{{list_of_packages}}"
```

Per més informació, https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#loops

4.-a) Escriu (i executa a les víctimes) un playbook que creï els fitxers /home/usuari/pepe.txt, /home/usuari/luisa.txt i /home/usuari/manolo.txt fent servir la construció *loop*

En alguna ocasión nos irá bien reintentar una tarea hasta que una determinada condición se cumpla. Para ello deberemos usar la opción *register* junto con el bucle *until*. La opción *register*, escrita dentro de una determinada tarea, sirve para indicar en qué variable se guardará el resultado de dicha tarea y la opción *until* comprueba que ese resultado cumpla una determinada condición para saber si ha de reintentar de nuevo ejecutar dicha tarea o ya no y puede seguir adelante con el resto del playbook. Antes de poner un ejemplo de bucle *until* antes hay que entender, no obstante, cuál es el contenido concreto que recibe la variable indicada en *register* tras haberse ejecutado la tarea para poder entonces escribir una condición en consecuencia. Para ello, continúa con el siguiente apartado:

b) Executa el següent playbook i observa el contingut mostrat a pantalla de la variable *pepe* . Hauràs de veure que conté un array anomenat "results" amb un element per cada resultat obtingut en la tasca on s'ha omplert (en aquest cas hauran de ser dos elements perquè la tasca "Populate pepe" s'ha realitzat dos vegades a causa del *loop*):

```
- name: Register playbook example
  hosts : all
  remote_user: usuari
  tasks:
     - name: Populate pepe
       debug: msg="{{ item }}"
       loop: ["one", "two" ]
       register: pepe
    -  name: See pepe's content
       debug: msg="{{ pepe }}"
```

NOTA: Qualsevol variable definida a l'opció *register* contindrà una sèrie d'elements que venen definits a https://docs.ansible.com/ansible/latest/reference_appendices/common_return_values.html

c) A partir d'haver vist el resultat de l'apartat anterior, dedueix què faria el següent playbook (i prova'l després a veure si estaves en el cert), tenint en compte que el subobjecte *stdout* de l'array *results* sempre existirà i representa el missatge que hauria aparegut a la sortida estàndar (pantalla) en realitzar-se la tasca en qüestió en un terminal, i que el mètode *find* d'aquest subobjecte busca una determinada cadena en aquesta sortida (retornant -1 si no la troba):

```
- name: Until playbook example
  hosts : all
  remote_user: usuari
  tasks:
     - name: Repeat
       shell: echo "És xulo passejar"
       register: pepe
       until: pepe.stdout.find("És xula la platja") != -1
       retries: 5
       delay: 10
```

NOTA: Les opcions *retries* i *delay* són opcionals perquè ambdues tenen valors per defecte predefinits (3 i 5, respectivament). La primera limita la repetició de la tasca al màxim indicat (per evitar així bucles infinits); la segona indica el temps entre intent i intent (en segons)

NOTA: The registered variable will have a key “attempts” which will have the number of the retries for the task.

NOTA: You can run every retry manually ("step by step") adding the *--step* argument to *ansible-playbook* command. In fact, this argument can be used to run several task from a playbook step by step without the presence of any loop, too (usefult for debugging).

Condicionales

A menudo la ejecución de una tarea depende del valor de una variable, un "fact" o el resultado de una tarea anterior. Para definir qué condición/nes se han de cumplir para ejecutar una determinada tarea se puede usar el ítem *when*. Este ítem tiene como valor una condición del tipo *nombreVariable operador valor* o *nombreFact operador valor* , donde *nombreVariable* o *nombreFact* **no** llevan las llaves {{ y }} alrededor y *operador* puede ser ==, != , < , > , => o <=. Por ejemplo: *when: ansible_distribution == "Debian"*

Si quisiéramos incluir varias condiciones en una línea *when* podemos usar las palabras clave *and* o *or* junto con el uso de paréntesis, así por ejemplo: *when: (ansible_distribution == "CentOS" and ansible_distribution_major_version == "6") or (ansible_distribution == "Debian" and ansible_distribution_major_version == "7")*

También se pueden definir condiciones a partir del estado final de una tarea anterior, que es algo que se utiliza mucho. Para ello deberemos usar el ítem *register* que ya hemos estudiado en el apartado anterior. Como ya sabemos, este ítem, escrito dentro de una determinada tarea sirve para indicar en qué variable se guardará el resultado de dicha tarea. Si entonces utilizamos esa variable en un elemento *when* de una tarea posterior del mismo playbook, estaremos definiendo una condición que, basándose en el valor recogido en esa variable, servirá para decidir si se ejecuta (o no), esa tarea posterior.

En este sentido, existen un conjunto de condiciones concretas predefinidas relacionadas con el estado de finalización de una tarea anterior que pueden usarse para ejecutar tareas posteriores dependiendo de si esa tarea anterior ha finalizado correctamente o no. Para ello debemos utilizar la variable indicada en el ítem *register* de esa tarea anterior, así:

nombreVariable is succeeded
nombreVariable is failed
nombreVariable is defined
nombreVariable is undefined
nombreVariable (para comprobar si vale *true*)
not nombreVariable (para comprobar si vale *false*)

A continuación presentamos un ejemplo de "playbook" donde se anima al lector a deducir qué acciones realiza:

