# Instalación de ansible.

Para que la instalación de ansible sea correcta hay que asegurarse de que la instalación de python esté bien. Para asegurarme de que todo lo necesario está instalado primero he instalado python-pip, el gestor de módulos de python. Esto ha instalado todos los requisitos necesarios.

Para instalar ansible puedo hacerlo directamente desde ubuntu con apt-get, así se instala ansible con sus dependencias. Lo que recomienda la página de ansible es lo siguiente para la instalación:

apt-get install software-properties-common

apt-add-repository ppa:ansible/ansible

apt-get update

apt-get install ansible

Tras la instalación, al lanzar el comando ansible --version, obtengo la versión instalada de ansible:

root@ubuntuconsolas:/etc/ansible# ansible --version

ansible 2.0.0.1

config file = /etc/ansible/ansible.cfg

configured module search path = Default w/o overrides

con estos comandos la instalación se hace bien, aunque a pesar de haber configurado el apt para usar proxy he necesitado abrir el acceso http y https para la máquina para poder añadir el repositorio.

# Funcionamiento básico de ansible.

Directamente, según la documentación de ansible, una vez que está instalado el software no necesitará una base de datos y no habrá demonios que arrancar ni que deban estar ejecutándose. SE BASA EN SSH.

Donde he instalado ansible se dice que es la máquina de control. Para empezar de forma sencilla edito el archivo que contiene mi lista de hosts o equipos que quiero gestionar con ansible. Este archivo, tras la instalación, es /etc/ansible/hosts y, para probar, añado un par de switches, con lo que lo dejo así:

## Switches planta PTM.

[switches-PTM]

e2p0sis.gmv.es

e2p0sis2.gmv.es

Este archivo se denomina inventory. Una vez creado, con estos switches, lanzamos el comando de prueba ping:

ansible all -m ping

con lo que intentará conectarse a los hosts del inventario con el usuario que estemos usando en este momento, por tanto intenta conectar con root. Para conectar con otro usuario, por ejemplo el usuario cisco, el comando es:

ansible all -m ping -u cisco

Al estar basado en SSH y claves SSH, en el caso de que no podamos fijar claves SSH para SSH sin password puedo forzar que pida una contraseña:

ansible all -m ping -u cisco --ask-pass

así se conectará con el usuario cisco y pedirá la password.

Para poder usar un usuario diferente que el que está lanzando ansible usaré la opción -u pero ese usuario debe existir en la máquina o máquinas remotas, por tanto la solución sencilla es copiar la clave pública rsa ssh del usuario root en el fichero authorized\_keys de todos los nodos o hosts que quiera gestionar con ansible. Así me ahorro el tener que crear un usuario adicional en cada máquina, aunque lo más correcto y seguro sería crearlo.

En general, a pesar de ansible no requiere agente, si necesita que las máquinas tengan al menos el módulo simplejson o stdlib json. Cuando no está instalado y probamos un comando remoto con ansible se recibe un error como el siguiente:

bocdhcp.gmv.es | FAILED! => {

"changed": false,

"failed": true,

"msg": "Error: ansible requires the stdlib json or simplejson module, neither was found!"

}

Una forma rápida de solucionar esto es instalarlo desde el propio ansible lanzando un comando como el siguiente (ten en cuenta que el gestor de paquetes del nodo o nodos remotos debe estar bien configurado y que el nombre del paquete puede variar):

-> En un CentOS - ansible bocdhcp.gmv.es -m raw -a '/usr/bin/yum -y install python-simplejson'

es muy importante tener en cuenta al lanzar estos comandos con el módulo raw que, si un comando necesita interacción en forma de pregunta que hay que responder, es mejor usar las opciones que den las respuestas de manera automática. En el caso de la instalación del paquete usar la opción -y para que se responda yes a la pregunta (esto también te vale para la desinstalación).

# El inventario de ansible.

El archivo /etc/ansible/hosts es el inventario general y en él puedo fijar variables para los diferentes hosts que quiero administrar con ansible. Lo que se recomienda, sin embargo, es que en el archivo de inventario solo tenga los nombres de los hosts que quiero gestionar, sean o no los nombres de host y/o DNS de las máquinas a administrar y que, desde la carpeta de configuración de ansible haga una carpeta para los hosts y otra para los grupos. Por ejemplo, lo que he hecho yo es crear en la carpeta de los grupos, que debe llamarse group\_vars, un archivo con el nombre de cada uno de los grupos que hay en el archivo hosts. En esos archivos puede fijar las varaibles que deben aplicar a cada grupo. Por ejemplo, para los switches:

* En el fichero /etc/ansible/hosts tengo un grupo llamado switchesptm y entonces creo el archivo /etc/ansible/group\_vars/switchesptm con las variables:

ansible\_connection: ssh

ansible\_user: cisco

ansible\_ssh\_pass: lpdsplsdgmv\*666\*

estas variables aplicarán a todos los hosts que formen parte del grupo switchesptm.

* Para los servidores dhcp, he hecho un grupo llamado dhcpservers en /etc/ansible/hosts y luego el archivo /etc/ansible/group\_vars/dhcpservers con las variables:

ansible\_connection: ssh

ansible\_user: root

para así usar estas variables al conectar con los hosts que forman parte del grupo dhcpservers.