```
- name: When playbook example
  hosts : all
  remote_user: usuari
  vars:
     - docroot: /var/www/html/public
  tasks:
     - name: Add Nginx Repository
       apt_repository: repo='ppa:nginx/stable' state=present
       when: ansible_os_family == "Debian"
       register: ppastable
     - name: Install Nginx
       apt: name=nginx state=installed update_cache=true
       when: ppastable is succeeded
       register: nginxinstalled
       notify:
          - Start Nginx
     - name: Create Web Root
       when: nginxinstalled is succeeded
       file: path={{ docroot }} mode=775 state=directory owner=www-data group=www-data
       notify:
          - Reload Nginx
  handlers:
```

```
    - name: Start Nginx
      service: name=nginx state=started
    - name: Reload Nginx
      service: name=nginx state=reloaded
```

Otras condiciones que se pueden usar en una tarea para definir si se realizará o no son:

nombreVariable is directory
nombreVariable is file
nombreVariable is link
nombreVariable is exists
nombreVariable is mount
nombreVariable is samefile("/ruta/fitxer")
nombreVariable is match("cadenaCompletaACoincidir")
nombreVariable is search("cadenaDondeConcidirSubcadena")

NOTA: *match* y *search* admiten expresiones regulares

Per més informació https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html

5.- a) ¿Què fa aquest "playbook"? Prova'l (en mode verbós)

```
- name: Exercice 5a
  hosts : all
  remote_user: usuari
  tasks:
     - name : xxx
       debug:
           msg="System {{ inventory_hostname }} has gateway {{ ansible_default_ipv4.gateway }}"
       when: ansible_default_ipv4.gateway is defined
```

b) ¿Què fa aquest "playbook"? Prova'l (en mode verbós)

```
- name: Exercice 5b
  hosts : all
  remote_user: usuari
  tasks:
     - name: yyy
       shell: cat /etc/motd
       register: motd_contents
     - name: zzz
       debug: msg="motd contains the word hi"
       when: motd_contents.stdout.find('hi') != -1
```

c) ¿Què fa aquest "playbook"? Prova'l (en mode verbós)

```
- name: Exercice 5c
  hosts : all
  remote_user: usuari
  tasks:
     - name: www
       shell: ls /opt
       register: contents
     - name: jjj
       debug: msg="Directory is empty"
       when: contents.stdout == ""
```

d) ¿Què fa el playbook anomenat "When playbook example", mostrat en aquest apartat de teoria?

e) Com a valor de *when* es pot indicar una llista de múltiples condicions; en aquest cas, s'hauran de complir totes (és a dir, es fa un AND) per a què es consideri que es pot executar la tasca pertinent. Sabent això, ¿què creus que faria aquesta tasca d'un determinat playbook?