Luego puedo definir variables específicas y separadas para cada host, para esto, en el directorio /etc/ansible/host\_vars, crearé un archivo cuyo nombre sea el del host que hay en /etc/ansible/hosts. Así, por ejemplo, en el archivo /etc/ansible/hosts tengo el grupo dhcpservers con estos hosts:

## Servidores DHCP.

[dhcpservers]

metadir

metadir2

bocdhcp

metadirsev

metadirbcn

metadirval

metadirmas

metadirlis

metadirwar

luego, para cada host, tengo un archivo con el mismo nombre, pero con los contenidos en función de cada host, así por ejemplo, para metadir y metadir2 el contenido de esos archivos es:

/etc/ansible/host\_vars/metadir

ansible\_host: metadir-admin.gmv.es

/etc/ansible/host\_vars/metadir2

ansible\_host: metadir2-admin.gmv.es

así controlaré que, aunque use nombres descriptivos para los hosts en el inventario, luego estos apuntarán a los nombres DNS o direcciones IP correctas.

He hecho un grupo llamado vdiptm que contiene todos los servidores XenServer de virtualización de escritorios y luego he creado otros tres grupos por cada pool añadiendo los hosts que forman cada pool. Así tengo por un lado el grupo que engloba a todos, con el fichero correspondiente en /etc/ansible/group\_vars con el nombre vdiptm y luego un archivo para cada host en /etc/ansible/host\_vars. En estos últimos pongo como variable que el host es el nombre de máquina en la red de admin y en el del grupo vdiptm pongo las opciones de conexión y el usuario a usar para conectar. Adicionalmente podría crear también un archivo para cada grupo que define cada pool y que tengan diferentes variables.

# Usando comandos ad-hoc.

Una vez que ya tengo establecido un inventario básico y puedo interactuar con los hosts puedo probar comandos simples de ansible. Estos comandos simples, o comandos ad-hoc como se llaman en la documentación, están pensados para lanzar uno o más comandos simples a varios hosts con una sola línea. Es la forma más sencilla de utilizar ansible.

Lanzar un comando ad-hoc es tan fácil como usar el comando ansible siguiendo una sintaxis como la siguiente:

ansible host/grupo -m Módulo -a Argumentos -u nombreusuario --ask-pass

por ejemplo para lanzar un comando por ssh de fomra simple contra cartman haríamos una comando ad-hoc como el siguiente, después de haber añadido la entrada

de cartman en /etc/ansible/hosts:

ansible cartman.gmv.es -m raw -a "df -h" -u root --ask-pass

así usaré como usuario root y me pedirá la password. El módulo a usar es raw, que básicamente es mandar comandos por ssh y el argumento que se le pasa al módulo es el comando a enviar por ssh, que en este caso es un df -h.

Un módulo de prueba estándar, para comprobar que la máquina ansible conecta con los hosts, es el módulo ping, que se lanza:

ansible poolc -m ping

ansible vdiptm -m ping

Para este tipo de comandos es muy importante el usar comillas simples en vez de dobles si estamos usando variables, por ejemplo si el módulo que usamos es el módulo shell. Cuando use el módulo shell y quiera usar variables QUE DEBEN EXPANDIRSE O EVALUARSE EN LOS HOSTS REMOTOS tengo que usar comillas simples.

El módulo copy me deja copiar archivos entre mi máquina ansible y los hosts remotos. Por ejemplo, con este comando, copiaría el fichero /etc/ansible/hosts a la ruta remota /tmp/hosts.BORRAR en todos los hosts que forman parte del grupo poolc.

ansible poolc -m copy -a "src=/etc/ansible/hosts dest=/tmp/hosts.BORRAR"

si quiero manipular los permisos de dicho archivo usaré el módulo file. Por ejemplo, cambio a 777 los permisos del archivo anterior:

ansible poolc -m file -a "dest=/tmp/hosts.BORRAR mode=777"

ahora, con el siguiente comando, cambio el modo y el propietario de ese archivo:

ansible poolc -m file -a "dest=/tmp/hosts.BORRAR mode=700 owner=qemu group=wheel"

y por último, con el siguiente comando, podré borrar ese fichero en todos esos hosts:

ansible poolc -m file -a "dest=/tmp/hosts.BORRAR state=absent"

Hay muchos más módulos, consúltalos en la web de ansible. Por ejemplo hay un módulo que te permite interactuar directamente con yum, otros para la gestión de usuarios y grupos o para realizar interacciones con sistemas de control de versiones como git o subversion y, si lo que quiero es controlar servicios de forma remota puedo hacerlo con el módulo service.

Para arrancar el servicio iptables en todos los equipos del poolc:

ansible poolc -m service -a "name=iptables state=restarted"

Para tareas que sé que van a tardar puedo hacer que se pongan en background y tengo la opción de lanzar dichas tareas con la posibilidad de poder comprobar

su estado mediante un polling. Por ejemplo:

ansible all -B 1800 -P 60 -a "/usr/bin/long\_running\_operation --do-stuff"

así se especifica que la tarea se ejecutara durante 1800 segundos (30 minutos) y que se comprobará su estado cada 60 segundos. Este comando me devuelve un identificador de tarea o job id. Si quiero lanzarlo sin polling solo tengo que especificar 0 con la opción -P, pero si lo hago y más tarde decido que si quiero hacer un pool del estado de la tarea puedo usar el módulo async\_status con el job id devuelto por el comando para saber el estado de la tarea.

ansible web1.example.com -m async\_status -a "jid=JOB ID devuelto por el comando"

# Usando playbooks.

Los playbooks de ansible son básicamente un conjunto de directivas que permiten la configuración de múltiples servidores remotos o la orquestación de algún proceso o procesos que involucran a muchos sistemas. Se escriben en YAML.

Un fichero en YAML empieza siempre con --- y termina siempre con ..., son las cabeceras que indican su inicio y fin.

En general, en YAML, tengo listas y diccionarios. Una lista se representa siempre así:

---

frutas:

- manzana

- naranja

- platano

- melon

...

mientras que un diccionario es un conjunto de líneas en la forma clave: valor, por ejemplo:

- martin:

name: Nombre

job: Trabajo

skill: elite

Esto se puede abreviar y representar de forma resumida del siguiente modo:

---

employees:

- martin: {name: Nombre, job: Trabajo, skill: Elite}

frutas: ['manzana', 'naranja', 'platano', 'melon']

...

Para valores booleanos, YAML permite varias opciones:

create\_key: yes

use\_cvs: false

needs\_agent: no

Solo es importante tener en cuenta que, si quiero usar el símbolo : en una cadena entonces debo poner la cadena entre "".

Para definir variables ansible usa {{ }}, así que siempre que quiera usar variables tendré que especificarla así.

De forma simple y resumida, un playbook está compuesto por una o más plays, donde cada play contendrá una serie de tareas a realizar. Una tarea es solamente una llamada a un módulo de ansible. Un playbook se procesa de arriba hacia abajo.

Una de las definiciones básicas y fundamentales que hacen falta siempre en un play es los hosts a los que hay que aplicar el play así como el usuario remoto que se usará para realizar las tareas. Por tanto, cualquier play, debes empezarlo con:

---

- hosts: poold

remote\_user: root

aunque luego, para determinadas tareas, se puede especificar un usuario remoto diferente. Para las ejecuciones remotas, como no, se puede usar y especificar que hay que usar sudo, tanto para todo el play como para tareas específicas.

Cada lista de tareas se ejecuta en orden, una cada vez, a todos los hosts especificados al comienzo del play. Cada tarea consistirá en ejecutar un módulo y es importante tener en cuenta que los módulos son idempotentes con lo que solo se cambian cosas si hay que cambiarlas. Esto lleva a que se puede volver a lanzar un playbook muchas veces, ya que no se cambiarán cosas ya cambiadas.

Las tareas se definen de una forma muy sencilla, con un nombre, que es opcional y es la salida que dará el procesado del playbook, el módulo que se va a ejecutar y los parámetros del mismo. Por ejemplo, para asegurar que el servicio apache se está ejecutando, una tarea sería:

tasks:

- name: Arrancar apache si es necesario

service: name=httpd state=running

para los módulos shell y command la sintaxis varía ya que no usan el formato clave=valor como el resto, así, para asegurar que selinux está deshabilitado, la tarea sería como:

tasks:

- name: Deshabilitar selinux

command: /sbin/setenforce 0

Por último, se pueden definir notificaciones de tareas y manejadores para dichas notificaciones. Esto consiste en lanzar una tarea si, una tarea anterior se ha realizado correctamente. Por ejemplo, tenemos una tarea que copia el fichero de configuración de apache y, en el caso de que se ejecute la tarea de forma correcta, queremos que se lance otra para reiniciar el proceso apache. Para esto, la tarea de copia del fichero tendrá un notify que hará referencia al handler correspondiente:

tasks:

- name: Copiar fichero de configuracion apache

copy: src=/srv/httpd/httpd.conf.ALL dest=/etc/httpd/httpd.conf

notify:

- restart apache <--- REFERENCIA A MI HANDLER POSTERIOR. ESTO SE HARÁ SOLO SI EL FICHERO SE CAMBIA.

......

handlers:

- name: restart apache <--- ESTE ES EL NOMBRE DEL HANDLER AL QUE LLAMA EL NOTIFY

service: name=apache state=restarted

estos handlers se ejecutan siempre al final del playbook, cuando se hayan ejecutado todos los plays del playbook.

Para ejecutar un playbook usaré el comando ansible-playbook con el nombre del playbook.

ansible-playbook fichero\_playbook -f Paralelismo

Ejemplo de playbook. Como ejemplo de playbook sencillo voy a hacer uno que copie el archivo resolv.conf correcto y cree el directorio /backup para los backups de thumper, en todos los hosts ssoo que tenemos. En este caso el playbook debería ser:

---

- hosts: vdiptm

remote\_user: root

tasks:

- name: Copia el archivo resolv.conf correcto

copy: src=/tmp/resolv.conf.correcto dest=/etc/resolv.conf backup=yes

- name: Crea el directorio /backup

file: dest=/backup state=directory mode=755

...

una vez escrito puedo comprobar si es correcto con la opción --syntax-check:

ansible-playbook basic\_vdi.yaml --syntax-check

y para comprobar a que hosts va a aplicar este playbook puedo usar el siguiente comando:

ansible-playbook basic\_vdi.yaml --list-hosts

playbook: basic\_vdi.yaml

play #1 (vdiptm): TAGS: []

pattern: [u'vdiptm']

hosts (9):

ssoo7

ssoo9

ssoo8

ssoo11

ssoo10

ssoo13

ssoo12

ssoo15

ssoo14

Una vez lanzado el playbook con el comando ansible-playbook la salida es similar a la siguiente:

TASK [Copia el archivo hosts correcto] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [ssoo7]

changed: [ssoo15]

changed: [ssoo10]

changed: [ssoo8]

changed: [ssoo9]

TASK [Crea el directorio /backup] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [ssoo8]

ok: [ssoo10]

changed: [ssoo15]

ok: [ssoo7]

ok: [ssoo9]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ssoo10 : ok=3 changed=1 unreachable=0 failed=0

ssoo11 : ok=0 changed=0 unreachable=1 failed=0

ssoo12 : ok=0 changed=0 unreachable=1 failed=0

ssoo13 : ok=0 changed=0 unreachable=1 failed=0

ssoo14 : ok=0 changed=0 unreachable=1 failed=0

ssoo15 : ok=3 changed=2 unreachable=0 failed=0

ssoo7 : ok=3 changed=1 unreachable=0 failed=0

ssoo8 : ok=3 changed=1 unreachable=0 failed=0

ssoo9 : ok=3 changed=1 unreachable=0 failed=0

Como no ha podido conectar correctamente con los hosts ssoo11, ssoo12, ssoo13, ssoo14 el campo changed es 0 para ellos, no se ha cambiado nada en ellos. En el host ssoo15 changed es 2 porque hemos copiado el archivo y creado el directorio y en el resto el changed es solo 1 porque el directorio ya existía y solo se ha cambiado el archivo, haciéndose además una copia por si acaso. En los equipos que ha fallado, ha sido porque no estaba la clave ssh en esos hosts, una vez que lo he corregido, si lo vuelvo a lanzar solo modificará los hosts que antes no ha podido modificar y los otros los dejará como estaban.