```
tasks:
  - name: Mistery
    command: /sbin/shutdown -t now
    when:
      - ansible_facts['distribution'] == "CentOS"
      - ansible_facts['distribution_major_version'] == "7"
```

f) ¿Què passa si executes aquesta comanda *ansible-playbook -e "epic=false" hola.yml* , on "hola.yml" és un playbook amb el següent contingut? ¿I si executes *ansible-playbook -e "epic=true" hola.yml* ?

```
- name: When playbook with variables
  hosts : all
  remote_user: usuari
  vars:
     - epic: {{ epic }}
  tasks:
     - name: Let's see
       shell: echo "This certainly is epic!" > /home/usuari/epic.txt
       when: epic
```

g) Si es combina when amb loop, cal tenir en compte que when es processa per separat per cada ítem del bucle. Sabent això, dedueix què faria aquesta tasca d'un determinat playbook:

```
tasks:
  - name: Mistery
    command: echo {{ item }}
    loop: [ 0, 2, 4, 6, 8, 10 ]
    when: item > 5
```

"Lookups"

Ansible es capaz de obtener datos de fuentes externas (y guardarlos en variables dentro de un "playbook") mediante un "plugin" especial llamado "lookup". Por ejemplo, para almacenar en una variable el contenido de un fichero de texto (llamado "lorem.txt" y ubicado en la misma carpeta donde se encuentre el "playbook" en cuestión) se puede usar el "lookup" "file", así:

```
vars:
  - content: "{{ lookup('file','lorem.txt') }}"
```

Por otro lado, si queremos, por ejemplo, obtener un valor determinado almacenado en un archivo CSV, podemos usar el "lookup" "csvfile". Por ejemplo, para conseguir el valor de la tercera columna (empiezan por 0) que se corresponde con que el valor de la primera columna sea "hola" y cuyo delimitador de campos es la coma, se puede hacer así:

```
vars:
  - content: "{{ lookup('csvfile','hola file=lorem.csv delimiter=, col=2') }}"
```

Si queremos obtener, en cambio, el valor de una determinada variable de entorno existente en la máquina donde estamos ejecutando Ansible, se puede hacer así:

```
vars:
  - content: "{{ lookup('env','nombreVariableEntorno','otraVariableEntorno','otra',...) }}"
```

Hay muchos más: el "lookup" "url", por ejemplo, sirve para almacenar el contenido (HTML, presumiblemente) de la URL indicada; el "lookup" "pipe" sirve para almacenar la salida de un determinado comando ejecutado "ad-hoc", etc. Para saber la lista entera de "lookups", ver http://docs.ansible.com/ansible/latest/plugins/lookup.html

6.-a) Escriu un playbook que executi una tasca implementant el mòdul "user" d'aquesta manera. ¿Què passarà?

```
- name: Misterio
  user:  name=pepe  expires="{{lookup('pipe','date +%s')}}"
```

b) ¿Quina diferència hi ha entre el que es veuria en executar aquesta tasca ...

```
- name: Misterio2
  debug:  msg="{{lookup('env','HOME')}}"
```

... i el que es veuria executant aquesta altra, la qual fa servir el "fact" ansible_env?