Lo lanzo de nuevo, y en este caso ya solo aparecen como change, con valor 1, los que fallaron antes y en los que solo se ha copiado el archivo ya que el directorio ya existía:

TASK [Copia el archivo hosts correcto] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [ssoo7]

ok: [ssoo9]

ok: [ssoo10]

ok: [ssoo8]

changed: [ssoo11]

changed: [ssoo13]

ok: [ssoo15]

changed: [ssoo12]

changed: [ssoo14]

TASK [Crea el directorio /backup] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [ssoo10]

ok: [ssoo7]

ok: [ssoo9]

ok: [ssoo8]

ok: [ssoo11]

ok: [ssoo13]

ok: [ssoo12]

ok: [ssoo15]

ok: [ssoo14]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ssoo10 : ok=3 changed=0 unreachable=0 failed=0

ssoo11 : ok=3 changed=1 unreachable=0 failed=0

ssoo12 : ok=3 changed=1 unreachable=0 failed=0

ssoo13 : ok=3 changed=1 unreachable=0 failed=0

ssoo14 : ok=3 changed=1 unreachable=0 failed=0

ssoo15 : ok=3 changed=0 unreachable=0 failed=0

ssoo7 : ok=3 changed=0 unreachable=0 failed=0

ssoo8 : ok=3 changed=0 unreachable=0 failed=0

ssoo9 : ok=3 changed=0 unreachable=0 failed=0

# Usando variables en playbooks.

En el inventario puedo definir variables, tanto variables específicas para hosts como variables que apliquen a grupos enteros. Para definir las variables que aplican a un grupo solo tengo que definir una entrada como la siguiente en el archivo de inventario:

[grupo]

host1

host2

host3

[grupo:vars]

var1=valor1

var2=valor2

var3=valor3

Para poder organizar los hosts en grupos, y aplicar variables específicas a cada uno de ellos en función de localización, red, organización, etc... puedo hacer grupos de grupos. El problema está en que un grupo, contenido en otro grupo, no se expande, es decir, si yo creo un grupo de hosts y luego ese grupo lo incluyo en otro grupo, no se expande el grupo a los hosts que lo forman.

Una de los usos de las variables es el usar ficheros de plantillas. El ejemplo más fácil y sencillo es el de un playbook que copia y/o crea los archivos resolv.conf en máquinas Linux/Unix en función de los servidores DNS que correspondan. Así, por ejemplo, yo defino un grupo que se llama vdiptm, que contiene todos los servidores ssoo que tenemos. Esto, como ya sé, está definido en el inventario de ansible así:

[vdiptm]

ssoo7

ssoo8

ssoo9

ssoo10

ssoo11

ssoo12

ssoo13

ssoo14

ssoo15

luego, en el directorio group\_vars, especifico un archivo que contiene las variables de ese grupo, así que en el archivo /etc/ansible/group\_vars/vdiptm tengo definido lo siguiente:

ansible\_connection: ssh

ansible\_user: root

dns1\_server: 172.18.18.101

dns2\_server: 172.18.18.102

ntp1\_server: 172.18.18.101

ntp2\_server: 172.18.18.102

con lo que estas variables se aplican a todos los hosts que forman parte de dicho grupo.

Ahora quiero configurar todos esos hosts con el mismo fichero /etc/resolv.conf, para lo cual creo un archivo de template que contiene lo siguiente:

domain gmv.es

nameserver {{ dns1\_server }}

nameserver {{ dns2\_server }}

search gmv.es

y luego, en el playbook que se va a utilizar defino una tarea usando el módulo template del siguiente modo:

- name: Copia el archivo resolv.conf correcto

template: src=/etc/ansible/templates/resolv.conf.j2 dest=/tmp/resolv.conf backup=yes

de este modo, al usar el módulo template, lo que hace ansible es buscar una variable que se llame igual que las que aparecen en el archivo de template y lo sustituirá por los valores que hayamos definido en el archivo de variables de ese grupo, en este caso las variables que hay definidas en el archivo de variables del grupo vdiptm -> /etc/ansible/group\_vars/vdiptm.

Ahora, para seguir con el uso de variables, voy a definir otro fichero de template para el archivo hosts pero, en este caso, voy a usar variables que deben ser locales al host remoto. Una forma sencilla de ver las variables locales de un host remoto es usar el módulo setup como comando ad-hoc. Con este módulo se obtienen lo que ansible llama facts, o lo que es lo mismo, la configuración del host remoto. Así, si lanzo un comando ad-hoc ansible con el módulo setup tengo una salida como la siguiente (esta recortada claro, que saca un huevo de cosas):

ansible ssoo10 -m setup | more

ssoo10 | SUCCESS => {

"ansible\_facts": {

"ansible\_all\_ipv4\_addresses": {

"10.22.2.80"

},

"ansible\_all\_ipv6\_addresses": {

"fe80::225:90ff:fe8f:7c76",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff",

"fe80::fcff:ffff:feff:ffff"

},

"ansible\_architecture": "x86\_64",

"ansible\_bios\_date": "02/06/2014",

voy a crear, por tanto, una plantilla para copiar el archivo hosts a los equipos que formen parte del grupo vdiptm donde se especifique el host con su ip a partir de una variable local del propio host. Es decir, si no me equivoco, debería poder crear una plantilla como esta:

127.0.0.1 localhost localhost.localdomain

{{ ansible\_default\_ipv4.address }} {{ ansible\_hostname }} {{ ansible\_fqdn }}

10.22.2.51 thumper thumper.gmv.es

y defino una nueva tarea en el playbook usando el módulo template del siguiente modo:

- name: Copia el archivo hosts correcto

template: src=/etc/ansible/templates/hosts.vdiptm.j2 dest=/tmp/hosts backup=yes

de esta manera ansible es capaz de sustituir los valores que se corresponden al host local remoto sin problemas.

Por tanto, de forma resumida, puedo definir variables para grupos en los ficheros contenidos en group\_vars, darles el nombre que yo quiera a dichas variables y los valores que necesite para dichas variables. Luego, en las plantillas, solo tendré que especificar los nombres de esas variables como {{ nombre\_var }} y especificar en la tarea correspondiente el module template. Las variables que define el propio ansible de cada host remoto, que puedo obtener con el

modulo setup, también puedo usarlas directamente en una plantilla.

Por tanto, ahora, tengo el siguiente playbook para el grupo de hosts vdiptm:

---

- hosts: vdiptm

tasks:

- name: Copia el archivo hosts correcto

template: src=/etc/ansible/templates/hosts.vdiptm.j2 dest=/tmp/hosts backup=yes

- name: Copia el archivo resolv.conf correcto

template: src=/etc/ansible/templates/resolv.conf.j2 dest=/tmp/resolv.conf backup=yes

- name: Crea el directorio /backup

file: dest=/backup state=directory mode=755

...

y como templates definidas, que podrían usarse con cualquier grupo de hosts, tengo las siguientes:

/etc/ansible/templates/hosts.vdiptm.j2

127.0.0.1 localhost localhost.localdomain

{{ ansible\_default\_ipv4.address }} {{ ansible\_hostname }} {{ ansible\_fqdn }}

10.22.2.51 thumper thumper.gmv.es

/etc/ansible/templates/resolv.conf.j2

domain gmv.es

nameserver {{ dns1\_server }}

nameserver {{ dns2\_server }}

search gmv.es

En estas plantillas solo tienes que considerar que lo que hará ansible será usar las variables definidas, que coincidan con el nombre de la variable, en las plantillas que uses. Esas variables podré haberlas definido yo o bien ser variables propias de ansible, como las que se obtienen cuando uso el módulo setup para obtener los facts del host remoto.

Como ejemplo de archivo de variables para un grupo pongo el archivo en el que defino todas las variables que, en principio, me dejan caracterizar a la red de Barcelona, está mezclado el direccionamiento nuevo que tendrá con el actual por las pruebas que hago:

## /etc/ansible/group\_vars/BCN

ansible\_connection: ssh

ansible\_user: root

delegacion: bcn

dc1\_server: 172.22.209.101

dc2\_server: 172.22.209.102

dns1\_server: 172.22.209.101

dns2\_server: 172.22.209.102

network: 172.18.112.0

network\_netmask: 255.255.254.0

network\_gateway: 172.18.112.1

cidr\_netmask: 23

ntp1\_server: 172.22.209.101

ntp2\_server: 172.22.209.102

metadir\_server: 172.22.209.103

cimmeria\_ip: 172.18.112.9

crom\_ip: 172.18.112.4

De este modo, cuando lance playbooks solo para Barcelona podré usar esas variables con los hosts que pertenezca a dicho grupo.

# Playbook para el cambio de direccionamiento de una delegación.

Voy a hacer un playbook para cambiar la configuración de los recursos de DHCP y METADIR de los clusters de firewalls en las delegaciones.

Un firewall de delegación tiene el servicio DHCP siempre, con lo que tengo que cambiar esa configuración. Para ello defino un playbook con estas tareas:

tasks:

- name: Parar el recurso DHCP.

command: /usr/sbin/crm resource stop DHCP

- pause: seconds=10

- name: Cambiar el fichero de configuracion del recurso DHCP.

template: src=/etc/ansible/templates/dhcpd.conf.j2 dest=/tmp/dhcpd.conf backup=yes

- name: Parar servidor LDAP de configuracion DHCP.

service: name=dhcpserver-ldap state=stopped

- pause: seconds=10

- name: Cambiar el archivo de arranque del servidor LDAP de configuracion DHCP.

template: src=/etc/ansible/templates/dhcpserver-ldap.j2 dest=/tmp/dhcpserver-ldap backup=yes

- name: Arrancar servidor LDAP de configuracion DHCP.

service: name=dhcpserver-ldap state=started

- name: Parar el recurso METADIR.

command: /usr/sbin/crm resource stop METADIR

ignore\_errors: yes

notify:

- Copiar archivo de configuracion METADIRECTORIO.

- Parar el recurso METADIRLOCAL.

- Cambiar configuracion recurso METADIRLOCAL.

- Cambiar configuracion recurso METADIR.

- Arrancar recurso METADIRLOCAL.

- Arrancar recurso METADIR.