```
- name: Misterio2
  debug:  msg="{{ ansible_env.HOME }}"
```

"Callbacks"

"Callbacks" are essentially hooks provided by Ansible: Ansible will send an event and you can react to it with a callback. You could use a callback to do things like print additional details, record the playbook run data somewhere, etc. so, in symmary, "callback plugins" enable adding new behaviors to Ansible when responding to events. You can see the list of available "callback plugins" here (http://docs.ansible.com/ansible/latest/plugins/callback.html).

Uno de los aspectos más útiles que los plugins "callback" pueden gestionar es, pues, el formato (y destino!) del resultado de las tareas ejecutadas por el comando *ansible* o *ansible-playbook* . Concretamente, el comando *ansible* por defecto utiliza un callback llamado "minimal" y el comando *ansible-playbook* utiliza otro llamado "default". Ambos "callbacks" son de tipo "stdout", lo que significa que el resultado ambos lo envían a la salida por terminal, pero cada uno con formatos diferentes. No obstante, como acabamos de decir, hay muchos más plugins "callback", los cuales pueden no ser de tipo "stdout"; si no lo son, entonces enviarán la salida a otros destinos que no son un terminal (como por ejemplo servidores REST remotos, servidores de bases de datos, mensajes de correo, etc).

Para definir el "callback" de tipo "stdout" (solo puede haber uno) que deseamos, debemos modificar la línia *stdout_callback = ...* del archivo "/etc/ansible/ansible.cfg". Ejemplos de "callbacks" de este tipo son: "actionable", "debug", "dense", "full_skip", "yaml" o "json" (ver documentación)

> NOTA: La línea anterior por defecto solo afecta a ansible-playbook. Si queremos que afecte también al comando *ansible* deberemos modificar además la línea *bin_ansible_callbacks* estableciéndola a *true*

Para definir los "callbacks" que no sean de tipo "stdout" debemos modificar la línia *callback_whitelist= ...* del archivo "/etc/ansible/ansible.cfg" (pueden activarse los que se deseeen, separados

por comas). Ejemplos de "callbacks" de este tipo son: "log_plays", "syslog_json" o "mail" (ver documentación)

> NOTA: You can activate a custom callback by either dropping it into a "callback_plugins" directory adjacent to your playbook or by putting it in one of the callback plugin directory sources configured in ansible.cfg.

7.-a) Consulta la documentació oficial per esbrinar què mostra el "callback" "actionable"

b) Consulta la documentació oficial per esbrinar què mostra el "callback" "log_plays". ¿És compatible amb el "callback" "actionable"?

c) Canvia el format de sortida de la comanda *ansible* per a què mostri el resultat en format YAML. Prova-ho executant el mòdul "setup" a tots els hosts de l'inventari.

d) Utilitza els callbacks "actionable" i "log_plays" i comprova quin és el resultat executant el mòdul "setup" a tots els hosts de l'inventari.

"Connection plugins"

Connection plugins allow Ansible to connect to the target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time. The most commonly used are the "ssh" and "local" connection types. The desired transport can be specified using the *-c* argument of *ansible* or *ansible-playbook* commands, using the *ansible_connection* option in inventory or editing "ansible.cfg" file suitably. It can be specified too inside a playbook

> NOTA: You can extend Ansible to support other custom transport (such as SNMP or message bus) by either dropping it into a "connection_plugins" directory adjacent to your playbookor by putting it in one of the connection plugins directory sources configured in ansible.cfg.

8.-¿Què pretén aquest "connection plugin" (no oficial): https://github.com/tomeon/ansible-connection-machinectl ?

"Strategies"

Strategies are a way to control play execution. By default, plays run with the "linear" strategy, in which all hosts will run each task before any host starts the next task, using the number of forks (default 5) to parallelize. Another strategy shipped with Ansible is "free" which allows each host to run until the end of the play as fast as it can.

Let's explain this: with strategy set to linear, Ansible waits until the current task has run on all hosts before starting the next task on any host. Ansible may have forks free, but will not use them until all hosts have completed the current task. If each task in your playbook must succeed on all hosts before you run the next task, use the linear strategy. Using the free strategy, however, Ansible uses available forks to execute tasks on each host as quickly as possible. Even if an earlier task is still running on one host, Ansible executes later tasks on other hosts. The free strategy uses available forks more efficiently. If your playbook stalls on each task, waiting for one slow host, consider using strategy: free to boost overall performance.

Strategy can be specified using the *ANSIBLE_STRATEGY* environment variable, or editing the "strategy=linear" line present below "[strategy]" section of "ansible.cfg" file or inside a playbook (or even in an individual task) like this:

```
- name: Free playbook example
  hosts: all
  remote_user: usuari
```

```
strategy: free
tasks:
...
```

NOTA: All strategy plugins shipped with Ansible are enabled by default. You can enable a custom strategy plugin by putting it in one of the lookup directory sources configured in ansible.cfg

9.-Executa la següent tasca en totes les víctimes definides al teu inventari utilitzant primer l'estratègia "linear" (per defecte) i després l'estratègia "free". ¿Quina ha finalitzat abans? ¿I si tornes a executar un altre cop la mateixa tasca amb les dues estratègies, es manté la mateixa situació? Per saber això amb números reals i no sensacions, en executar el playbook pots escriure la comanda *time* abans d'*ansible-playbook* (així: *time ansible-playbook hola.yml* ...): aquesta comanda informa del temps trigat en finalitzar la comanda que s'hi indiqui a continuació.

```
- name: Instalar nmap
  apt: name=nmap state=latest
```

"Vaults"

Vault is a way to encrypt sensitive information in ansible scripts by encrypting it. Per més informació, https://docs.ansible.com/ansible/latest/user_guide/playbooks_vault.html

10.-Canvia la configuració del *sudo* de les víctimes per a que torni a demanar contrasenya (continua amb l'autenticació per claus SSH, això sí). La idea d'aquest exercici és poder fer servir aquest *sudo* de forma automàtica (és a dir, sense que demani res interactivament) gràcies a tenir escrita la contrasenya que necessita (de forma encriptada) en un fitxer de la màquina on executis Ansible. Per aconseguir això, realitza els passos següents:

a) Executa la comanda *ansible-vault create secret.txt* i, després d'introduir la contrasenya que desitjis, a l'editor de texte per defecte que aparegui, escriu la següent línia, guarda i surt:

```
ansible_become_pass: lacontrasenyaquesigui
```

NOTA: Podràs editar aquest fitxer en qualsevol moment amb la comanda ansible-vault edit secret.txt (et preguntarà la contrasenya que vas introduir en el moment de crear el fitxer la primera vegada).

b) Crea amb un editor de text un altre fitxer (l'anomenarem "vault.txt") i escriu-hi la contrasenya introduïda al pas anterior. Assegura't de que aquest fitxer tingui uns permisos adients per a què ningú més que l'usuari que executi Ansible pugui veure (ni modificar!) el seu contingut.

c) Executa el següent playbook (l'anomenarem "hola.yml") d'aquesta manera: *ansible-playbook hola.yml --vault-password-file=vault.xt* i comprova que, efectivament, no et demani cap contrasenya.

```
- name: Playbook d'exemple de vault
  hosts: all
  remote_user: usuari
  become: yes
  vars_files:
    - secret.txt
  tasks:
    - name: Instalar nmap
      apt: name=nmap state=latest
```

NOTA: Si es vol que es demani interactivament la contraseña del fitxer "secret.txt", llavors caldrà indicar el paràmetre *--ask-vault-pass* en comptes de *--vault-password-file*

NOTA: No cal crear l'arxiu encriptat secret.txt si només es vol encriptar una dada; es pot escriure directament aquesta dada encriptada dins del playbook que la necessiti. En el cas que estem veient, l'encriptació del valor de la variable *ask_become_pass*, es faria així: primer executem *ansible-vault encrypt_string --name 'ask_become_pass' '1234'* i llavors escrivim això al playbook en qüestió:

```
vars:
  ask_become_pass: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    347566848434639493437439760476300347520345987203487209823048234052837 40
    56543645i335897203594823045827340528743520398752034859720345897203458923
    56345634563456346....
```

NOTA: You can encrypt entire playbook doing this: *ansible-vault encrypt hola.yml.* To execute it, you should do something like this, then: *ansible-playbook hola.yml --ask-vault-pass*

Parámetros interesantes de *ansible* y *ansible-playbook* para pruebas

To check if a playbook is correctly formatted, the option *--syntax-check* of *ansible-playbook* looks at all involved playbooks and verifies the correct syntax. During a syntax check, no playbooks are executed.

With complex playbooks and dynamic inventories it sometimes is hard to say against which hosts a playbook will actually be executed. In such cases, the option *--list-hosts* of *ansible-playbook* will output a list of affected hosts, including the name of the actual play and the pattern with which the hosts were chosen. Again, no tasks are actually execute when the list of hosts is queried.

Another thing which can get pretty complicated is the list of tasks actually executed: think of complex playbooks including other complex playbooks. Here the option *--list-tasks* of *ansible-playbook* comes in handy.: it lists what will be done, showing the names of the tasks but not executing any of them

Imagine that a playbook runs without errors, but somehow the result is not what exactly what was expected. In such cases one way to debug everything is to go through each task at a time, step by step, checking the state of all involved components after each task. This can be done with the option *--step* of *ansible-playbook*. And yes, this time the tasks are actually executed on the target node!

During debugging and development it might make sense to start playbooks not at the beginning, but somewhere in between. For example, because a playbook failed at task 14, and you don't want to go through the first 13 tasks again. Starting at a given task requires the appropriate name of the task and the option *--start-at-task* of *ansible-playbook*. The proper name of each task is listed with the *--list-tasks* option

Usually, when a file is changed, Ansible just highlights that a change occurred – but not what was actually changed. In such cases, the option *--diff* comes in handy: it shows the diff in typical patch form.

11.-Llegeix els paràgrafs de teoria anteriors (o consulta el manual d'*ansible*) i digues per a què serveixen els paràmetres *--syntax-check*, *--list-hosts* , *--list-tasks*, *--step*, *--start-at-task i –diff.* Pots provar alguns d'ells, si vols, amb algun playbook que tinguis a mà.

ARA

It’s not uncommon to have hundreds, if not thousands of jobs running every day for testing, building, compiling, deploying and so on. Keeping up with a large amount of Ansible runs and their outcome, not just in the context of CI, is challenging. ARA (https://github.com/openstack/ara) make Ansible runs easier to visualize, understand and troubleshoot.

ARA is four things:

1. An Ansible callback plugin to record playbook runs into a local or remote database
2. A pair of Ansible modules ("ara_record" and "ara_read") to record and read persistent data
3. A CLI client to query the database
4. A web interface to visualize the database. It is dynamic and database-driven but it can also be generated and served from static files.

ARA organizes recorded playbook data in a way to make it intuitive for you to search and find what you’re interested for as fast and as easily as possible. It provides summaries of task results per host or per playbook. It allows you to filter task results by playbook, play, host, task or by the status of the task. You’re able to easily drill down from the summary view for the results you’re interested in, whether it’s a particular host or a specific task. Since you have access to data from multiple runs, you're able to recognize patterns (ex: this particular host is always failing this particular task).

12.-a) Instal.la Ara a la màquina on estàs executant Ansible. Els passos per fer això (com a root) són els següents:

*A Ubuntu: *apt install gcc python-dev libffi-dev libssl-dev && pip install ara*
*A Fedora: *dnf install gcc python-devel libffi-devel openssl-devel && pip install ara*