- name: Parar el recurso IPDhcp.

command: /usr/sbin/crm resource stop IPDhcp

- pause: seconds=10

- name: Cambiar la direccion IP del recurso IPDhcp.

command: /usr/sbin/crm resource param IPDhcp set ip="{{ metadir\_server }}"

- name: Cambiar la mascara de red del recurso IPDhcp.

command: /usr/sbin/crm resource param IPDhcp set cidr\_netmask="{{ cidr\_netmask }}"

- name: Arrancar el recurso IPDhcp.

command: /usr/sbin/crm resource start IPDhcp

- name: Arrancar el recurso DHCP.

command: /usr/sbin/crm resource start DHCP

handlers:

- name: Copiar archivo de configuracion METADIRECTORIO.

template: src=/etc/ansible/templates/slapd-metadir.conf.j2 dest=/tmp/slapd.conf backup=yes

- name: Parar el recurso METADIRLOCAL.

command: /usr/sbin/crm resource stop METADIRLOCAL

- pause: seconds=15

- name: Cambiar configuracion recurso METADIRLOCAL.

command: /usr/sbin/crm resource param METADIRLOCAL set cmdline\_options="-f /metaldap/slapd-metaldap-local/etc/openldap/slapd.conf -4 -h ldap:/{{ metadir\_server }}3:390"

- name: Cambiar configuracion recurso METADIR.

command: /usr/sbin/crm resource param METADIR set cmdline\_options="-f /metaldap/slapd-metaldap/etc/openldap/slapd.conf -4 -h ldap:/{{ metadir\_server }}3"

- name: Arrancar recurso METADIRLOCAL.

command: /usr/sbin/crm resource start METADIRLOCAL

- pause: seconds=10

- name: Arrancar recurso METADIR.

command: /usr/sbin/crm resource start METADIR

como no hay un módulo específico para controlar un cluster heartbeat, usaré el módulo command para hacer las configuraciones de los recursos del cluster.

En este playbook uso un notify para que, en el caso de que no exista el recurso METADIR, es decir solo haya un servidor DHCP y no haya un servidor de metadirectorio, solo haga la parte del DHCP y la del metadirectorio opase de ella. Para esto se usa el notify, que hará que se llamen a los handlers que hay al final del playbook solo si la tarea en la que está incluido el notify se ejecuta con exito.

A lo largo de este playbook se usan variables definidas en el fichero de grupo BCN, al que pertenecen los dos hosts donde se va a lanzar el playbook (esto hay que controlarlo al estar en cluster los dos), así como variables que son facts de las máquinas remotas. En las plantillas de los archivos que se usan también se usan dichas variables.

La idea de este playbook es que se pueda usar tanto para clusters que tengan el DHCP y el METADIRECTORIO como aquellos que solo tengan el DHCP. Por tanto, para asegurar que el playbook se ejecuta aunque el comando de parada del servicio de METADIRECTORIO falle, se especifica la línea ignore\_errors a yes. Con esta línea, si el recurso no existe, se producirá un error PERO CONTINUARÁ la ejecución del playbook PERO SIN LANZARSE LOS HANDLERS ya que la tarea no se ha ejecutado con éxito. En caso de que se ejecute correctamente la tarea el notify lanzará los handlers asociados al final del playbook.

Siguiendo con este playbook es importante considerar que estoy configurando recursos de un cluster y que el playbook es para dos hosts, con lo que si no lo controlo de algún modo, lanzaría más de una vez los comandos de cluster, lo que no tiene sentido ya que podría lanzarse a la vez en los dos con los posibles problemas que eso pudiese ocasionar. Tengo que asegurar que las tareas que tienen que ver con el cluster se lancen en solo uno de los dos nodos. Para esto se pueden usar las opciones de configuración de tareas delegate\_to y run\_once. Por tanto, si quiero asegurarme de que en un cluster heartbeat, que repito haces así ya que no hay un módulo específico en ansible, los comandos del cluster se ejecuten SOLO EN UNO DE LOS NODOS, tengo que especificar en la tarea correspondiente las opciones delegate\_to y run\_once o usar una condición con when. Por ejemplo:

\*La forma más sencilla que se me ocurre es:

- name: Arrancar recurso METADIR.

command: /usr/sbin/crm resource start METADIR

delegate\_to: fw1bcn

run\_once: true

y esto sería para todas las tareas que impliquen comandos de heartbeat. Por tanto el playbook para el cambio de direccionamiento IP quedaría:

---

- hosts: fw1bcn fw2bcn

tasks:

- name: Parar el recurso DHCP.

command: /usr/sbin/crm resource stop DHCP

delegate\_to: fw1bcn

run\_once: true

- pause: seconds=10

- name: Cambiar el fichero de configuracion del recurso DHCP.

template: src=/etc/ansible/templates/dhcpd.conf.j2 dest=/tmp/dhcpd.conf backup=yes

- name: Parar servidor LDAP de configuracion DHCP.

service: name=dhcpserver-ldap state=stopped

- pause: seconds=10

- name: Cambiar el archivo de arranque del servidor LDAP de configuracion DHCP.

template: src=/etc/ansible/templates/dhcpserver-ldap.j2 dest=/tmp/dhcpserver-ldap backup=yes

- name: Arrancar servidor LDAP de configuracion DHCP.

service: name=dhcpserver-ldap state=started

- name: Parar el recurso METADIR.

command: /usr/sbin/crm resource stop METADIR

delegate\_to: fw1bcn

run\_once: true

ignore\_errors: yes

notify:

- Copiar archivo de configuracion METADIRECTORIO.

- Parar el recurso METADIRLOCAL.

- Cambiar configuracion recurso METADIRLOCAL.

- Cambiar configuracion recurso METADIR.

- Arrancar recurso METADIRLOCAL.

- Arrancar recurso METADIR.

- name: Parar el recurso IPDhcp.

command: /usr/sbin/crm resource stop IPDhcp

delegate\_to: fw1bcn

run\_once: true

- pause: seconds=10

- name: Cambiar la direccion IP del recurso IPDhcp.

command: /usr/sbin/crm resource param IPDhcp set ip="{{ metadir\_server }}"

delegate\_to: fw1bcn

run\_once: true

- name: Cambiar la mascara de red del recurso IPDhcp.

command: /usr/sbin/crm resource param IPDhcp set cidr\_netmask="{{ cidr\_netmask }}"

delegate\_to: fw1bcn

run\_once: true

- name: Arrancar el recurso IPDhcp.

command: /usr/sbin/crm resource start IPDhcp

delegate\_to: fw1bcn

run\_once: true

- name: Arrancar el recurso DHCP.

command: /usr/sbin/crm resource start DHCP

delegate\_to: fw1bcn

run\_once: true

handlers:

- name: Copiar archivo de configuracion METADIRECTORIO.

template: src=/etc/ansible/templates/slapd-metadir.j2 dest=/tmp/slapd.conf backup=yes

- name: Parar el recurso METADIRLOCAL.

command: /usr/sbin/crm resource stop METADIRLOCAL

delegate\_to: fw1bcn

run\_once: true

- pause: seconds=15

- name: Cambiar configuracion recurso METADIRLOCAL.

command: /usr/sbin/crm resource param METADIRLOCAL set cmdline\_options="-f /metaldap/slapd-metaldap-local/etc/openldap/slapd.conf -4 -h ldap:/{{ metadir\_server }}:390"

delegate\_to: fw1bcn

run\_once: true

- name: Cambiar configuracion recurso METADIR.

command: /usr/sbin/crm resource param METADIR set cmdline\_options="-f /metaldap/slapd-metaldap/etc/openldap/slapd.conf -4 -h ldap:/{{ metadir\_server }}"

delegate\_to: fw1bcn

run\_once: true

- name: Arrancar recurso METADIRLOCAL.

command: /usr/sbin/crm resource start METADIRLOCAL

delegate\_to: fw1bcn

run\_once: true

- pause: seconds=10

- name: Arrancar recurso METADIR.

command: /usr/sbin/crm resource start METADIR

delegate\_to: fw1bcn

run\_once: true

...

# Módulos de ansible.

En general el uso de ansible se basa en emplear módulos ya predeterminados para realizar tareas. En el apartado anterior ya he utilizado varios de ellos y ahora, siguiendo con un playbook para el cambio de direccionamiento IP en la oficina de Barcelona, voy a hacer otro playbook diferente que me permita cambiar el direccionamiento IP de cimmeria y de la zona crom contenida en dicho servidor.

En este caso, tanto el servidor cimmeria como crom, son sistema Solaris 10. La prueba más rápida es añadir la clave SSH pública de mi máquina central ansible al authorized\_keys de ambas máquinas y probar el módulo ping. Si todo es correcto recibiré la salida pong correspondiente.

root@ubuntuconsolas:/etc/ansible# ansible crom,cimmeria -m ping

crom | SUCCESS => {

"changed": false,

"ping": "pong"

}

cimmeria | SUCCESS => {

"changed": false,

"ping": "pong"

}

Una vez probado que hay comunicación correcta con las máquinas puedo empezar a hacer el playbook. En el caso de estos sistemas sé que hay que cambiar la IP, la máscara de red y el router por defecto, así como el resolv.conf y entradas en los ficheros hosts de ambas. Además habrá que modificar el archivo de configuración de samba y de apache en crom.

Para empezar, todas las variables relativas a la red las tengo definidas en el fichero de grupo BCN que está en /etc/ansible/group\_vars y que puedo ver [aquí](#_/etc/ansible/group_vars/BCN).

Para cambiar el contenido de un archivo de texto, como el /etc/hosts de una máquina Unix/Linux, usaré el módulo lineinfile como se ve en la siguiente descripción de tarea:

tasks:

- name: Copia el archivo resolv.conf correcto

template: src=/etc/ansible/templates/resolv.conf.j2 dest=/tmp/resolv.conf backup=yes

- name: Cambio las entradas necesarias del archivo hosts - DC1.

lineinfile: dest=/tmp/hosts state=present regexp="{{ delegacion }}dc1\." line="{{ dns1\_server }} {{ delegacion }}dc1 {{ delegacion }}dc1.gmv.es" b

ackup=yes

- name: Cambio las entradas necesarias del archivo hosts - DC2.

lineinfile: dest=/tmp/hosts state=present regexp="{{ delegacion }}dc2\." line="{{ dns2\_server }} {{ delegacion }}dc2 {{ delegacion }}dc2.gmv.es" b

ackup=yes

- name: Cambio las entradas necesarias del archivo hosts - METADIR.

lineinfile: dest=/tmp/hosts state=present regexp="metadir{{ delegacion }}\." line="{{ metadir\_server }} metadir{{ delegacion }} metadir{{ delegacion

}}.gmv.es" backup=yes

- name: Cambio las entradas necesarias del archivo hosts - CROM.

lineinfile: dest=/tmp/hosts state=present regexp="{{ cifs\_server }}\." line="{{ cifs\_server\_ip }} {{ cifs\_server }} {{ cifs\_server }}.gmv.es" bac

kup=yes

delegate\_to: crom

run\_once: true

- name: Cambio las entradas necesarias del archivo hosts - CIMMERIA.

lineinfile: dest=/tmp/hosts state=present regexp="cimmeria\." line="{{ cimmeria\_ip }} cimmeria cimmeria.gmv.es" backup=yes

delegate\_to: cimmeria

run\_once: true

- name: Cambio el archivo defaultrouter.