NOTA: En principi hauria d'aparèixer la línia callback_plugins de l'arxiu ansible.cfg apuntant a la ruta on es trobi instal.lat el callback proporcionat per ARA i també hi hauria d'aparèixer allà una nova secció titulada [ara] indicant configuracions diverses d'ARA que Ansible ha de tenir en compte, entre les quals es troba la directiva "dir", que indica la ruta de la base de dades SQLite que ARA farà servir per guardar els resultats. ARA podria fer servir altres tipus de SGBD com ara MySQL o Postgres, però caldria configurar-ho

NOTA: Per més informació: https://ara.readthedocs.io/en/latest/installation.html

b) Executa la següent comanda, que serveix per carregar les variables d'entorn necessàries per informar Ansible de com usar ARA indistintament de la versió de Python usada al sistema:

source <(python -m ara.setup.env)

c) Executa les següents tres comandes per tal de configurar Ansible per a què reconegui l'existència d'ARA:

python -m ara.setup.path
python -m ara.setup.action_plugins
python -m ara.setup.callback_plugins

NOTA: Les comandes anteriors haurien de modificar el contingut de l'arxiu /etc/ansible/ansible.cfg adientment.
Per més informació: https://ara.readthedocs.io/en/stable/configuration.html

d) Executa qualssevol dels playbooks que tinguis a mà, unes quantes vegades

e) Executa les següents comandes d'ARA i digues què et mostren:

ara playbook list
ara playbook show a73efa33-0d1e-4a7d-8e28-a76fa93b9377
ara play list --all
ara play show wehd3gg-4gwf-4ger-453453453333

ara task list --all
ara task show a152d1e5-fdab-4f50-b84a-de2d0b336124
ara host list
ara host show a73efa33-0d1e-4a7d-8e28-a76fa93b9377
ara host facts ip.un.host
ara result list [-f {csv|json|table|yaml}] [-c nomColumna [-c una altra] ...]
ara result show d9278719-209e-41f8-8ecc-4ad2681dcdf6 [--long]
ara stats list
ara stats show

e) Executa la següent comanda d'ARA: *ara-manage runserver -h 0.0.0.0 -p 9191* i seguidament obre un navegador qualsevol que apunti a la IP de la màquina on està funcionant. ¿Què veus? Investiga les opcions que et dóna aquesta interfície web

NOTA: Per a més informació sobre les possibilitats que ofereix el panell web d'ARA, es pot consultar https://ara.readthedocs.io/en/latest/faq.html#what-does-the-web-interface-look-like

NOTA: Per saber més detalls sobre el funcionament i configuració d'ARA, podeu consultar els següents enllaços:
https://ara.readthedocs.io/en/latest/configuration.html
https://ara.readthedocs.io/en/latest/webserver.html
https://ara.readthedocs.io/en/latest/usage.html

Altres

AWX (https://github.com/ansible/awx) proporciona un panell de control web i una REST API que permet executar Ansible d'una manera més còmoda i remotament. No obstant, per funcionar necessita una màquina de grans recursos (https://github.com/ansible/awx/blob/devel/INSTALL.md#system-requirements) Afortunadament, existeixen alternatives més lleugeres (no oficials) com ara https://github.com/ansible-semaphore/semaphore (una introducció molt interessant es pot trobar aquí: https://blog.strangeman.info/ansible/2017/08/05/semaphore-ui-guide.html#introduction) o també https://github.com/vstconsulting/polemarch .

13.-a) Consulta la pàgina web de Polemarch i digues quines possibilitats ofereix i per a quines circumstàncies se t'acut que seria interessant fer-lo servir

b) Llegeix https://galaxy.ansible.com/intro , visita https://galaxy.ansible.com/search i digues finalment què és Ansible Galaxy i per a què pot ser útil.

c) Visita https://github.com/fboender/ansible-cmdb i digues què fa i per a què serveix aquest programa

Ansible proporciona unes quantes comandes més, a part d'ansible i ansible-playbook, que són interessants de conèixer per si les necessitem en algun moment donat. A l'exercici següent es demana estudiar algunes d'elles.

14.-a) Consulta el manual de la comanda *ansible-config* (oficial d'Ansible) i digues per a què serveix. Prova-la si vols

b) Executa la comanda *ansible-console -u usuari -k* i, un cop a dins de la seva shell interna, executa les següents comandes i digues què fan:

cd victimes
list
shell free -h
cd ip.maq.B
ping

user name=pepe state=absent
exit

c) Llegeix aquest article (https://www.stavros.io/posts/automated-large-scale-deployments-ansibles-pull-mo) i digues per a què serveix la comanda ansible-pull (oficial d'Ansible) i quins beneficis té respecte la manera estàndar de funcionar d'Ansible, que és de tipus "push".

d) Després de llegir https://docs.ansible.com/ansible/latest/plugins/inventory.html#inventory-plugins , digues en quines circumstàncies faries servir els plugins d'inventory "advanced_host_list" o "nmap" (https://docs.ansible.com/ansible/latest/plugins/inventory/advanced_host_list.html) i (https://docs.ansible.com/ansible/latest/plugins/inventory/nmap.html)