lineinfile: dest=/tmp/defaultrouter regexp="^172.22" line="{{ network\_gateway }}" backup=yes

- name: Cambio la configuracion de apache - CROM.

lineinfile: dest=/tmp/httpd.conf regexp="AuthLDAPURL" line=' AuthLDAPURL "ldap://{{ metadir\_server }}:389/ou=GMV-SES,ou=Usuarios,dc=gmv,dc=es?sAMAcc

ountName?sub?(objectClass=person)"' backup=yes

delegate\_to: crom

run\_once: true

- name: Cambio la configuracion de samba, eliminacion linea password server - CROM.

lineinfile: dest=/tmp/smb.conf regexp="password server" state=absent

delegate\_to: crom

run\_once: true

- name: Cambio la configuracion de samba, linea hosts allow - CROM.

lineinfile: dest=/tmp/smb.conf regexp="hosts allow" line=" hosts allow = 172.18. 192.168.168. 192.168.240. 192.168.60. 192.168.52. 192.168.61. 192.16

8.62. 192.168.63." backup=yes

delegate\_to: crom

run\_once: true

- name: Para la zona solaris CROM.

solaris\_zones: name=CROM state=stopeed

delegate\_to: cimmeria

run\_once: true

- pause: seconds=10

- name: Arranca la zona solaris CROM.

solaris\_zone: name=CROM state=running

delegate\_to: cimmeria

run\_once: true

Con este play solo cambiaré las líneas que quiero, aunque veo que también añade las líneas aunque no estén, es decir, cuando están sustituye por lo que pongo, cuando no están simplemente las añade.

Además también uso el módulo que me permite controlar zonas de Solaris, en este caso lo que hago es parar la zona de Solaris que hay en el servidor cimmeria.

En este mismo playbook voy a rebotar el host físico cimmeria para asegurar que los cambios se aplican correctamente y antes voy a parar la zona CROM para, tras el arranque del servidor, volver a arrancarla. Esto implica que es necesario decirle a ansible que continúe cuando el host físico cimmeria haya vuelto a estar otra vez operativo. Para esto, lo que se recomienda es usar el módulo wait\_for creando unas tareas como las siguientes:

- name: Reinicio de cimmeria.

command: /usr/sbin/shutdown –y –g0 –i6

delegate\_to: cimmeria

run\_once: true

async: 0

poll: 0

ignore\_errors: true

- name: Esperar al arranque del servidor cimmeria.

local\_action: wait\_for host=cimmeria state=started delay=300

# Usando condiciones en playbooks.

Supongamos que quiero realizar una tarea dentro de un playbook solo cuando se cumpla una condición, dada por una variable o estado en el host remoto. Para esto ansible usa la palabra clave when, por ejemplo, con la siguiente tarea crearé un archivo en /tmp del host remoto solo si el interfaz eth2 no tiene dirección ip:

---

- hosts: metadirwar

tasks:

- name: Crear un archivo en si eth2 no tiene direccion IP.

shell: touch /tmp/NOTIENEIP

when: not ansible\_eth3.active

...

Usando when configuraré los interfaces de red de un host, por ejemplo los nodos de un cluster de firewalls de las delegaciones del siguiente modo:

---

- hosts: fw1vls fw2vls

tasks:

- name: Configurar interfaz de red eth2 si no tiene IP - FW1VLS.

command: /sbin/ifconfig eth2 {{ fw1\_heartbeat\_ip }} {{ netmask\_24bit }} up

when: not ansible\_eth2.active

delegate\_to: fw1vls

notify: Modificar fichero interfaces del nodo - FW1VLS.

- name: Configurar interfaz de red eth2 si no tiene IP - FW2VLS.

command: /sbin/ifconfig eth2 {{ fw2\_heartbeat\_ip }} {{ netmask\_24bit }} up

when: not ansible\_eth2.active

delegate\_to: fw2vls

notify: Modificar fichero interfaces del nodo - FW2VLS.

handlers:

- name: Modificar fichero interfaces del nodo - FW1VLS.

delegate\_to: fw1vls

blockinfile:

dest: /tmp/interfaces

backup: yes

state: present

insertafter: EOF

block: |

# HEARTBEAT - FW1VLS

auto eth2

iface eth2 inet static

address {{ fw1\_heartbeat\_ip }}

netmask {{ netmask\_24bit }}

- name: Modificar fichero interfaces del nodo - FW2VLS.

delegate\_to: fw2vls

blockinfile:

dest: /tmp/interfaces

backup: yes

state: present

insertafter: EOF

block: |

# HEARTBEAT - FW2VLS

auto eth2

iface eth2 inet static

address {{ fw2\_heartbeat\_ip }}

netmask {{ netmask\_24bit }}

...

En el playbook anterior actuaré sobre los nodos del firewall para configurar su interfaz de heartbeat, que será eth2 y tendrá las direcciones IP 10.10.250.1 y 10.10.250.2 para cada uno de los nodos. Estas direcciones, dadas como variables, estarán en el fichero de variables para el grupo, que describirá la delegación, al que pertenezcan ambos firewalls y estarán en las variables fw1\_heartbeat\_ip y fw2\_heartbeat\_ip pero esto solo se hace si el interfaz eth2 no está activo, que es lo mismo que no esté configurado. Para controlar este punto de configuración del interfaz eth2 uso la variable ansible\_eth2.active, que ansible obtiene de los facts del host remoto.

Esté playbook usa además unos handlers en los que, si el interfaz no está activo y por tanto sin configurar, incluirán la configuración necesaria en el fichero interfaces de los nodos.